UNIVERSIDAD
DE LA REPUBLICA
URUGUAY

Framework para el análisis de calidad de datos en Data Warehouse

Christian Azziorri, Matías Esmoris, Fernando Kwilman.

Tutoras: Adriana Marotta, Camila Sanz.

Informe de Proyecto de Grado en Ingeniería en Computación
Facultad de Ingeniería - Universidad de la República

Montevideo – Uruguay

Abril de 2022

RESUMEN

Los sistemas de *Data Warehouse* han tomado gran relevancia a nivel de las organizaciones, debido a que son utilizados para el análisis y el proceso de toma de decisiones. Por lo tanto una mala calidad de datos en estos sistemas puede afectar negativamente estos procesos, conduciendo a malas decisiones. En este tipo de sistemas es inevitable sufrir algún problema de calidad durante las diferentes etapas de construcción y utilización de los mismos. Es por esto que es importante contar con una herramienta para evaluar aspectos de la calidad de datos en los sistemas de *Data Warehouse*.

Este proyecto está basado principalmente en la tesis doctoral "Context based Data Quality Rules for Multidimensional Data". En dicha tesis se plantea abordar el problema de evaluar la calidad de datos en *Data Warehouse* como un conjunto de subproblemas: 1) Formalización del *Data Warehouse*, 2) Formalización y definición de contexto, 3) Mecanismo de interacción entre contexto y *Data Warehouse*, 4) Definición y formalización de reglas de evaluación y mejora de calidad de datos, 5) Implementación de la solución, 6) Experimentación con un caso de uso real.

En base a esto, en este proyecto se realiza la implementación de una herramienta que permite definir e instanciar un *Data Warehouse* con sus cubos (modelo multidimensional), siguiendo la formalización e implementación Datalog planteada en la tesis "Context based Data Quality Rules for Multidimensional Data", permitiendo también realizar operaciones de *roll-up* sobre dichos cubos.

Por otra parte, se formalizan métricas de calidad que permiten la evaluación de la calidad de un *Data Warehouse* y se implementan dentro de la herramienta, permitiendo ejecutar las mismas para el *Data Warehouse* definido por el usuario.

Palabras claves:
Data Warehouse, Calidad de datos, Datalog, Sumarizabilidad, Electron, Node.js.

Tabla de contenidos

Capítulo 1

Introducción

En este primer capítulo se presentan las principales características del proyecto, incluyendo el contexto en el cual está basada la realización del mismo. Se plantean también los objetivos que fueron abordados, los resultados esperados y una breve descripción de la estructura del documento.

1.1. Contexto

Este proyecto se enmarca en el grupo de investigación Concepción de Sistemas de Información del Instituto de Computación de la Facultad de Ingeniería de la UdelaR, en las líneas investigación de Sistemas de *Data Warehouse* y Calidad de Datos.

En un contexto en el que cada vez se acumulan más datos, los sistemas de *Data Warehouse* son de mucha importancia en las organizaciones, ya que hacen un uso intensivo de su análisis para el proceso de toma de decisiones, una mala calidad en los datos puede afectar negativamente a este proceso. En estos sistemas es inevitable sufrir algún problema de calidad en el ciclo de vida del *Data Warehouse* (en adelante, DW): cuando se extraen los datos de diversas fuentes heterogéneas, cuando se ejecutan procesos de transformación y carga, cuando los datos se cargan en cubos, o cuando se realizan operaciones de agregación sobre estos cubos. Es por esto que es fundamental contar con herramientas que permitan evaluar y gestionar la calidad de los datos en DW.

Este proyecto esta basado principalmente en la tesis doctoral *"Context based Data Quality Rules for Multidimensional Data"* [31] donde se aborda la calidad de los datos como un concepto multifacético, ya que al tratar de medirla y definirla se deben de tratar muchos aspectos, los cuales se conocen como dimensiones de calidad. La tesis doctoral se enfoca en la resolución del problema de la evaluación de la calidad de datos en DW, tomando en cuenta el contexto de los datos y considerando principalmente las dimensiones de consistencia, exactitud y completitud. La consistencia refiere a la capacidad de los datos de mantener sin contradicciones propiedades asociadas al dominio y la realidad, la exactitud es el grado en que los datos reflejan correctamente la realidad, y la completitud es la capacidad de representar todos los aspectos de la realidad.

Para la resolución del problema planteado en [31] propone resolver la formalización de los conceptos de DW y Contexto, así como la definición y formalización de reglas de evaluación de la calidad sobre el DW. En primer lugar, presenta una especificación formal para un DW genérico basada en el trabajo de Hurtado-Mendelzon [29] y luego propone un conjunto de reglas Datalog para modelarlo. Estas reglas pueden ser utilizadas para

instanciar cualquier DW particular.

En base a esto, surge la necesidad de implementar una herramienta que permita instanciar las formalizaciones planteadas para un DW cualquiera. Otra necesidad a resolver es la definición y formalización de reglas de evaluación de la calidad, que puedan ser ejecutadas en la herramienta para el DW instanciado.

1.2. Objetivos

El objetivo principal del proyecto es la implementación de una herramienta que permita al usuario a partir de un DW, definir los cubos correspondientes, navegar sobre los mismos y obtener los resultados de ejecutar métricas de calidad sobre el DW. A continuación se mencionan los objetivos específicos:

- Definición y formalización de métricas de calidad de datos que permitan la evaluación de la calidad de un DW.
- Desarrollar una herramienta para instanciar cualquier DW basándose en la formalización e implementación en Datalog planteada en la tesis *"Context based Data Quality Rules for Multidimensional Data"* [31]. La herramienta debe permitir:
  - Navegar sobre los cubos definidos, obteniendo el código pyDatalog [25] asociado a las operaciones realizadas.
  - Ejecutar las métricas de calidad de datos formalizadas sobre el DW definido, obteniendo el resultado y el código pyDatalog asociado.

1.3. Resultados esperados

Una vez culminado este proyecto se espera contar con una herramienta que permita al usuario obtener resultados de evaluar métricas de calidad sobre un DW. Dicha herramienta debe permitir al usuario definir y establecer una conexión a una base de datos, y contar con una interfaz para definir el DW y los cubos asociados.

Luego de definidos el DW y los cubos, la herramienta debe permitir al usuario navegarlos mediante operaciones de agregación, obteniendo los resultados correspondientes y el código pyDatalog generado para realizar dichas operaciones.

Por último, la herramienta debe permitir al usuario ejecutar métricas de calidad (definidas previamente) sobre el DW, obteniendo los resultados y el código pyDatalog generado para realizar dichos cálculos.

1.4. Estructura del documento

El documento se encuentra estructurado en 7 capítulos:

- En el capítulo 2, "Marco téorico" se presentan conceptos básicos de sistemas de DW, calidad de datos y lenguaje Datalog, necesarios para un mejor entendimiento del abordaje del proyecto.
- En el capitulo 3, "Métricas de calidad propuestas" se mencionan los trabajos relacionados en los cuales esta basado este proyecto, y se propone la formalización de métricas de calidad que serán utilizadas en la implementación de la herramienta.
- En el capítulo 4, "Análisis y diseño del framework" se presenta el análisis de los requerimientos y los casos de uso necesarios para implementar la herramienta, junto con la arquitectura diseñada y la interfaz de usuario.
- En el capítulo 5, "Implementación" se describen todos los aspectos que fueron tomados en cuenta para la implementación de la herramienta, tales como las tecnologías utilizadas, bases de datos e implementación de las consultas pyDatalog.
- En el capítulo 6, "Experimentación" se detallan las pruebas realizadas para corroborar el correcto funcionamiento de la herramienta.
- En el capítulo 7, "Conclusiones y trabajos a futuro" se presentan las conclusiones obtenidas del desarrollo del proyecto en base a los objetivos planteados, así como trabajos para ser abordados a futuro y que quedaron por fuera del alcance.

Adicionalmente este proyecto consta de tres anexos:

- Anexo A, con casos de uso generados en la etapa de análisis de la herramienta.
- Anexo B, con un ejemplo del código pyDatalog generado para una consulta.
- Anexo C, que contiene el manual para configurar e instalar la herramienta, y el manual de usuario para las funcionalidades de la herramienta.

Capítulo 2

Marco teórico

En este capítulo se presentan una serie de definiciones y conceptos necesarios para el abordaje de este proyecto y un mejor entendimiento de lo que resta del documento, asumiendo que el lector posee conocimientos básicos sobre bases de datos. En particular se abordan los temas DW, Datalog y Calidad de datos.

2.1. Sistemas de Data Warehouse

En esta sección se detallan conceptos básicos sobre los sistemas de DW y la definición del modelo multidimensional en el que se basan los mismos.

2.1.1. Introducción a los sistemas de *Data Warehouse*

Dentro de las organizaciones, surge la necesidad de integrar datos desde múltiples sistemas operacionales con el objetivo de realizar análisis de datos sofisticados para enriquecer los procesos de toma de decisiones.

Las bases de datos transaccionales no satisfacen los requerimientos para realizar estos análisis, ya que están diseñadas para soportar las operaciones diarias de una organización y su principal preocupación es garantizar un acceso rápido y simultáneo a los datos. Esto requiere capacidades de procesamiento de transacciones y control de concurrencia. Esto hace que no tengan el rendimiento esperado cuando se ejecutan consultas complejas que involucran la unión de muchas tablas relacionales [33].

En base a esta necesidad se propusieron los sistemas de DW, que son sistemas informáticos capaces de ofrecer información para toma de decisiones, y cuya pieza principal es un DW. Según Inmon [30], un DW se puede definir como un conjunto de datos orientados a temas, integrados, no volátiles e históricos, organizados para soportar un proceso de toma de decisiones. Esta definición engloba varias características destacadas de los DW:

- **Orientado a temas:** los datos se organizan en torno a uno o varios temas de análisis, de acuerdo a los requisitos de la organización.

- **Integrado:** el contenido del DW, es el resultado de la integración de datos de varios sistemas operacionales y externos. Esto implica que a partir de fuentes heterogéneas, se almacenen los datos de forma homogénea.

- **No volátil:** los datos deben ser lo suficientemente estables como para permitir análisis de larga duración sin que cambien durante el mismo.

- **Histórico:** se deben manejar los datos con su referencia temporal.

La arquitectura de un sistema de DW según [33] consta de varios componentes: un conjunto de fuentes de datos diversas, un conjunto de herramientas ETL, del inglés *Extraction, Transformation and Loading* (Extracción, Transformación y Carga) utilizadas para extraer datos de diversas fuentes, realizar transformaciones y cargar la información en el DW, la base de datos destinada al DW, y un conjunto de herramientas de análisis: herramientas OLAP para análisis multidimensional, que permiten la construcción, navegación y consulta de cubos, herramientas de generación de reportes y estadísticas, y herramientas de minería de datos. En la figura 2.1 se muestran estos componentes y el flujo de los datos.

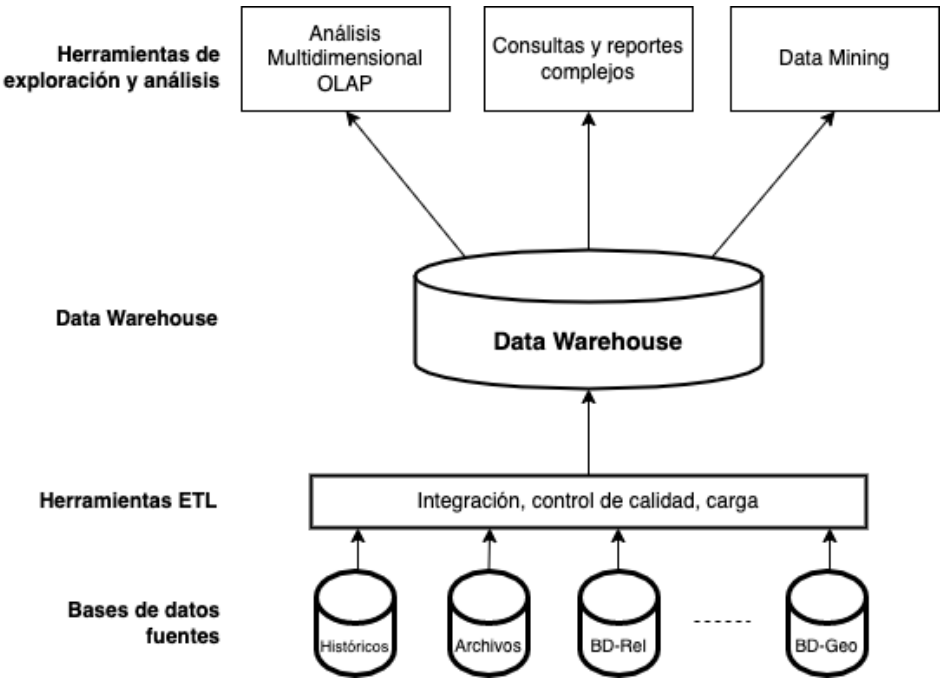


Figura 2.1: Arquitectura de un *DW*

En este proyecto se trabaja únicamente sobre el propio DW, es decir que no se aplican procesos ETL, ni se utilizan herramientas de análisis. En particular se trabaja sobre el modelo multidimensional de un DW, que será definido en la sección 2.1.2.

2.1.2. Modelo Multidimensional

Los sistemas de DW se basan en el modelo multidimensional, donde los datos se representan como matrices de n-dimensiones. Cada una de estas matrices se denomina

Cubo, que tienen en sus ejes los criterios de análisis, llamados **Dimensiones** y en cada celda se tiene el valor a analizar, llamado **Medida**.

Una dimensión es una colección de información relacionada con una medida específica. Un DW organiza atributos descriptivos como columnas en las tablas de dimensión. Por ejemplo, una dimensión de Cliente podría incluir atributos como el nombre, genero, fecha de nacimiento, etc.

Las dimensiones se estructuran en **Jerarquías** de agregación, donde a cada nivel de agregación de una jerarquía se le denomina **Nivel**. Toda dimensión tiene por lo menos una jerarquía con un único nivel y puede tener más de una jerarquía [27].

Caso de ejemplo

A modo de ejemplo proponemos un DW que contiene información ficticia sobre ventas de productos en cadenas de supermercados. Interesa registrar los productos que se vendieron para cada mes y cadena de supermercados. Para presentar el ejemplo utilizaremos un **esquema conceptual de dimensiones y medidas** y un esquema lógico denominado **esquema estrella**.

El esquema conceptual de dimensiones y medidas es una forma de representar las dimensiones, sus jerarquías y las medidas. Cada jerarquía es representada como un camino entre el nivel más bajo y el nivel *All*. En la figura 2.2 se presenta el esquema conceptual de dimensiones y medidas utilizadas para el caso de ejemplo

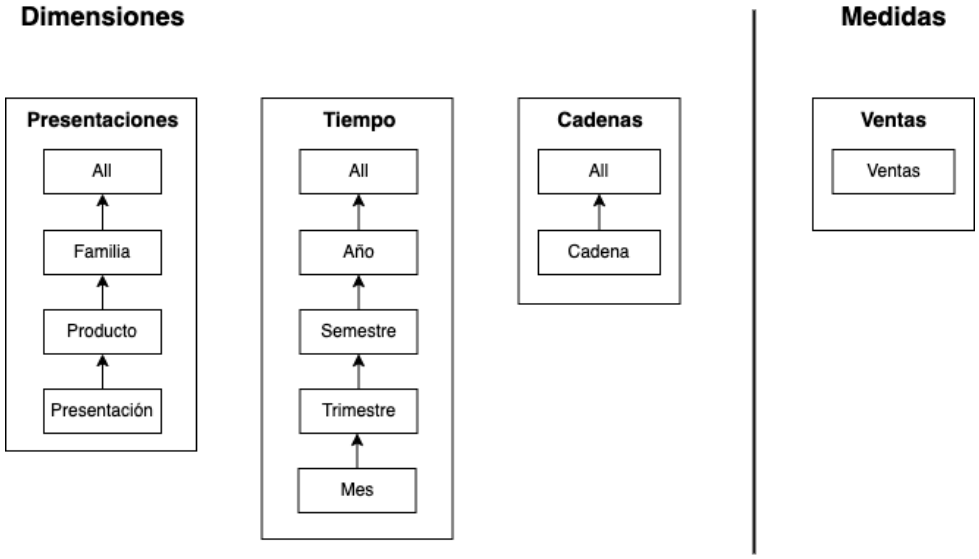


Figura 2.2: Esquema conceptual de dimensiones y medidas

La dimensión **Cadenas** hace referencia a cadenas de supermercados y consta de una única jerarquía con un nivel: Cadena. La dimensión **Presentaciones** hace referencia a los productos de los supermercados y consta de una jerarquía con tres niveles: Presentación, Producto y Familia. La dimensión **Tiempo** agrega un marco temporal y consta de una

jerarquía con cuatro niveles: Mes, Trimestre, Semestre y Año. La medida **Ventas** refiere a la cantidad de ventas por presentación, cadena y tiempo. En la figura 2.3 se puede observar un ejemplo de un cubo tridimensional para estas dimensiones y medida.

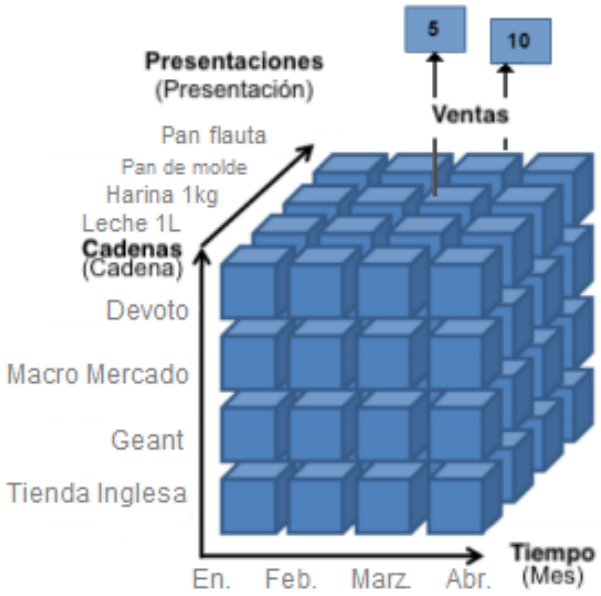


Figura 2.3: Cubo de cantidad de ventas.

Para presentar un esquema lógico del ejemplo se utiliza el **esquema estrella**, el cual es un tipo de esquema de base de datos relacional ampliamente utilizado en el diseño de DW. Este esquema consta de una tabla de hechos central rodeada de tablas de dimensiones.

Las tablas de dimensiones son tablas desnormalizadas y con jerarquías embebidas. Las tablas de hechos deben encontrarse unidas a todas las de dimensión en relaciones 1:N y su clave primaria es la concatenación de las claves primarias de todas las dimensiones.

En la figura 2.4 se presenta el esquema estrella para el ejemplo. La tabla de hechos es Ventas y las tablas de dimensiones son Presentaciones, Tiempo y Cadenas. Las ramas en los enlaces entre las tablas de dimensión con la de hechos representan la relación 1:N.

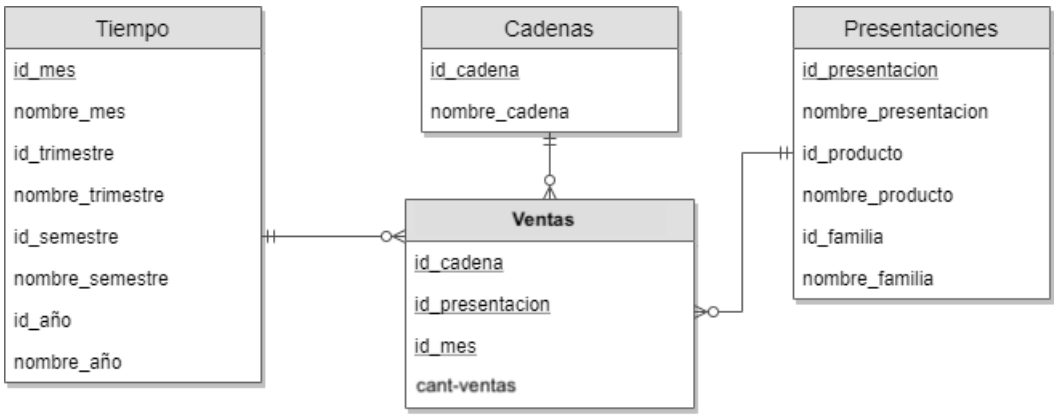


Figura 2.4: Esquema estrella para el caso de ejemplo

Operaciones *OLAP*

Son operaciones que pueden ejecutarse sobre los cubos y permiten visualizar los datos desde distintas perspectivas y niveles de jerarquía en las dimensiones [33].

Las operaciones más utilizadas son las siguientes:

- ***Roll-Up***: La operación de *roll-up* agrega medidas a lo largo de una jerarquía de dimensiones (mediante una función de agregación) para obtener las medidas con una granularidad más gruesa.
- ***Drill-Up* y *Drill-Down***: Estas operaciones permiten pasar de un nivel más general a un nivel mas detallado dentro de una jerarquía (*drill-down*), o a la inversa, de un nivel más detallado a un nivel más general (*drill-up*).
- ***Slice***: La operación de *slice* remueve una dimensión en un cubo, obteniendo un cubo de n-1 dimensiones, a partir de un cubo de n dimensiones. Para la dimensión a quitar, se debe fijar un valor único en el nivel.
- ***Dice***: La operación *dice* mantiene las celdas de un cubo que cumplen con cierta condición booleana sobre niveles de dimensión, atributos y medida.
- ***Pivot***: La operación de *pivot* (o rotación) permite rotar los ejes de un cubo proporcionando un orden de visualización alternativo de los datos.
- ***Drill-Across***: Esta operación combina celdas de dos cubos de datos que tienen el mismo esquema e instancia. Análoga a la operación *join* en el álgebra relacional.

En este proyecto se utilizará únicamente la operación *roll-up*, por lo que profundizaremos únicamente en esta operación.

Operación Roll-Up

La operación *roll-up* calcula las medidas en función de agrupamientos. Para poder utilizar correctamente esta operación debe especificarse cuál es la operación que calcula

el nuevo valor de la medida, la cual puede ser suma, promedio, etc. En la figura 2.5 se ilustra el resultado de aplicar la operación *roll-up* sobre la dimensión Presentaciones.

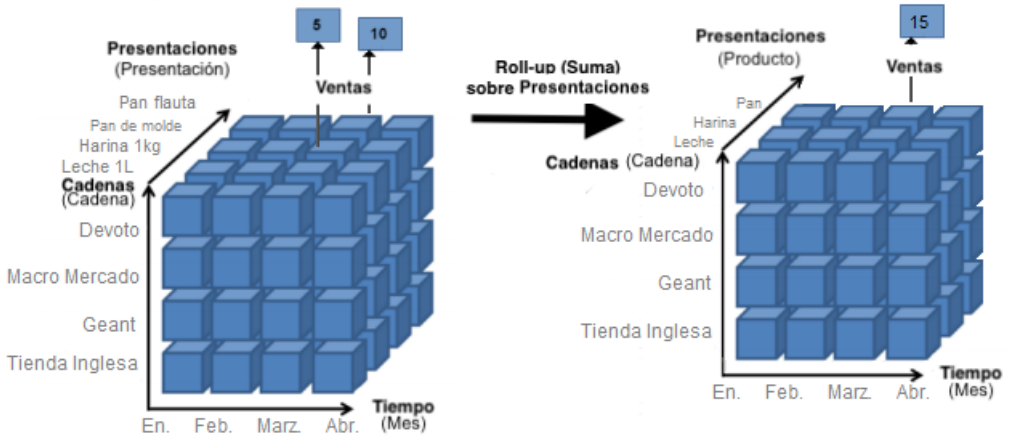


Figura 2.5: *Roll-Up*

Para realizar este *roll-up* se utiliza la operación de suma. Podemos ver en la figura 2.5, como luego de realizada la operación *roll-up*, el valor de la medida sobre las Presentaciones de Pan flauta y Pan de molde, con valores de 10 y 5 respectivamente, son sumados en el cubo de la derecha a 15 luego de agruparse como productos de Pan.

2.2. Lenguaje Datalog

Datalog es un lenguaje de programación lógica declarativo que se introdujo como lenguaje de consulta para bases de datos deductivas a finales de los años 70 y en los últimos años ha encontrado variedad de nuevas aplicaciones, en particular: integración de datos y extracción de información.

La sintaxis de Datalog proviene fuertemente de la programación lógica y los caracteres de una entrada Datalog se recopilan en **tokens**, existiendo cuatro clases: signos de puntuación, variables, identificadores y cadenas [1].

- **Signos de puntuación:**

  `'(', ',', ')', '=', ':-', '.', '~', '?', '"'`

- **Variables:** Secuencia de letras mayúsculas y/o minúsculas, dígitos y guión bajo. Una variable debe comenzar con una letra mayúscula.

- **Identificadores:** Secuencia de caracteres que no contiene ninguno de los tokens de puntuación y espacio. No deben comenzar con una letra mayúscula.

- **Cadenas:** Secuencia de caracteres entre comillas dobles.

Un programa Datalog se compone de un conjunto de cláusulas de la forma: *head :- body*. Donde el *head* es un literal (átomos o negaciones de átomos) que se define como $p(t1, t2, ..., tn)$ con p símbolo de predicado (un identificador o una cadena) y $t1, t2, ..., tn$ términos que pueden ser variables o constantes. El *body* es una lista de literales separados por coma $(B1, B2, ..., Bn)$ el cual puede ser vacío, en este caso la cláusula se conoce como un hecho, de lo contrario se denomina regla. Opcionalmente un programa puede incluir una consulta, que se define con un literal seguido de un signo de interrogación [32].

Una cláusula se lee como *head* $\leftarrow$ *body*, por ejemplo, el significado de la cláusula $H \leftarrow B_1, ...B_n$ es: Si $B_1, B2, \ldots y\, Bn$ son ciertos, entonces H es cierto.

```
1    arista(a, b).
2    arista(b, c).
3    arista(c, d).
4    arista(d, a).
5    camino(X, Y) :- arista(X, Y).
6    camino(X, Y) :- camino(X, Z), camino(Z, Y).
7    camino(X, Y)?
```

Listing 2.1: Ejemplos de programa Datalog

En las lineas 1, 2, 3 y 4 del ejemplo 2.1 el *body* es vacío, por lo que se tiene un hecho donde `arista` es un predicado, mientras que `a`, `b`, `c` y `d` son términos (constantes). Las lineas 5 y 6 están conformadas por reglas donde `camino` y `arista` son predicados, mientras que `X`, `Y` y `Z` son términos (variables). En la linea 7 se tiene una consulta donde `camino` es predicado y `X` e `Y` son términos (variables).

En Datalog los predicados son clasificados como intensionales o extensionales:

- **Predicados extensionales:** son predicados cuyos hechos verdaderos son almacenados en la base de datos. En el ejemplo 2.1 arista es un predicado extensional.
- **Predicados intensionales:** son predicados que están definidos a partir de reglas. En el ejemplo 2.1 camino es un predicado intensional.

Los predicados que aparezcan en el *head* de una o más reglas deben ser intensionales, mientras que los predicados que aparezcan en el *body* pueden ser intensionales o extensionales, y únicamente pueden pertenecer a uno de estos.

Al ejecutarse el programa Datalog del ejemplo 2.1, primero se procesan los literales declarados en las lineas 1, 2, 3 y 4, luego las reglas de las lineas 5 y 6 empezando de arriba para abajo. Al llegar a la línea 7 se ejecuta la consulta `camino(X, Y)`, que significa "todos los nodos que tengan camino entre ellos" y entonces se consideran todas las posibles sustituciones de las constantes `a`, `b`, `c` y `d` por las variables `X` e `Y` de la regla `camino(X, Y)`, probando las reglas con este head en el orden en que fueron procesadas y devolviendo los pares de constantes que hagan que en el body al menos una de las reglas sea verdadera.

PyDatalog

Es una biblioteca de Python, la cual agrega el paradigma de la programación lógica al lenguaje. Puede ser utilizado para simular comportamiento inteligente, consultar conjuntos complejos de información relacionada (por ejemplo en la integración de datos o el procesamiento de lenguaje natural) o realizar algoritmos recursivos. La sintaxis específica de pyDatalog se encuentra en la documentación oficial [25].

2.3. Calidad de datos

Los datos se han convertido en uno de los activos más importantes dentro de las organizaciones, utilizados principalmente como materia prima para la toma de decisiones. La presencia de problemas de calidad de datos, entre ellos el uso de datos inexactos, incompletos o desactualizados puede ocasionar grandes problemas, debido a esto, varias instituciones y organizaciones internacionales reconocen la relevancia de la calidad de los datos tanto en los procesos de toma de decisiones como en los operativos [26].

En general, se reduce el concepto de calidad de datos a "exactitud de datos" , sin embargo es un concepto que abarca múltiples dimensiones y que depende del contexto y el consumidor. Los consumidores quieren que los datos sean relevantes para su uso, correctos, consistentes, actualizados, en el formato adecuado, y de fácil acceso [26].

Algunos problemas típicos de calidad de datos son: datos incorrectos, datos inconsistentes con la realidad, datos desactualizados, datos incompletos, datos difíciles de acceder, entre otros. Estos problemas podrían generarse en diferentes etapas como la producción de los datos, el procesamiento, el almacenamiento o la utilización de los mismos [26].

2.3.1. Dimensiones de calidad

La calidad es caracterizada mediante múltiples dimensiones que ayudan a calificar los datos dentro de un Sistema de Información (de aquí en adelante SI). Cada dimensión captura un aspecto específico respecto a la calidad de los datos.

Se puede definir una jerarquía de conceptos de calidad, donde las **dimensiones** representan las facetas de la calidad a alto nivel. Cada dimensión está compuesta por un conjunto de **factores**, los cuales representan aspectos particulares de la dimensión. Cada uno de los factores puede medirse con diferentes **métricas de calidad**, las cuales son instrumentos que indican la forma de medir un factor y se definen con: nombre, descripción (qué se mide), unidades de medición (por ejemplo, tiempo de respuesta en ms, volumen en GB, valor entre 0 y 1) y granularidad de la medida (celda, tupla, columna, tabla, grupo de tablas, base de datos). Por último se tienen los **métodos de medición**, que son formas de implementar una métrica brindando un valor cuantitativo. Cada métrica puede ser implementada por varios métodos de medición.

Algunas de las dimensiones más estudiadas son: exactitud, completitud, frescura, consistencia y unicidad. Se presentan a continuación ejemplos de dimensiones de calidad (completitud y consistencia) junto con su jerarquía de conceptos:

- **Completitud:** la completitud busca medir hasta qué punto el SI representa el mundo real correspondiente, es decir, que porción de la información de interés se tiene. Abarca aspectos extensionales (cantidad de entidades/estados de la realidad representados) e intencionales (cantidad de datos sobre cada entidad/estado). En la tabla 2.1 se presenta un ejemplo de métrica de calidad para esta dimensión.

Dimensión: Completitud
Factor: Densidad
Métrica: Densidad-Grado
Descripción Grado de densidad de una columna.
Unidad de medida: [0..1]
Granularidad: Columna
Método de medición: Contar valores nulos de una columna y dividirlo por la cantidad de filas.

Tabla 2.1: Métrica de calidad para dimensión Completitud

  El factor de **densidad** refiere a la cantidad de información que se tiene y que falta sobre las entidades del SI. Se caracteriza por la presencia de los valores nulos.

- **Consistencia:** esta dimensión captura la satisfacción de reglas semánticas definidas sobre los datos. Interesa medir qué tan bien se satisfacen las reglas de integridad, y en base a esto, se definen tres factores de consistencia:

  - **Integridad de dominio:** refiere a la satisfacción de reglas sobre el contenido de un atributo (por ejemplo, edad entre 0 y 120 años).
  - **Integridad intra-relación:** refiere a la satisfacción de reglas entre uno o varios atributos de una misma relación. Las reglas más típicas son: dependencias de clave y unicidad, dependencias funcionales, dependencias de valores y expresiones condicionales.
  - **Integridad inter-relación:** refiere a la satisfacción de reglas entre atributos de varias relaciones. Las reglas más típicas son las dependencias de inclusión (clave foránea, integridad referencial).

  En la tabla 2.2 se presenta un ejemplo de métrica de calidad para esta dimensión.

Dimensión: Consistencia
Factor: Integridad de intra-relacion
Métrica: Columnas N:1
Descripción: Indica si dos columnas de una relación son N:1.
Unidad de medida: 0,1
Granularidad: Par de columnas
Método de medición: Dado una relación y dos columnas i,j se buscan tuplas en la relación con el mismo valor en la columna i y distinto valor en la columna j. Si existe alguna se devuelve 0, sino 1.

Tabla 2.2: Métrica de calidad para dimensión Consistencia

2.3.2. Calidad en Data Warehouse

Cada medida definida en un cubo debe tener asociada al menos una función de agregación, que permite combinar varios valores de medida en uno solo. La agregación de medidas tiene lugar cuando se cambia el nivel de detalle en el que se visualizan los datos de un cubo, al cambiar la granularidad del mismo [33].

Siguiendo con el ejemplo de la sección 2.1.2, si consideramos el cubo de granularidad (Cadena, Presentación, Mes), para cambiar la granularidad en la jerarquía de la dimensión Tiempo del nivel Mes a Trimestre, la cantidad de ventas para cada presentación y cadena en el mismo trimestre se deben agregar usando, por ejemplo, la operación de suma.

Una propiedad que interesa que cumplan los DW es la **sumarizabilidad**, que refiere a la correcta agregación de medidas de un cubo a lo largo de las jerarquías de las dimensiones, para obtener resultados de agregación consistentes. Según Hurtado-Mendelzon [28] para garantizar sumarizabilidad, una dimensión debe satisfacer las siguientes condiciones sobre su integridad:

Disyunción de instancias (Jerarquía N:1): la agrupación de instancias en un nivel con respecto a su nivel superior debe resultar en subconjuntos disjuntos. Por ejemplo, en la jerarquía de la dimensión Presentaciones de la figura 2.2, un producto no puede pertenecer a dos familias, de lo contrario la cantidad de ventas de ese producto se contarían dos veces al agrupar, una para cada familia. Supongamos los siguientes datos en la tabla de dimensión Presentaciones (tabla 2.3) y en la tabla de hechos Ventas (tabla 2.4).

Presentaciones		
id_presentacion	id_producto	id_familia
1	1	1
2	1	1
3	2	2
4	2	1

Tabla 2.3: Relación Presentaciones

Ventas			
id_cadena	id_presentacion	id_mes	ventas
1	1	202101	15
1	2	202101	10
1	3	202101	20
1	4	202101	20

Tabla 2.4: Relación Ventas

En la figura 2.6 podemos ver cómo se agrupan los distintos niveles de la dimensión Presentaciones, al aplicar operaciones *roll-up* sobre la jerarquía de la dimensión Presentaciones de la figura 2.2. Luego de realizar la primer operación *roll-up* agrupando por producto, podemos deducir que los valores de las medidas resultantes son 25 (15 + 10) para el id_producto = 1 y 40 (20 + 20) para el id_producto = 2. Al aplicar la segunda operación *roll-up* agrupando por familia, los valores de la medida son: 65 para el id_familia = 1 y 40 para el id_familia = 2. Este resultado es incorrecto ya que al agrupar por familia, el valor de la medida para las presentaciones con id_producto=2 se está sumando tanto para el id_familia = 1 como para el id_familia = 2.

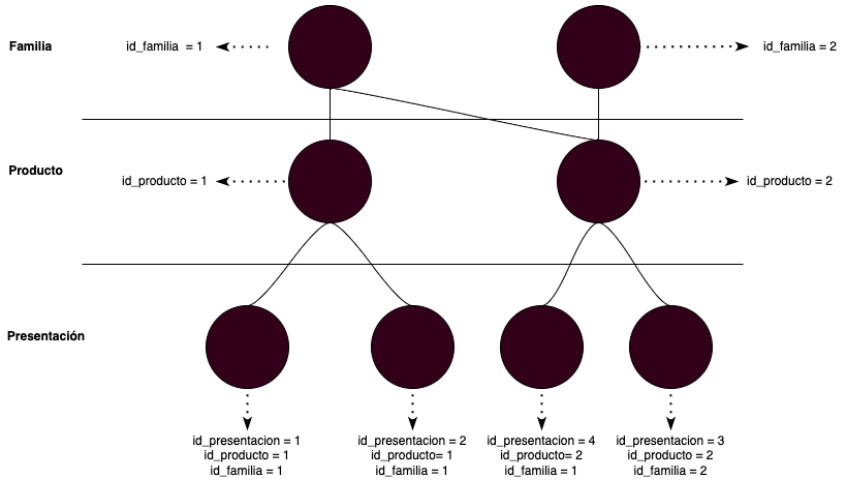


Figura 2.6: Grafo de jerarquía en dimensión Presentaciones

Completitud (Totalidad): todas las instancias deben incluirse en la jerarquía y cada instancia debe estar relacionada con un elemento del siguiente nivel. Por ejemplo, en la jerarquía de la dimensión Tiempo de la figura 2.2, las instancias deben contener todos los meses de interés y cada mes debe tener un trimestre correspondiente. De no cumplirse esta condición, la agregación de mes a trimestre sería incorrecta, ya que habría meses para los que no se contabilizarían las cantidades en canasta. Supongamos los siguientes datos en la tabla de dimensión Tiempo (tabla 2.5) y en la tabla de hechos Ventas (tabla 2.6).

Tiempo			
id_mes	id_trimestre	id_semestre	id_año
202101	20211	20211	2021
202102	-	20211	2021

Tabla 2.5: Relación Tiempo

Ventas			
id_cadena	id_presentacion	id_mes	ventas
1	1	202101	5
1	1	202102	25

Tabla 2.6: Relación Ventas

En la figura 2.7 podemos ver como al intentar agrupar los niveles de la dimensión Tiempo pasando del nivel Mes a Trimestre, se pierde el valor de la medida para la tupla con id_mes = 202102, al no poder agruparlo por id_trimestre.

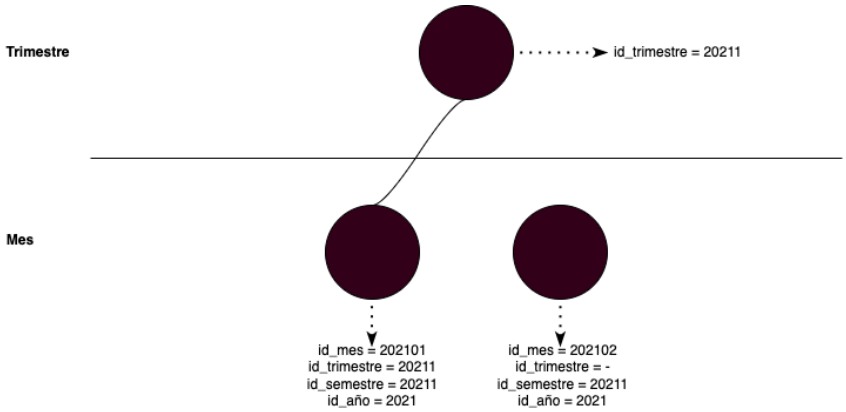


Figura 2.7: Grafo de jerarquía en Dimensión Tiempo

Exactitud: refiere al uso correcto de las funciones de agregación (suma, promedio o ambas según la dimensión, máximo, mínimo) de acuerdo al tipo de la medida. Las medidas se pueden clasificar de la siguiente manera:

- **Aditivas:** se pueden agrupar a lo largo de todas las dimensiones utilizando la operación de suma.
- **Semi-aditivas:** se pueden agrupar mediante la suma en algunas dimensiones, pero no en todas. Se podría, por ejemplo agregar, utilizando el promedio a lo largo de las dimensiones no-aditivas, y la suma en las otras dimensiones.
- **No aditivas:** no se pueden agrupar mediante la suma en ninguna dimensión, se podría por ejemplo, utilizar el promedio u otras como el mínimo o el máximo.

El problema de la sumarizabilidad

El problema de la sumarizabilidad puede darse si al menos una dimensión es inconsistente con respecto a algunas de las condiciones sobre su integridad: disyunción de instancias, completitud y exactitud. Actualmente este problema no fue planteado en base a los términos de dimensiones y métricas de calidad mencionados en la sección 2.3.1.

Para ejemplificar el problema, vamos a utilizar el DW definido en la sección 2.1.2. En este caso, la dimensión Tiempo tiene una jerarquía A, compuesta por los niveles: Mes, Trimestre, Semestre, Año. Suponiendo otra jerarquía B en la dimensión Tiempo con niveles: Mes, Bimestre, Semestre y Año, con los datos de la tabla 2.7 para la tabla de dimensión Tiempo y los datos de la tabla 2.8 para la tabla de hechos Ventas.

Tiempo				
id_mes	id_bimestre	id_trimestre	id_semestre	id_año
202101	20211	20211	20211	2021
202102	20211	-	20211	2021
202103	20212	20211	20211	2021

Tabla 2.7: Relación Tiempo

Ventas			
id_cadena	id_presentacion	id_mes	Ventas
1	1	202101	5
1	1	202102	25
1	1	202103	10

Tabla 2.8: Relación Ventas

Si se realiza la operación de *roll-up* sobre la dimensión Tiempo de Mes a Bimestre, y de Bimestre a Semestre con la operación de suma y sobre los datos contenidos en 2.8, el valor de la medida cant_en_canasta para el id_semestre=20211 es 40. Mientras que si se realiza la operación de *roll-up* sobre la dimensión Tiempo de Mes a Trimestre, y de Trimestre a Semestre, el resultado es 15.

Por lo tanto obtenemos distinto resultado de la medida para la granularidad (Presentacion, Cadena, Semestre) dependiendo si se opera sobre la jerarquía A o la B. Esto se debe que la dimensión Tiempo en su jerarquía B no cumple con la condición de completitud, ya que para el id_mes=202102, el valor del siguiente nivel id_trimestre es nulo.

2.4. Antecedentes

Como se mencionó en la sección 1.1, este proyecto esta basado en parte en el trabajo de investigación realizado por Camila Sanz en el marco de su tesis doctoral [31], la cual está siendo desarrollada en concordancia con los lineamientos de investigación del grupo de Concepción de Sistemas de información del Instituto de Computación, Facultad de Ingeniería, Universidad de la República.

La motivación de dicho trabajo se centra en la importancia de la calidad de los datos para DW considerando el contexto de los datos. El análisis multidimensional que proporcionan los sistemas DW es un recurso fundamental para la toma de decisiones a nivel organizativo. Como los datos pueden verse afectados por el resultado de múltiples transformaciones y provenir de diferentes fuentes heterogéneas, es probable que se generen problemas de calidad de datos que afecten negativamente a la toma de decisiones.

Como consecuencia, la calidad de datos en sistemas de DW es de mucha importancia, y no solo implica la gestión de la calidad sobre los atributos de los datos, sino también sobre el resultado de aplicar operaciones multidimensionales.

El objetivo principal del trabajo presentado en [31] es obtener un conjunto de reglas formales que permita al usuario evaluar y mejorar la precisión, la consistencia y la completitud de un DW en particular, teniendo en cuenta el contexto de los datos. Como se busca una solución formal que permita la instanciación para cualquier problema particular, se plantean objetivos en dirección a eso. En particular se propone obtener una formalización del DW, basada en la formalización de Hurtado-Mendelzon [29], una definición formal del contexto de los datos, un mecanismo de interacción entre el contexto de los datos y el DW y finalmente un conjunto de reglas de evaluación y mejora que consideren el contexto, para las dimensiones de calidad exactitud, completitud y consistencia.

Nuestro trabajo se enmarca en lo planteado anteriormente y propone resolver algunos de los problemas mencionados: dado un DW, generar su definición en Datalog, permitir la realización de operaciones de *roll-up* y generar reglas de calidad de datos, independientes del contexto, en ambos casos utilizando Datalog.

2.4.1. Especificación formal del *Data Warehouse*

Para especificar formalmente un DW, es necesario definir tres aspectos principales: dimensiones, espacio multidimensional y cubos. Estos aspectos que se presentan son tomados de la tesis doctoral *"Context based Data Quality Rules for Multidimensional Data"* [31], que están basados en las definiciones de Hurtado-Mendelzon [29]

- **Dimensiones:** Un esquema de dimensión se define como una tupla $\mathcal{S} = \langle \mathcal{C}, \nearrow \rangle$ donde $\mathcal{C} = \{c_1, \ldots, c_n\}$ es un conjunto finito de categorías y $\nearrow$: $\mathcal{C} \times \mathcal{C}$ es una relación binaria e irreflexiva sobre los elementos de $\mathcal{C}$ (siendo $\nearrow^*$ su clausura transitiva). Se distinguen dos elementos *All* y c_b llamado categoría base que cumplen las siguientes

propiedades respectivamente:

- Toda categoría c está relacionada por $\nearrow^*$ con All: $(\forall c \in \mathcal{C})(c \nearrow^* All)$
- La categoría base está relacionada por $\nearrow^*$ con toda categoría c: $(\forall c \in \mathcal{C})\ (c_b \nearrow^* c)$.

Una instancia de dimensión para un esquema $\mathcal{S}$ es una tupla $\mathcal{D} = \langle \mathcal{U}, <, mem \rangle$, donde $\mathcal{U}$ es un conjunto finito no vacío de miembros, $\mathcal{U} \subseteq \bigcup_{i=1}^{n} Dom(c_i)$ y para cada categoría $Dom(c_i)$ es el conjunto posible de valores de c_i.

La relacion $<$ es binaria e irreflexiva sobre los elementos de $\mathcal{U}$ (siendo $<^*$ su clausura transitiva), esta relación debe ser consistente con la relación $\nearrow$.

Por último, $mem : \mathcal{U} \to \mathcal{C}$ es una función total que para cada valor de $\mathcal{U}$ devuelve la categoría a la que pertenece. Existe un miembro all tal que $mem(all) = All$.

- **Espacio multidimensional:** Un esquema de espacio multidimensional $\mathfrak{G} : \mathcal{S}[1], \ldots, \mathcal{S}[n]$ es una lista de esquemas de dimensión.

  Un esquema de granularidad, para un esquema multidimensional $\mathfrak{G}$, es una lista de categorías $G : G[1], ..., G[n]$ donde cada $G[i]$ es una categoría de la dimensión $\mathcal{S}[i]$.

  Para simplificar los predicados de Datalog que serán presentados en la sección 2.4.2, se definen los siguientes conceptos:

  - Una **granularidad base** G_b es un esquema de granularidad, donde cada categoría es la categoria base de la dimensión correspondiente.
  - Dos esquemas de granularidad G_1 y G_2 para el mismo esquema multidimensional $\mathfrak{G}$, se denominan **granularidades adyacentes** sobre la categoría i, si y solo si, difieren en la dimensión i, y $G_1[i] \nearrow^* G_2[i]$.

  Una **instancia de espacio multidimensional** para un esquema $\mathfrak{G}$, es una lista de instancias de dimensión $\mathfrak{D} : \mathcal{D}[1], \ldots, \mathcal{D}[n]$ donde cada $\mathcal{D}[i] = \langle \mathcal{U}_i, <_i, mem_i \rangle$ es una instancia para el esquema de dimensión $\mathcal{S}[i]$

- **Cubos:** Un esquema de cubo base es una tupla $cubeBase = \langle G_b, m \rangle$, donde G_b es una granularidad base para un esquema multidimensional $\mathfrak{G}$ y m una medida.

  Para introducir el concepto de *cubeView*, que refleja la agregación entre dos categorías de una dimensión, como en un modelo multidimensional, se van a utilizar operadores del álgebra relacional junto con la función de agregación.

  Dado un esquema de espacio multidimensional $\mathfrak{G}$, la siguiente expresión representa la agregación de un esquema de cubo base, $cubeBase = \langle G_b, m \rangle$, a un esquema de granularidad G sobre la instancia de espacio multidimensional $\mathfrak{D}$, utilizando la función de agregación SUM sobre la medida m:

$$cubeView_{G,SUM(m)}(\mathfrak{D}, cubeBase) =$$
$$\Pi_{G,SUM(m)}(cubeBase \bowtie R^{G[1]}_{G_b[1]}(\mathcal{D}[1]) \bowtie ... \bowtie R^{G[n]}_{G_b[n]}(\mathcal{D}[n]))$$

Siendo $R_{G_1}^{G_2}(\mathcal{D}) = \{(e_1, e_2) | mem(e_1) = c_1 \wedge mem(e_2) = c_2 \wedge e_1 <^* e_2\}$ la relación *roll-up* entre las categorías c_1 y c_2 pertenecientes a las dimensiones de $\mathcal{D}$.

Dado que es necesario agregar sobre múltiples dimensiones y no solo desde un esquema de granularidad base, una agregación de un esquema de granularidad G a G' se denota como $cubeView_{G',SUM(m)}(\mathfrak{D}, cubeView_{G,SUM(m)}(\mathfrak{D}, cubeBase))$.

2.4.2. Implementación en Datalog

En esta sección se presenta la implementación en Datalog que fue propuesta en [31], la cual está basada en los conceptos mencionados en la sección 2.4.1. Para representar estos conceptos utiliza la sintaxis estándar de Datalog mediante la definición de reglas, con la adición de funciones de agregación:

- **Predicados extensionales:** Dado n esquemas de dimensión $S[i] = \langle C, \nearrow \rangle$, se agrega un predicado relacional $Dim_i(X_{c_1}, ..., X_{c_p})$ para cada uno de ellos. Cada predicado Dim_i debe cumplir que $|C_i| = p$ y $c_j \in C_i, \forall j \in \{1, ..., p\}$

  Considerando el espacio multidimensional asociado a los esquemas de dimensión, $\mathfrak{G} : S[1], ..., S[n]$, y el correspondiente esquema de cubo base, $cubeBase = \langle G_b, m \rangle$, se agrega un predicado relacional $Cube\_base(X_{G_b[1]}, ..., X_{G_b[n]}, m)$

- **Predicados intensionales:** Los predicados intensionales adaptan la definición de cubos presentada en la formalización anterior. Para definir de manera completa la definición de dimensión, es necesario representar el concepto de jerarquía definiendo la regla $Hier_i(X_{c_j}, X_{c_k}) \leftarrow Dim_i(X_{c_1}, ..., X_{c_p})$. $Hier_i$ debe cumplir con $c_j \nearrow^* c_k$. La siguiente regla representa el cubo que se agrega sobre la dimensión Dim_i desde la categoria base a una superior $G'[i]$, considerando $X_{G'}$ como $X_{G'[1]}, ..., X_{G'[n]}$:

$$Cube\_view_{G_b,Dim_i}(X_{G'}, SUM(X_m)) \leftarrow Cube\_base(X_{G_b}, X_m), \\ Dim_i(..., X_{G_b[i]}, ..., X_{G'[i]}, ...), \\ Hier_i(X_{G_b[i]}, X_{G'[i]})$$

  Es necesario agregar otra regla que represente cubos para permitir la agregación en múltiples dimensiones. La siguiente regla modela el cubo que se agrega sobre la dimensión Dim_i desde $G[i]$ a una categoría superior $G'[i]$:

$$Cube\_view_{G,Dim_i}(X_{G'}, SUM(X_m)) \leftarrow Cube\_view_{G,Dim_k}(X_G, X_m), \\ Dim_i(..., X_{G[i]}, ..., X_{G'[i]}, ...), \\ Hier_i(X_{G[i]}, X_{G'[i]})$$

  Se considera que cualquier definición válida de $Cube\_view_{G,Dim_i}(X_{G'}, SUM(X_m))$ debe cumplir que G y G' sean granularidades adyacentes sobre la dimensión i y se puedan agregar elementos en $Dom(X_m)$.

Capítulo 3

Métricas de calidad propuestas

En este capítulo se presenta la definición y formalización de las métricas de calidad que proponemos para evaluar la calidad de cualquier DW en la herramienta a implementar. Previo a esto se presenta un caso de estudio que tomamos de [31], el cual consta de un DW particular que será utilizado para instanciar las formalizaciones de la sección 2.3.2 y ejemplificar las métricas propuestas.

3.1. Caso de estudio

En esta sección se ejemplifica con un caso de estudio la instanciación de un DW particular basándonos en la formalización para un DW genérico presentada en la sección 2.4.2. El caso de estudio consta de un DW que obtuvimos de [31], el cual será utilizado de aquí en adelante. El mismo busca representar, con un conjunto de datos abiertos muy reducido, la cantidad de ventas de libros dado el local, autor, ubicación geográfica, rango de precio y tiempo.

Consta de seis dimensiones: *Book* (representa los libros), *Author* (representa los autores de libros), *Time* (representa la dimensión temporal), *Shop* (representa los locales de venta), *Location* (representa la ubicación geográfica del autor), *Price* (representa los rangos de precio de los libros).

La tabla de hechos contiene los identificadores de las seis dimensiones (book_id, author_id, date_id, shop_id, city_id, price_subrange_id) y la medida *quantity* representa la cantidad de libros vendidos dado un libro, autor, fecha, ciudad y precio.

En la sección 2.1.2 se presentaron dos esquemas de diseño: esquema conceptual de dimensiones y medidas, y el esquema estrella. En la figura 3.1 se muestra un esquema conceptual de las dimensiones y medidas para nuestro ejemplo, y en el esquema estrella de la figura 3.2, se muestran las tablas de dimensiones y tabla de hechos del DW.

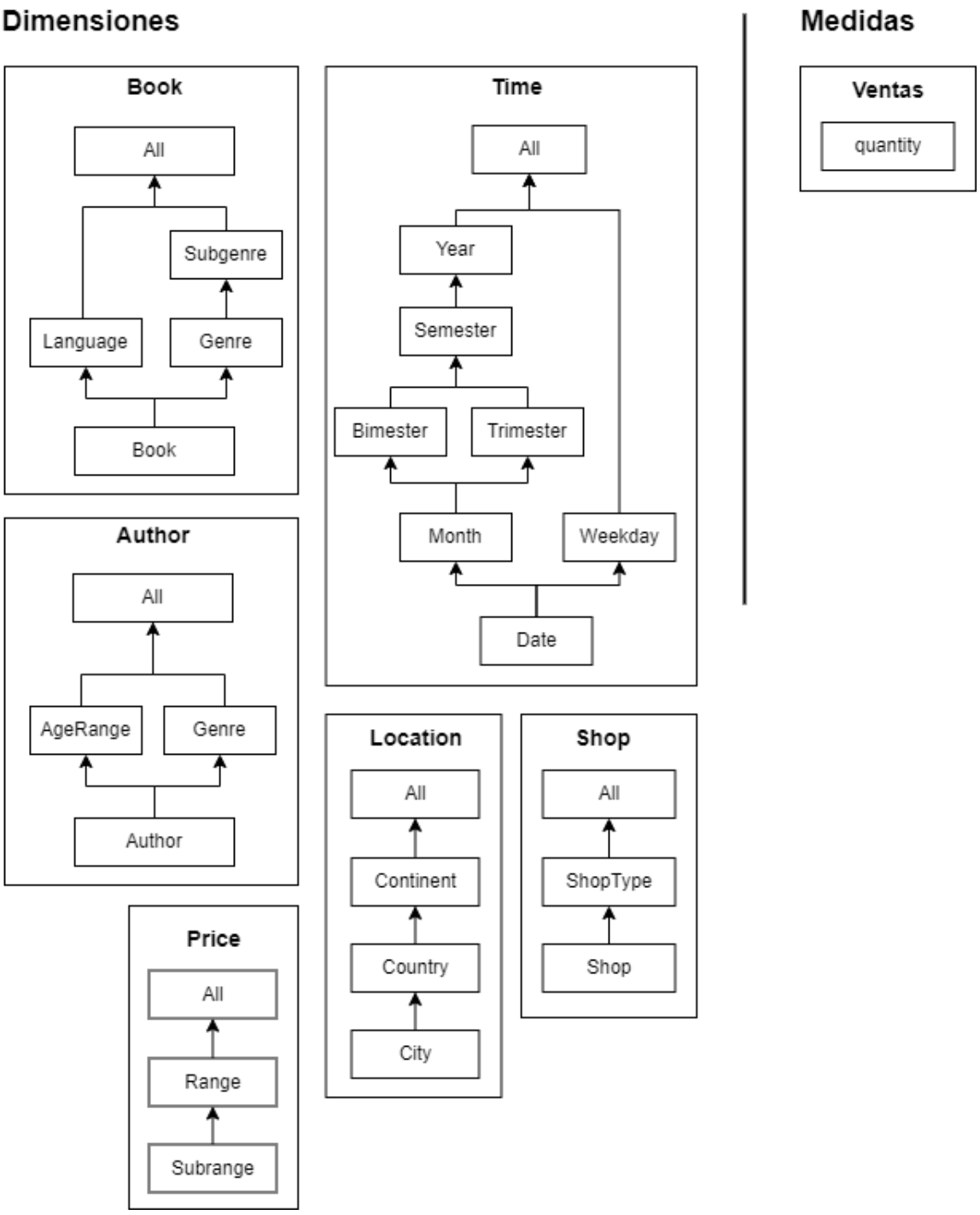


Figura 3.1: Esquema conceptual de las dimensiones del caso de estudio.

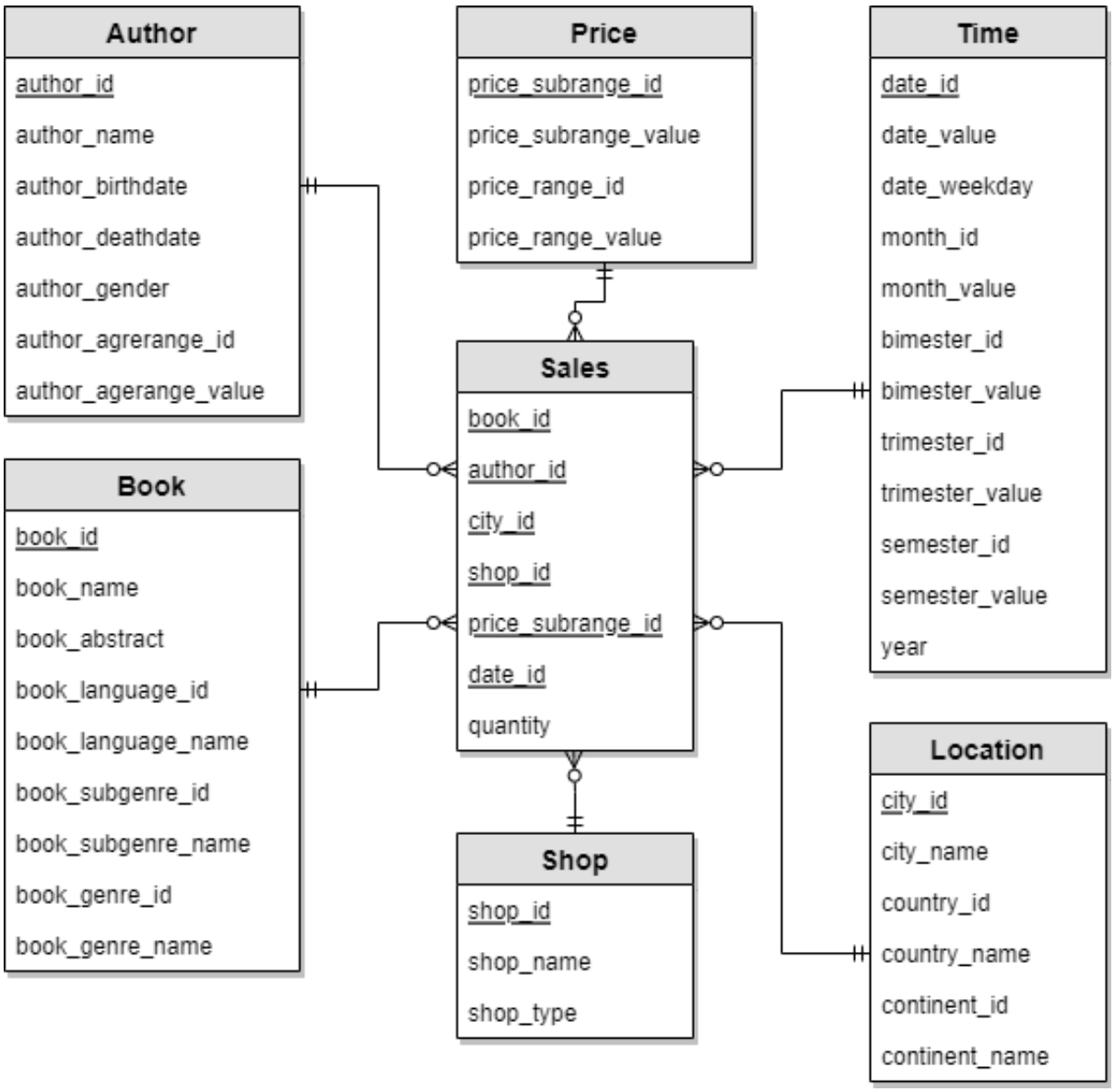


Figura 3.2: Esquema estrella del DW utilizado como caso de estudio.

3.1.1. Instanciación de formalizaciones

A partir de la formalización de Data Warehouse obtenida de [31], la cual fue mencionada en la sección 2.4.1, vamos a instanciar nuestro caso de estudio. Para eso es necesario definir las dimensiones, espacios multidimensionales y cubos:

- **Dimensiones:** Como se puede observar, el DW a utilizar tiene seis dimensiones: *Book*, *Author*, *Time*, *Price*, *Location* y *Shop*. Se agrega entonces un esquema de dimensión para cada una de ellas:

  $\mathcal{S}_{Book} = \langle \mathcal{C}_{Book}, \nearrow_{Book} \rangle$

$\mathcal{C}_{Book} = \{book, genre, subgenre, language, All\}$
$\nearrow_{Book} = \{(book, genre), (genre, subgenre), (subgenre, All),$
$(book, language), (language, All)\}$

$\mathcal{S}_{Author} = \langle \mathcal{C}_{Author}, \nearrow_{Author} \rangle$

$\mathcal{C}_{Author} = \{author, genre, ageRange, All\}$
$\nearrow_{Author} = \{(author, genre), (genre, All), (author, ageRange), (ageRange, All)\}$

$\mathcal{S}_{Price} = \langle \mathcal{C}_{Price}, \nearrow_{Price} \rangle$

$\mathcal{C}_{Price} = \{subrange, range, All\}$
$\nearrow_{Price} = \{(subrange, range), (range, All)\}$

$\mathcal{S}_{Shop} = \langle \mathcal{C}_{Shop}, \nearrow_{Shop} \rangle$

$\mathcal{C}_{Shop} = \{shop, shopType, All\}$
$\nearrow_{Shop} = \{(shop, shopType), (shopType, All)\}$

$\mathcal{S}_{Location} = \langle \mathcal{C}_{Location}, \nearrow_{Location} \rangle$

$\mathcal{C}_{Location} = \{city, country, continent, All\}$
$\nearrow_{Location} = \{(city, country), (country, continent), (continent, All)\}$

$\mathcal{S}_{Time} = \langle \mathcal{C}_{Time}, \nearrow_{Time} \rangle$

$\mathcal{C}_{Time} = \{date, weekday, month, bimester, trimester, semester, year, All\}$
$\nearrow_{Time} = \{(date, weekday), (date, month), (month, bimester), (month, trimester),$
$(bimester, semester), (trimester, semester), (semester, year), (year, All)\}$

La siguiente es una instancia posible para el esquema de dimensión $S_{Location}$:

$\mathcal{D}_{Location} = \langle \mathcal{U}_{Location}, <_{Location}, mem_{Location} \rangle$ con

$\mathcal{U}_{Location}$ = {California, Boston, Leeds, Porto, Beijing, United States, United Kingdom, Portugal, China, North America, Europe, Asia, ... , All}

$<_{Location}$ = {(California, United States), (Boston, United States), (Leeds, United Kingdom), (Porto, Portugal), (Beijing, China), (United States, North America), (United Kingdom, Europe), (Portugal, Europe), (China, Asia), (North America, All), (Europe, All), (Asia, All), ...}

$city = mem_{Location}(X), \forall X \in$ {California, Boston, Leeds, Porto, Beijing, ...}

$country = mem_{Location}(X), \forall X \in$
{United States, United Kingdom, Portugal, China, ...}

$continent = mem_{Location}(X), \forall X \in$ {North America, Europe, Asia, ...}

$All = mem_{Location}(\text{all})$

Se define también la relación de *roll-up* sobre $D_{Location}$:

$\mathcal{R}^{country}_{city}(\mathcal{D}_{Location}) = \{$(California, United States), (Boston, United States), (Leeds, United Kingdom), (Porto, Portugal), (Beijing, China), ... $\}$

$\mathcal{R}^{continent}_{country}(\mathcal{D}_{Location}) = \{$(United States, North America), (United Kingdom, Europe), (Portugal, Europe), (China, Asia), ... $\}$

$\mathcal{R}^{All}_{continent}(\mathcal{D}_{Location}) = \{$(North America, All), (Europe, All), (Asia, All), ... $\}$

- **Espacio multidimensional:** El esquema de espacio multidimensional está dado por la lista de esquemas de dimensión $\mathfrak{G} : S_{Book}, S_{Author}, S_{Price}, S_{Shop}, S_{Location}, S_{Time}$. Su correspondiente granularidad base es $G_b : book, author, subrange, shop, city, date$ y una instancia del esquema de espacio multidimensional $\mathfrak{G}$ es $\mathfrak{D} : \mathcal{D}_{Book}, \mathcal{D}_{Author}, \mathcal{D}_{Price}, \mathcal{D}_{Shop}, \mathcal{D}_{Location}, \mathcal{D}_{Time}$.
- **Cubos:** El cubo base para el ejemplo es $cubeBase_{Sales} = \langle G_b, m_{quantity} \rangle$. Tomando una granularidad adyacente a G_b sobre los elementos de la instancia de dimensión $\mathcal{D}_{Location}$, $G : book, author, subrange, shop, country, date$, podemos realizar la siguiente agregación:

$$cubeView_{G,SUM(m_{quantity})}(\mathfrak{D}, cubeBase_{Sales}) =$$
$$\Pi_{G,SUM(m_{quantity})}(cubeBase_{Sales} \bowtie R^{country}_{city}(\mathcal{D}_{Location}))$$

Datalog

Siguiendo la implementación en Datalog propuesta en [31], que fue presentada en la sección 2.4.2, definimos en el lenguaje Datalog los **predicados extensionales** relativos a las dimensiones, algunos de jerarquía, particularmente los de la jerarquía de la dimensión $S_{Location}$, y el cubo base para nuestro caso de estudio.

$Dim_{Book}(X_{book}, X_{language}, X_{genre}, X_{subgenre})$
$Dim_{Author}(X_{author}, X_{ageRange}, X_{genre})$
$Dim_{Price}(X_{subrange}, X_{range})$
$Dim_{Shop}(X_{shop}, X_{shopType})$
$Dim_{Location}(X_{city}, X_{country}, X_{continent})$
$Dim_{Time}(X_{date}, X_{weekday}, X_{month}, X_{bimester}, X_{trimester}, X_{semester}, X_{year})$
$HierLocation_{CityCountry}(X_{city}, X_{country})$
$HierLocation_{CountryContinent}(X_{country}, X_{continent})$
$HierLocation_{CityContinent}(X_{city}, X_{continent})$
$Cube\_base_{Sales}(X_{book}, X_{author}, X_{subrange}, X_{shop}, X_{city}, X_{date}, X_{quantity})$

Por último, se define un ejemplo de **predicado intensional** de la vista de cubo dada por la agrupación en los elementos de la dimensión *Location* de la categoría *city* a *country*:

$$Cube\_view_{G_b,Dim_{Location}}(X_{book}, X_{author}, X_{subrange}, X_{shop}, X_{country}, X_{date}, SUM(X_{quantity})) \leftarrow$$
$$Cube\_base_{Sales}(X_{book}, X_{author}, X_{subrange}, X_{shop}, X_{city}, X_{date}, X_{quantity}),$$
$$Dim_{Location}(X_{city}, X_{country}, X_{continent}),$$
$$HierLocation_{CityCountry}(X_{city}, X_{country})$$

3.2. Propuesta de métricas de calidad

Uno de los objetivos de este trabajo consiste en la definición y ejecución de métricas de calidad que permitan evaluar la calidad de un DW definido.

En base a lo mencionado en la sección 2.3.2, para evaluar la calidad del DW, se debe determinar que las dimensiones sean N:1 con totalidad, y se debe chequear la sumarizabilidad de las medidas del cubo.

Se definen entonces tres métricas: Sumarizabilidad, Totalidad y Jerarquía N:1.

3.2.1. Definiciones previas

Antes de definir formalmente las métricas a estudiar junto con su método de medición, es necesario mencionar algunos conceptos que serán incluidos posteriormente:

Instancia de granularidad: Dada una instancia de espacio multidimensional $\mathfrak{D}$ y un esquema de granularidad $\mathfrak{G}_i$, se define instancia de granularidad como $\mathfrak{G}_i(\mathfrak{D})$, es decir la proyección de los miembros de $\mathfrak{D}$ a las categorías presentes en $\mathfrak{G}_i$. La definición formal propuesta es tomada de Hurtado-Mendelzon [29].

$$\mathfrak{G}_i(\mathfrak{D}) = \Pi_{G_i}(cubeBase \bowtie \mathfrak{G}_i(\mathfrak{D}_1) \bowtie \mathfrak{G}_i(\mathfrak{D}_2) \bowtie \cdots \bowtie \mathfrak{G}_i(\mathfrak{D}_n))$$

Celda: Dado una instancia de granularidad $\mathfrak{G}_i(\mathfrak{D})$ se define celda como cada uno de los elementos $c_j \in \mathfrak{G}_i(\mathfrak{D})$. Esta definición es propuesta en este trabajo.

Ejemplo: Dado el DW definido en la sección 3.1, sea $\mathfrak{G}_i = \{$ *book*, *author*, *subrange*, *shop*, *city*, *date* $\}$ y $\mathfrak{D} = \{$ *Book*, *Author*, *Price*, *Shop*, *Location*, *Time* $\}$, un ejemplo de instancia de granularidad es:

$$\begin{aligned}\mathfrak{G}_i(\mathfrak{D}) = \{&(Book1, Author1, 1020, Shop1, California, 20200125),\\ &(Book2, Author1, 50100, Shop2, Boston, 20210225),\\ &(Book10, Author3, 1020, Shop15, Beijing, 20210814), \dots\}\end{aligned}$$

Y una celda perteneciente a instancia de granularidad es:

$$c_j = (Book1, Author1, 1020, Shop1, California, 20200125)$$

3.2.2. Definición de las métricas

Para formalizar las métricas a implementar, se utilizará la jerarquía de conceptos de calidad definida en la sección 2.3.1: dimensión, factor, métrica y método de medición.

Sumarizabilidad

Dado dos vistas de cubo $cubeViewA$ y $cubeViewB$ sobre el espacio multidimensional $\mathfrak{D}$ y esquema de granularidad $\mathfrak{G}$, para evaluar la sumarizabilidad se comparan los valores de la medida para cada una de sus celdas. Interesa particularmente que los $cubeViewA$ y $cubeViewB$ se obtengan por caminos distintos. Por ejemplo, siendo $(cubeBase, cubeView_1, \ldots, cubeView_i, \ldots, cubeViewA)$ el camino para obtener el $cubeViewA$ y $(cubeBase, cubeView'_1, \ldots, cubeView'_i, \ldots, cubeViewB)$ el camino para obtener el $cubeViewB$, son distintos si existe al menos un i tal que $cubeView_i \neq cubeView'_i$.

A continuación se presenta su formalización:

Dimensión: Consistencia
Factor: Integridad inter-cubeView
Métrica: Sumarizabilidad
Unidad de medida: 0 o 1
Granularidad: Celda
Método de medición: Dados $cubeView_1$ y $cubeView_2$ sobre el espacio multidimensional $\mathfrak{D}$, esquema de granularidad $\mathfrak{G}$ y siendo los elementos del $cubeView_i$ de la forma $< c, m_i >$ con c celda y m_i el valor de la medida. - Dados $cubeView1$ y $cubeView2$ con la misma granularidad $\mathfrak{G}$ - Para cada celda $c_i \in \mathfrak{G}(\mathfrak{D})$ y - Si $< c_i, m_1 > \in cubeView_1$, $< c_i, m_2 > \in cubeView_2$ y $m_1 = m_2$: - La métrica tiene valor 1 para c_i y se retorna: $< c_i, m_1, m_2, 1 >$ - Si $< c_i, m_1 > \in cubeView_1$, $< c_i, m_2 > \in cubeView_2$ y $m_1 \neq m_2$: - La métrica tiene valor 0 para c_i y se retorna: $< c_i, m_1, m_2, 0 >$ - Si $< c_i, m_1 > \in cubeView_1$ y $< c_i, m_2 > \notin cubeView_2$: - La métrica tiene valor 0 para c_i y se retorna: $< c_i, m_1, -, 0 >$ - Si $< c_i, m_1 > \notin cubeView_1$ y $< c_i, m_2 > \in cubeView_2$: - La métrica tiene valor 0 para c_i y se retorna: $< c_i, -, m_2, 0 >$

En caso de que el resultado de aplicar esta métrica sea 0 para al menos una celda, podemos garantizar que la propiedad de Sumarizabilidad no se cumple. En caso de que el resultado sea 1 para todas las celdas, no se puede asegurar que se cumpla dicha propiedad ya que no garantiza que las métricas de Totalidad y Jerarquía N:1 se cumplan.

Ejemplo: En base al DW presentado en la sección 3.1, elegimos $cubeView1$ y $cubeView2$ cuyos esquema de granularidad $\mathfrak{G}$ sea el nivel base para todas las dimensiones y en la dimensión *Book* el nivel sea *All*. Para obtener el $cubeView1$ hacemos un *roll-up* de *Book* a *Language* y luego de *Language* a *All*, mientras que para obtener el

cubeView2 hacemos *roll-up* de *Book* a *Genre*, luego de *Genre* a *Subgenre* y por último de *Subgenre* a *All*.

Si para simplificar tomamos el *cubeBase* como se muestra en la tabla 3.1 donde sólo se considera la dimensión *Book* que se muestra en la tabla 3.2 y agrupamos para obtener *cubeView1*, obtenemos que la cantidad es 5, pero por otro lado si agrupamos para obtener *cubeView2* obtenemos que para la misma celda (la única celda) la cantidad es 3. La medida de la métrica de sumarizabilidad para esa celda nos da 0, por lo que podemos asegurar que la sumarizabilidad no se cumple, en este caso porque *Genre* no está definido para todos los libros, por lo que no se cumple la totalidad.

cubeBase	
book_id	quantity
1	2
2	3

Tabla 3.1: *cubeBase* omitiendo columnas y solamente con la dimensión *Book*

Book			
book_id	book_language_id	book_genre_id	book_subgenre_id
1	1	-	1
2	1	2	2

Tabla 3.2: Dimensión *Book* omitiendo las columnas de los atributos no identificadores

Totalidad

Para evaluar la totalidad de las dimensiones, se contabilizan la cantidad de nulos dado un nivel de una jerarquía perteneciente a dicha dimensión.

Dimensión: Completitud
Factor: Densidad
Métrica: Totalidad
Unidad de medida: < Cantidad de nulos, Cantidad de no nulos >
Granularidad: Instancia de nivel
Método de medición: - Dado un nivel n de una jerarquía perteneciente a una dimensión del cubo - Se definen los contadores cant_nulos = 0 y cant_no_nulos = 0 - Para cada tupla en la tabla de la dimensión correspondiente: - Si el valor correspondiente al nivel n es nulo: - cant_nulos = cant_nulos + 1 - Sino: cant_no_nulos = cant_no_nulos + 1

Ejemplo: Tomando la dimensión *Book* como se muestra en la tabla 3.2 tenemos que para el nivel *Language* la métrica de calidad da 1 ya que en la columna de los identifica-

dores de ese nivel no hay nulos y para el nivel *Genre* la métrica de calidad da 0 porque para el libro con book_id=1, el book_genre_id es nulo.

Jerarquía N:1

Para evaluar que las relaciones de una jerarquía cumplan N:1 se tomarán dos niveles adyacentes de una jerarquía perteneciente a una determinada dimensión, contabilizando aquellas tuplas pertenecientes al nivel inferior que se correspondan con una única tupla en el nivel superior.

Dimensión: Consistencia
Factor: Integridad inter-nivel
Métrica: Jerarquía N:1
Unidad de medida: 0 o 1
Granularidad: Par de instancias de nivel
Método de medición: - Dados dos niveles adyacentes de una jerarquía perteneciente a una dimensión del cubo, n_1 inferior y n_2 superior: - Para cada par $t_1 = (t_{1_{n_1}}, t_{1_{n_2}})$ con $t_{1_{n_1}} \in n_1$ y $t_{1_{n_2}} \in n_2$ - Si existe un par $t_2 = (t_{2_{n_1}}, t_{2_{n_2}})$ tal que $t_{1_{n_1}} = t_{2_{n_1}}$ y $t_{1_{n_2}} \neq t_{2_{n_2}}$: - Entonces la métrica vale 0 - En caso contrario: - La métrica vale 1

Ejemplo: En este caso si tomamos la dimensión *Book* como se muestra en la tabla 3.3, tenemos que si chequeamos para el par de niveles *Genre* y *Subgenre*, la métrica da 0, lo que significa que no se cumple que la jerarquía sea N:1, ya que hay un libro con book_genre_id=2 que tiene book_subgenre_id=1 y otro con mismo book_genre_id pero que tiene book_subgenre_id=2, o sea, un mismo elemento de nivel inferior está asociado a dos elementos distintos de nivel superior.

Book			
book_id	book_language_id	book_genre_id	book_subgenre_id
1	1	2	1
2	1	2	2

Tabla 3.3: Dimensión *Book* omitiendo las columnas de los atributos no identificadores

Capítulo 4

Análisis y diseño del framework

Este capítulo tiene como objetivo detallar el proceso realizado antes de llevar a cabo la implementación. En primer lugar, se describe el proceso de análisis de requerimientos, de donde se obtienen los casos de uso a implementar. Posteriormente se describen y se muestran mediante diagramas de arquitectura, las decisiones de diseño tomadas.

4.1. Análisis de requerimientos

En esta sección se expone el análisis de los requerimientos que deben ser considerados en la etapa de implementación de la herramienta. Para eso, se realiza una breve descripción del problema a resolver, detallando a continuación los requerimientos tanto funcionales como no funcionales, y finalizando con la especificación de los casos de uso. Esta especificación servirá como insumo para la etapa de implementación.

Problema a resolver:

El *framework* a implementar debe brindar la posibilidad al usuario de definir su DW a partir una conexión a la base de datos. En base a esta definición, el usuario debe poder especificar, ejecutar, visualizar y guardar consultas de tipo *roll-up*, así como ejecutar métricas de calidad que aporten a la medición de la calidad del DW.

Se presentan a continuación los requerimientos funcionales y no funcionales, establecidos de acuerdo a la descripción y el análisis de la realidad a desarrollar.

Requerimientos funcionales

- **Manejo de sesión de usuario:** la aplicación debe manejar la creación y el inicio de sesión de usuario, donde cada usuario maneje sus conexiones a base de datos junto con las respectivas definiciones de DW y operaciones guardadas.

- **Conexión a base de datos:** la aplicación debe permitir al usuario configurar una conexión a base de datos y establecer conexión con dicha base, la cual será utilizada para definir el DW y obtener los datos según las operaciones realizadas.

- **Definir un DW:** una vez establecida y configurada la conexión con la base de datos ingresada, se debe permitir al usuario ingresar la definición del DW. En este

punto se definirán los cubos junto con su tabla de hechos, medidas, dimensiones, jerarquías y niveles que correspondan.

- **Ejecutar operaciones de *roll-up*:** luego de definido el DW, el usuario debe poder especificar y ejecutar operaciones de *roll-up*, partiendo del cubo base.
- **Guardar las operaciones realizadas:** se deben persistir las operaciones realizadas, de manera que el usuario pueda recuperarlas e iniciar su ejecución sin tener que especificar nuevamente la operación.
- **Ejecución de métricas de calidad:** la aplicación debe permitir la ejecución de las métricas de calidad Sumarizabilidad y Totalidad, que permitan evaluar la calidad de datos del DW.
- **Visualizar resultados de las operaciones:** se debe poder visualizar en forma tabular, el resultado de las operaciones ejecutadas, tanto para operaciones de *roll-up*, como para cálculos de métricas.
- **Generar código pyDatalog:** el usuario debe poder obtener el código pyDatalog utilizado para ejecutar la operación correspondiente, de manera de poder utilizarlo por fuera de la aplicación.

Requerimientos no funcionales

- **Conexión con bases de datos:** el sistema debe poder establecer conexión con bases de datos locales y externas.
- **Utilizar pyDatalog:** el sistema debe realizar las consultas referentes a las vistas de cubo y métricas de calidad en código pyDatalog, para adecuarse al sistema matemático presentado en la sección 2.4.2.

4.1.1. Casos de uso

En base al análisis de los requerimientos funcionales de la aplicación, se identificaron los siguientes casos de uso a implementar:

ID	Nombre
01	Crear sesión A.1
02	Iniciar sesión A.2
03	Cerrar sesión A.3
04	Configurar conexión a base de datos 4.1
05	Conectarse a base de datos A.4
06	Borrar conexión a base de datos A.5
07	Definir DW 4.2
08	Definir Cubo A.6
09	Definir Dimensión A.7
10	Definir Jerarquía A.8
11	Definir Nivel A.9
12	Generar pyDatalog 4.3
13	Ejecutar consultas Roll-up 4.4
14	Guardar consultas A.10
15	Ejecutar métricas de calidad 4.5

Especificación de casos de uso

Se detallan a continuación los casos de uso más importantes para comprender el funcionamiento de la aplicación. La especificación del resto de los casos de uso se pueden consultar en el anexo A presentado en el capítulo A.

04 - Configurar conexión a base de datos

Identificador: 04
Nombre: Configurar conexión a base de datos
Objetivo: Configurar una conexión de base de datos
Actores: Usuario
Precondiciones: Se ha iniciado sesión
Descripción: Un usuario desea configurar una conexión a base de datos asociada a su sesión. Con posibilidad de editar todas sus conexiones.
Flujo normal: 1. Usuario selecciona la opción de configurar conexión. 2. Sistema despliega un listado con las conexiones existentes y un botón para ingresar una nueva conexión. 3. Usuario presiona el botón de ingresar una nueva conexión. 4. Sistema despliega un formulario con los siguientes campos: - DBMS (seleccionar de listado) - Nombre de la conexión (requerido) - Hostname (requerido) - Puerto (requerido) - Nombre de la base de datos (requerido) - Usuario (requerido) - Contraseña (requerido) 5. Usuario completa los campos y selecciona la opción de guardar. 6. «extiende» Conectarse a base de datos.
Flujos alternativos: 3A. Usuario selecciona una conexión existente: 3A.1. Sistema despliega un formulario con los campos completados de la conexión seleccionada (campos mencionados en paso 4). 3A.2. Usuario modifica los datos y selecciona la opción de guardar. 3A.3. Volver al paso 6.
Poscondiciones: Se guarda la conexión en las conexiones de la sesión

Tabla 4.1: Configurar conexión a base de datos

07 - Definir DW

Identificador: 07
Nombre: Definir DW
Objetivo: Definir el DW asociado a la conexión
Actores: Usuario
Precondiciones: Se ha establecido una conexión a base de datos
Descripción: Un usuario define el DW asociado a la conexión.
Flujo normal: 1. Usuario selecciona la opción de definir DW. 2. Sistema despliega los campos para definir el esquema asociado: - Nombre (requerido) - Descripción (opcional) 3. Usuario completa los datos y selecciona la opción de guardar. 4. «incluye» Definir Cubo A.6. 6. Fin del caso de uso.
Flujos alternativos: 5A. Usuario desea agregar otro cubo: 5A.1. Vuelve al paso 4
Poscondiciones: Se guarda en el sistema la definición del DW

Tabla 4.2: Definir DW

12 - Generar pyDatalog

Identificador: 12
Nombre: Generar pyDatalog
Objetivo: Generar el código pyDatalog para ejecutar consultas
Actores: Usuario
Precondiciones: Existe una conexión y un DW definido.
Descripción: A partir de la definición del DW se genera el archivo pyDatalog asociado para realizar las operaciones de *roll-up* correspondientes.
Flujo normal: 1. Usuario selecciona la opción de generar pyDatalog. 2. Sistema genera el código pyDatalog necesario para realizar operaciones de *roll-up* sobre las distintas jerarquías del DW definido. 3. Fin del caso de uso.
Poscondiciones: Se genera el archivo pyDatalog asociado al DW

Tabla 4.3: Generar pyDatalog

13 - Ejecutar consultas Roll-Up

Identificador: 13
Nombre: Ejecutar consultas Roll-Up
Objetivo: Ejecutar una consulta para el DW definido
Actores: Usuario
Precondiciones: Se generó el código pyDatalog para el DW.
Descripción: El usuario selecciona la consulta a ejecutar y el sistema despliega el resultado correspondiente.
Flujo normal: 1. Usuario selecciona la opción del cálculo de consultas. 2. Sistema despliega una pantalla con la opción de ingresar nueva consulta, utilizar la consulta anterior y un listado de consultas guardadas. 3. Usuario selecciona la opción de ingresar nueva consulta. 4. Sistema despliega el listado de tablas de hechos del DW. 5. Usuario selecciona una de las tablas de hechos. 6. Sistema despliega las dimensiones asociadas a la tabla de hechos, con sus jerarquías y niveles asociados. 7. Usuario selecciona la dimensión y el nivel por el cual realizar la operación y selecciona la opción de calcular resultado. 8. Sistema despliega el resultado de la consulta. 9. Fin del caso de uso.
Flujos alternativos: 3A. Usuario desea utilizar la consulta anterior: 3A.1. Usuario utiliza la consulta generada anteriormente, seleccionando la dimensión y el nivel por el cual subir. 3A.2. Vuelve al paso 8 3B. Usuario desea seleccionar una consulta guardada: 3B.1. Usuario selecciona una consulta guardada 3B.2. Vuelve al paso 8 7A. Usuario desea seleccionar otra dimensión y nivel para realizar la operación 7A.1. Vuelve al paso 7
Poscondiciones: Se ejecuta una consulta y se visualiza el resultado.

Tabla 4.4: Ejecutar consultas de Roll-up

15 - Ejecutar métricas de calidad

Identificador: 15
Nombre: Ejecutar métricas de calidad
Objetivo: Ejecutar una métrica de calidad sobre el DW definido por el usuario y obtener el resultado
Actores: Usuario
Precondiciones: Un DW fue definido previamente.
Descripción: Se selecciona una de las métricas de calidad disponibles para ejecutar sobre el DW, retornando el resultado y el código pyDatalog.
Flujo normal: 1. Usuario selecciona la opción de métricas de calidad. 2. Sistema lista las dimensiones, factores y métricas de calidad disponibles. 3. Usuario selecciona una dimensión de las disponibles. 4. Sistema lista los factores asociados a la dimensión seleccionada. 5. Usuario selecciona uno de los factores disponibles. 6. Sistema lista las métricas de calidad asociadas al factor seleccionado. 7. Usuario selecciona la métrica de calidad que desea calcular. 8. Sistema ejecuta las consultas necesarias para obtener el resultado de la métrica seleccionada, retornando el mismo en forma tabular y su código correspondiente en pyDatalog. 9. Fin del caso de uso.

Tabla 4.5: Ejecutar métricas

4.2. Diseño de la base de datos

Como se estableció en los requerimientos de la sección 4, la aplicación debe permitir almacenar y recuperar información de los usuarios, sus conexiones a bases de datos, las definiciones de DW y las consultas realizadas.

Para contemplar estos requerimientos, en primer lugar se identifican las grandes entidades a modelar, las cuales son: Usuario, Conexión, DW y Consulta. En base a estas entidades se diseñó un modelo entidad-relación (MER) con atributos, que se utilizó como insumo en la implementación de la base de datos que se detalla en la sección 5.2.

En la figura 4.1 se puede observar el MER donde se representan las entidades con sus atributos y relaciones entre ellas con sus respectivas cardinalidades. En primer lugar podemos ver como en la relación entre las entidades Usuario y Conexión, cada usuario puede tener varias conexiones. Luego se observa una relación entre DW y la agregación de Usuario y Conexión, debido a que para cada conexión de un usuario, se define un DW. Para contemplar todos los elementos a definir en el DW, se agregan las entidades Cubo, Dimensión, Jerarquía y Nivel. Cada DW puede contemplar N-Cubos, cada cubo tiene N-Dimensiones, cada dimensión tiene N-Jerarquías y cada jerarquía tiene N-Niveles. Por último se tiene la entidad Consulta y su relación con Cubo, debido a que las consultas se

realizan a partir de un cubo definido.

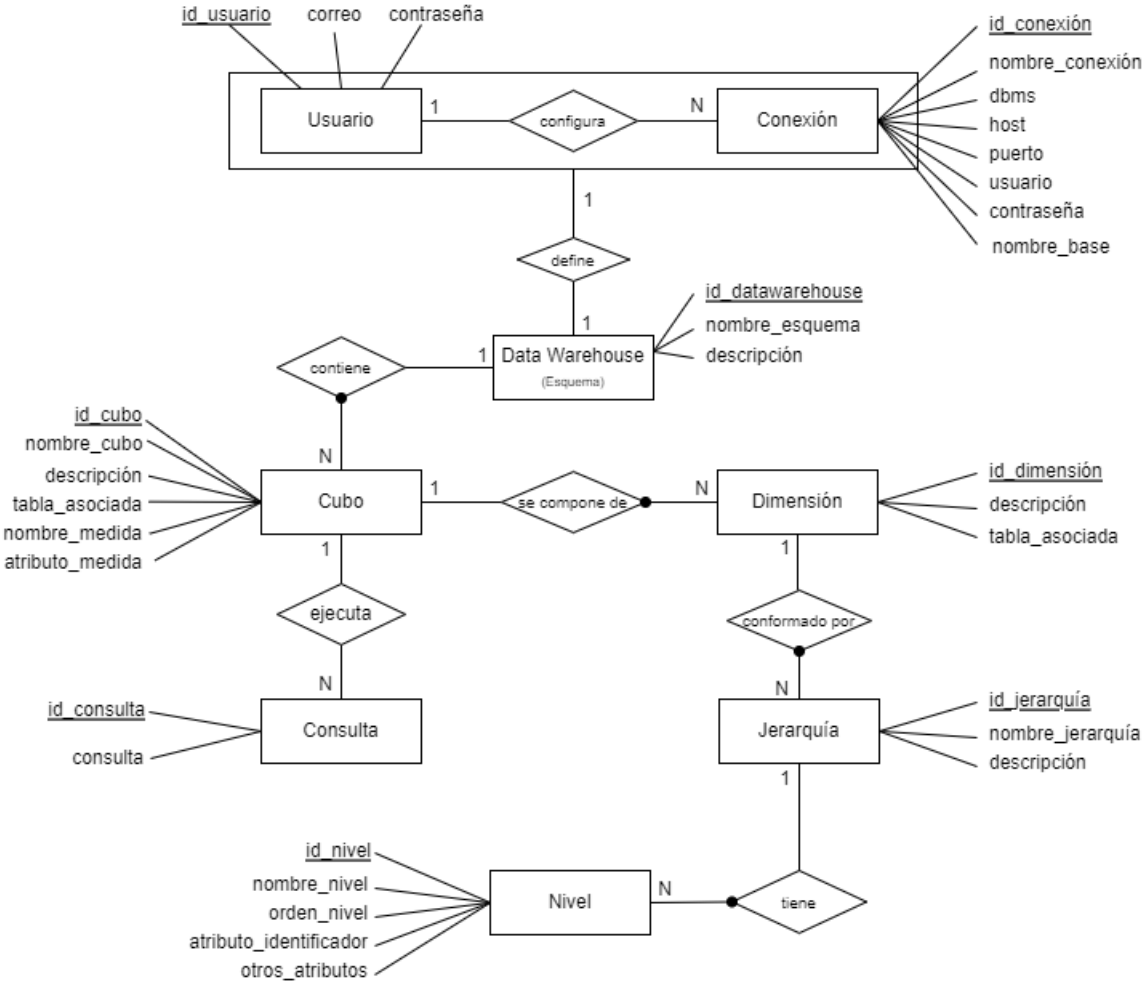


Figura 4.1: Modelo entidad-relación con atributos

4.3. Arquitectura del software

Para cumplir con los requerimientos planteados, se tomó la decisión de desarrollar una aplicación de escritorio, desde la cual los usuarios pueden establecer la conexión a su base de datos fuente, que será el insumo para poder realizar toda la definición del DW y las consultas mediante pyDatalog.

Por otra parte, la información de los usuarios, conexiones, definiciones y consultas realizadas, deben ser persistidas en una base de datos propia de la aplicación. La misma almacenará la información de todos los usuarios que hagan uso de la herramienta.

En las próximas secciones, se describe con mayor profundidad la arquitectura de la solución a desarrollar, justificando las decisiones de diseño que se fueron tomando.

4.3.1. Componentes de la aplicación

En la presente sección se detallan los componentes que forman parte de la aplicación, tanto internamente, como también componentes externos con los que se establece comunicación para el correcto funcionamiento. La aplicación está compuesta por tres grandes capas: presentación, servicios y la capa lógica de pyDatalog.

La capa de presentación esta compuesta por diferentes interfaces de usuario: autenticación, definición de conexión, definición del DW, consultas, etc, y es la encargada de ofrecer interacción con los usuarios finales.

En un segundo nivel se encuentra la capa de servicios, que es la encargada de procesar las interacciones de los usuarios, funcionando en ocasiones como intermediaria entre la capa de presentación y la capa de lógica de pyDatalog, y encargada de enviar los resultados obtenidos a la capa de presentación.

Por último, la capa lógica de pyDatalog, se encarga de instanciar la definición del DW del usuario en lenguaje Datalog, ejecutar las consultas recibidas y retornar los resultados a la capa de servicios.

Dentro de los componentes externos se encuentran:

- Base de datos de la aplicación, que será la encargada de almacenar todo lo referente a los usuarios, definiciones de DW y consultas realizadas.
- Base de datos del usuario, con la cual la aplicación debe establecer conexión para obtener datos tanto para la definición del DW, como para la ejecución de las consultas pyDatalog.

En la figura 4.2, se puede observar el diagrama de componentes de la aplicación, donde se encuentran todos los componentes mencionados, y la interacción entre ellos:

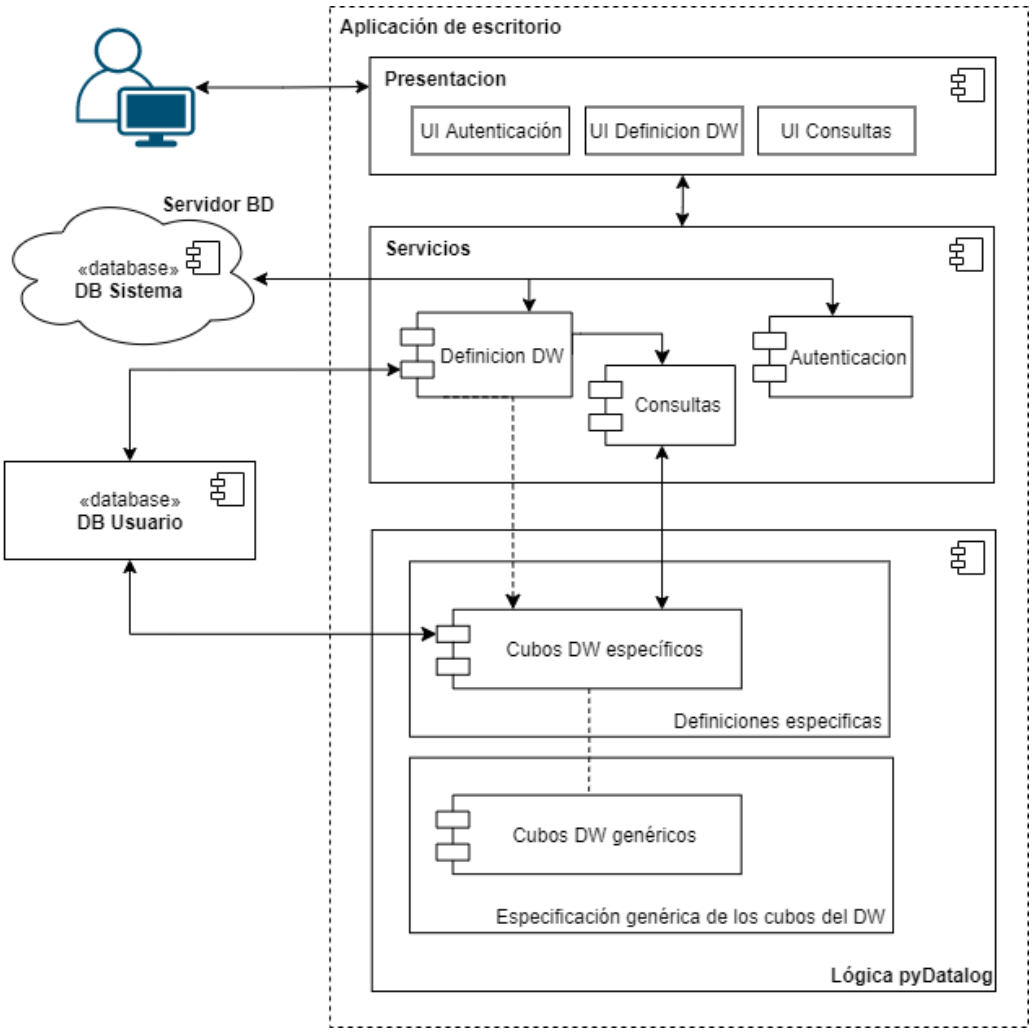


Figura 4.2: Diagrama de componentes.

La capa de servicios debe establecer comunicación con todos los componentes del sistema. En primer lugar, recibe desde la capa de presentación las indicaciones del usuario y le retorna la respuesta luego de interactuar con el resto de los componentes necesarios.

Luego debe establecer conexión con la base de datos de la aplicación, por ejemplo, para efectuar la autenticación de usuario y recuperar/almacenar sus conexiones, definiciones de DW asociadas y consultas realizadas.

Para la definición del DW se debe establecer comunicación con la base de datos del usuario, de manera de obtener las tablas y atributos a utilizar.

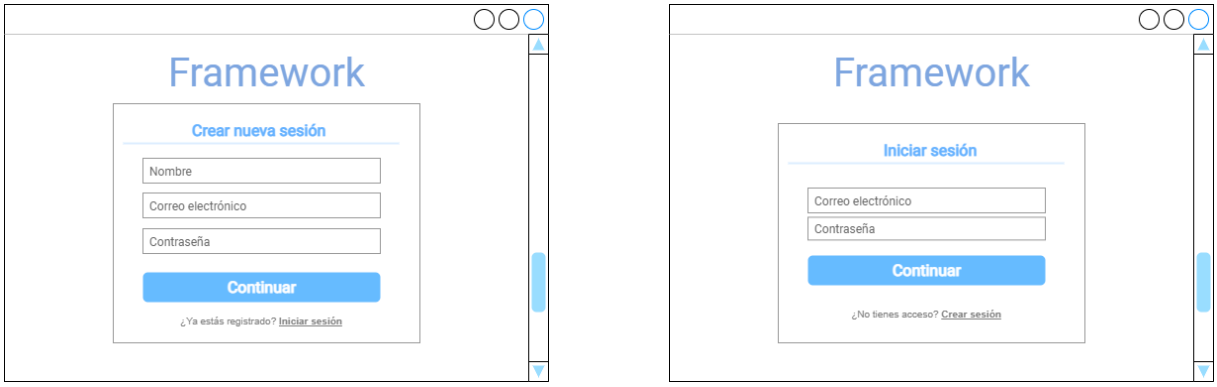
Por último, se tiene la comunicación con la capa lógica de pyDatalog. Esta capa contiene una definición genérica de DW en lenguaje Datalog, la cual se instancia con la definición del usuario. Esta definición es recibida desde la capa de servicios, así como también las consultas a ejecutar. La capa lógica se encarga de ejecutar las consultas y enviar el resultado a la capa de servicios, la cual a su vez la procesa y envía a la capa de presentación para que puedan ser visibles por el usuario.

4.4. Interfaz de usuario

Con el objetivo de reducir el tiempo en la fase de implementación y llegar con las ideas de funcionalidades a implementar lo más claras posible, se decidió diseñar una serie de bosquejos de la interfaz de usuario a implementar.

Crear sesión e iniciar sesión

En esta funcionalidad el usuario puede iniciar sesión con una cuanta registrada, o bien registrarse para obtener una cuenta.



(a) Crear sesión **(b)** Iniciar sesión

Figura 4.3: Crear sesión e iniciar sesión

Configurar conexión

En esta funcionalidad se carga la lista de conexiones creadas por el usuario y se permite la configuración de una nueva conexión. Por último se puede conectar a la base deseada, que será la utilizada para definir el DW.

Figura 4.4: Configurar conexión

Definición del Data Warehouse

En la figura 4.5 se ilustra el proceso de definición del DW. En cada pantalla, a la izquierda se tiene un panel que despliega dinámicamente en forma de árbol la definición, desde el esquema hasta los niveles (los cuales se pueden reordenar) y se permite agregar elementos. Del lado derecho se muestra el formulario asociado al elemento seleccionado del árbol, permitiendo editar sus campos y borrar el elemento.

(a) Crear esquema

(b) Crear cubo

(c) Crear dimensión

(d) Crear jerarquía

(e) Crear nivel

Figura 4.5: Definición del DW

Generación de consultas

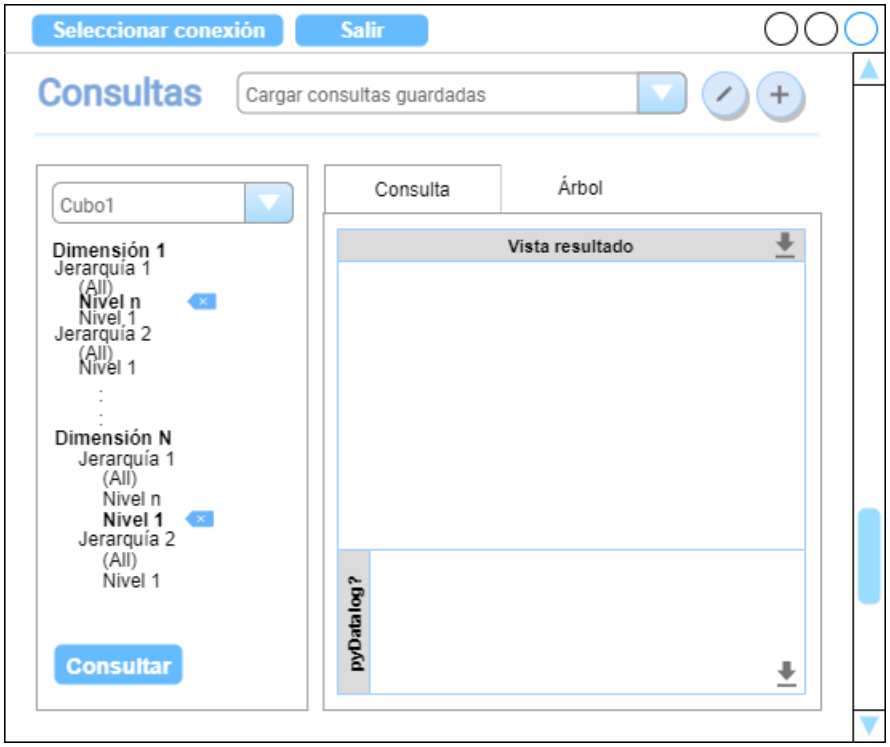


Figura 4.6: Consultas

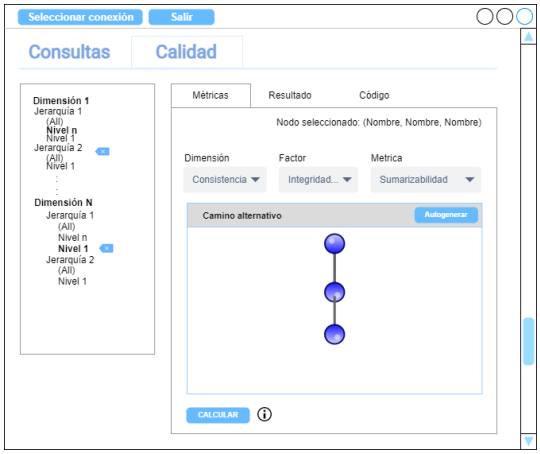
En la figura 4.6 se observa la página de consultas, a la cual se accede luego de tener definido el DW. En la parte superior se tiene la posibilidad de seleccionar operaciones guardadas, es decir, cubo y niveles seleccionados, para ejecutar. En el panel izquierdo, es donde se indica la operación que se quiere hacer, seleccionando el cubo y los niveles de cada dimensión involucrada.

En el panel derecho se puede observar en la primer pestaña, el resultado de la consulta y el código pyDatalog generado para ejecutar dicha consulta, mientras que en la otra pestaña se ubicará el árbol de consultas generadas.

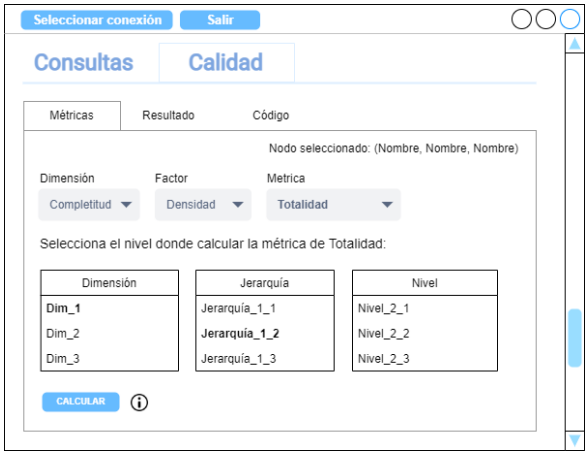
Cálculo de métricas de calidad

En esta pantalla se le permite al usuario seleccionar una métrica de calidad y ejecutarla sobre el DW definido, obteniendo los resultados y el código generado.

En la figura 4.7a podemos ver el diseño de la pestaña de cálculo de la métrica de Sumarizabilidad. En ésta debemos generar un camino alternativo al seleccionado en la pestaña de "Árbol", el cual se indica en la sección superior derecha de la pantalla, o bien podemos seleccionar la opción de "Generar" para obtener un camino alternativo de forma automática. En la sección inferior podemos ver la opción de "Calcular" la métrica sobre el camino alternativo seleccionado, y debajo de ésta el resultado del cálculo de la métrica.



(a) Métrica de Sumarizabilidad



(b) Métrica de Totalidad

La figura 4.7b muestra el diseño de la pestaña de cálculo de la métrica de Totalidad, la cual consiste en seleccionar dimensión, jerarquía y nivel (manteniendo este orden) donde quiere calcularse la métrica. Finalmente debajo de las tablas se encuentra el botón de "Calcular" para realizar el cálculo de la métrica sobre el nivel seleccionado.

Capítulo 5

Implementación

En el presente capítulo, se exponen con sus respectivos fundamentos, todos los aspectos considerados y decisiones tomadas referentes a la implementación de la aplicación, partiendo de lo obtenido en el análisis y diseño de la sección 3.2.2.

En la sección 5.1 se presentan las decisiones tomadas respecto a la implementación de la aplicación, como el tipo de aplicación a desarrollar, la infraestructura y las tecnologías.

En la sección 5.2 son presentadas las decisiones tomadas a nivel de base de datos, tanto para la base de datos de la aplicación como para la conexión con las bases del usuario.

Por último, en la sección 5.3 se describen las consideraciones a tener en cuenta con respecto a la implementación de las consultas y las métricas de calidad. Se describen las validaciones sobre el DW definido por el usuario, los diferentes componentes para la ejecución de consultas o métricas, y el manejo de los resultados.

5.1. Aplicación

En primer lugar se tomó la decisión de implementar una aplicación de escritorio ya que éstas brindan un mejor tiempo de respuesta, nos facilita la conexión con las bases de datos locales del usuario, y a diferencia de las aplicaciones web, no es necesario lidiar con las incompatibilidades de los diferentes navegadores.

Para el desarrollo de dicha aplicación, se tomo la decisión de utilizar la tecnología Electron.js, un *framework* que permite, entre sus características principales, el desarrollo de aplicaciones multiplataforma (Cross Platform Apps) y actualizaciones automáticas, lo que permite mitigar dos de las principales desventajas que presentaban las aplicaciones *desktop* contra las aplicaciones web.

Otro de los motivos por los cuales se optó por utilizar Electron.js, es el hecho de que las tecnologías que pueden ser utilizadas para este *framework* son del *stack* tecnológico del equipo (por ejemplo, React.js detallada más adelante), lo cual facilitó el desarrollo.

5.1.1. Tecnologías y librerías utilizadas

Las tecnologías utilizadas en la implementación de la aplicación son las siguientes:

- **Electron.js [2]:**

  Electron es un *framework* basado en Node.js (entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome).

  Los principales aspectos favorables que presenta son:

  - Crear aplicaciones de escritorio usando tecnologías web conocidas (Javascript/ES6, HTML 5, CSS 3) como librerías asociadas (jQuery, React, etc)
  - Al incluir Chromium y Node.js en su binario, permite desarrollar aplicaciones multiplataforma (Windows, Mac OS X y Linux) sin necesidad de desarrollo nativo.
  - Uso de componentes de terceros, por ejemplo semantic-ui.
  - Propiedades nativas para el User Experience (Native UX)
  - Integra una API para actualizaciones automáticas y variedad para hacer despliegues y distribuciones.
  - Herramientas para depuración y análisis de código y testing.
  - Integración con bases de datos relacionales o NoSQL

- **React.js [3]:** Es una biblioteca de JavaScript para la construcción de interfaces de usuario. React se basa en el uso de componentes independientes y reutilizables para crear interfaces modulares y adaptables.

- **Semantic UI [4]:** *Framework* para el desarrollo de *front-end*, que contiene componentes semánticos pre-construidos, lo que ayuda a agilizar el desarrollo, evitando tareas de algunos componentes necesarios y de alto costo. En el desarrollo de la aplicación fue utilizada la integración a React de Semantic UI.

- **JavaScript [5]:** Lenguaje de programación interpretado, que se utiliza más que nada en conjunto con otras tecnologías web del lado del cliente para darle comportamiento a las páginas web.

- **HTML [6]:** Lenguaje de marcado que es utilizado para estructurar un contenido web, por ejemplo, definiendo párrafos, encabezados y tablas de datos, o insertando imágenes y videos en la página.

- **CSS [7]:** Lenguaje de reglas de estilo que es utilizado para aplicar estilo a un contenido HTML, por ejemplo, estableciendo colores de fondo y tipos de letra, distribuyendo el contenido en múltiples columnas, y todo lo referente a el estilo visual.

- **SASS [8]:** Extensión de CSS que permite la definición de variables, reglas anidadas, *inline imports*, entre otras cosas.

- **Node.js [9]:** Es un entorno de ejecución *open source* y *cross-platform* para el desarrollo de aplicaciones *server-side* y de red, escritas en JavaScript.

- **Redux [10]:** Redux es una librería de *open source* en JavaScript para la gestión de estado en las aplicaciones.

- **Chromiun [11]:** Es un proyecto *open source browser*, que se utiliza como código base para nuevos navegadores. Chromium fue diseñado para brindar una interfaz más estable, rápida y minimalista al usuario.
- **Engine V8 JavaScript [12]:** V8 es un motor de JavaScript y WebAssembly [13] de alto rendimiento y *open source*, desarrollado en C++. Es usado en los conocidos Chrome y Node.js, entre otros. Este implementa ECMAScript [14] y WebAssembly, y corre en Windows 7 y posteriores, macOS 10.12+, y Linux que use x64, IA-32, ARM, o procesadores MIPS.
- **Python [15]:** Python es un lenguaje de programación interpretado, fuertemente tipado, orientado a objetos, multiplataforma y de alto nivel con semántica dinámica.
  - pyDatalog: biblioteca que agrega paradigmas de programación lógica al *toolbox* de Python. Permite entre otras cosas, ejecutar consultas lógicas en bases de datos u objetos Python, y definir clases de Python utilizando cláusulas lógicas.
- **Docker [16]:** Docker es una plataforma *open source* que permite el desarrollo y ejecución de aplicaciones desde cualquier infraestructura, ya que provee la habilidad de empaquetar y correr una aplicación en un entorno levemente aislado llamado contenedor. Esto permite la ejecución de aplicaciones sin depender de lo instalado actualmente en el *host*.

5.2. Base de datos

A nivel de base de datos, se presentaron dos decisiones importantes a tomar. En primer lugar, el tipo y el sistema de base de datos a utilizar para almacenar los datos generados por nuestra aplicación, y por otra parte, la forma o herramienta para establecer la conexión entre nuestra aplicación y la base de datos del usuario. En los siguientes apartados, se presentarán y justificarán ambas decisiones.

5.2.1. Tecnologías utilizadas

Las tecnologías utilizadas para las conexiones con las bases de datos son las siguientes:

- **Firebase [17]:** Servicio de Google alojado en la nube, utilizado para la autenticación y almacenamiento de datos de la aplicación.
- **Python [15]:** Lenguaje utilizado para implementar, mediante la biblioteca pyDatalog [25] con SQLAlchemy [18], las consultas al DW definido por el usuario
- **Node.js [9]:** Entorno de ejecución donde, mediante el uso del package **Knex [19]:**, se realizan las conexiones y consultas SQL a las bases de datos del usuario. Además, también se ejecutan las consultas lógicas haciendo pedidos a un código Python desarrollado por la aplicación.

5.2.2. Base de datos de la aplicación

Para la base de datos de la aplicación, lo primero que debimos decidir fue si implementarla de forma local o remota. Una base de datos local tiene la ventaja de no necesitar conexión a internet pero tiene algunas limitaciones:

- Los datos del usuario se almacenan en el equipo que fue creado, por lo que un usuario no podrá recuperar sus datos desde otro equipo
- Una ruptura en el equipo puede ocasionar la perdida total de la información

En consecuencia, se decidió implementar la base de datos remota, almacenando los datos en un servidor accesible por todos los usuarios que instalen la aplicación. De esta forma, un usuario puede recuperar su información desde cualquier equipo.

En cuanto al tipo de base de datos, se decidió utilizar una base de datos no relacional, del tipo documental, donde se utilizan colecciones y documentos en formato JSON. Esta decisión fue tomada basándonos fuertemente en que la definición del DW en la aplicación, se va construyendo y guardando temporalmente en forma de JSON debido a la forma jerárquica que requiere la misma. Esto nos facilita las operaciones de almacenamiento y su recuperación, las cuales son prácticamente directas.

Otros datos que deben almacenarse son los usuarios creados, las conexiones a base de datos de los usuarios y las consultas realizadas.

Firebase

En base a estas condiciones es que se optó por utilizar Firebase de Google [17], una plataforma que ofrece servicios de base de datos en la nube. Este servicio permite conectar aplicaciones con esa base de datos y actualizaciones a tiempo real.

Por otra parte, Firebase también ofrece un servicio de autenticación, lo cual facilita el desarrollo relacionado con el manejo de usuarios.

Los servicios de Firebase que fueron utilizados serán listados a continuación:

- **Firebase Authenticacion SDK:** Es un conjunto de servicios *backend* para el manejo y autenticación de usuario, los cuales se pueden integrar fácilmente en las aplicaciones, reduciendo el trabajo a la implementación del *frontend* [20].
- **Cloud Firestore:** Es una base de datos NoSQL alojada en la nube a la que puede acceder nuestra aplicación. Los datos se almacenan en documentos en formato JSON. Estos documentos se almacenan en colecciones, que son contenedores para organizar los datos y compilar consultas [21].

Colecciones y estructura de documentos

Como se mencionó anteriormente, los datos se guardan en colecciones y documentos estructurados. Si bien las bases documentales no siguen al pie un modelo de dominio, para

la definición sobre qué colecciones utilizar y la estructura de los documentos a guardar, se utilizó como base lo analizado en la sección 4.2.

De acuerdo a lo observado en la figura 4.1, las entidades identificadas fueron: Usuario, Conexión, DW, Cubo, Dimensión, Jerarquía, Nivel y Consulta. Para modelar estas entidades, fueron utilizados diferentes criterios de acuerdo a los beneficios que nos brinda Firebase, y un esquema documental:

- La entidad Usuario es modelada por el propio servicio de Firebase Authentication, el cual maneja una colección de usuarios registrados en nuestra aplicación, identificados por un UID (universal ID) en todo el proyecto.
- Las entidades Conexión, DW y Consulta son modeladas como colecciones independientes dentro de Cloud Firestore.
- Las entidades Cubo, Dimensión, Jerarquía y Nivel no fueron modeladas como colecciones, ya que basándonos en su estructura jerárquica, es más beneficioso modelarlo como parte de la estructura de los documentos de la colección DW.

En cuanto a las estructuras de los documentos, los usuarios son almacenados en el servicio de Firebase Authentication e identificados por el atributo universal ID (único) y por el correo electrónico. Otros meta-datos que son almacenados son el nombre de usuario y la contraseña (mediante técnicas de cifrado *hash*).

Las conexiones de los usuarios son almacenadas en la colección "Connections" y sus documentos tienen la estructura presentada en 5.1.

```
UID_connection: {
    baseName: " ",
    connectionName: " ",
    dbms: " ",
    hostname: " ",
    port: " ",
    user: " ",
    password: " ",
    userID : UID_user
}
```

Listing 5.1: Estructura del documento *Connection*

El documento de conexión, al igual que todo documento de la base, está identificado por el atributo universal ID (UID) y almacena todos los datos necesarios para conectarse a la base. A este documento se agrega el campo `userID`, que será utilizado para indicar de qué usuario es la conexión almacenada.

La definición del DW (cubo, dimensiones, jerarquías, niveles) es almacenada en la colección "Datawarehouses", y sus documentos tienen la estructura presentada en 5.2.

```
UID_datawarehouse: {
    schemas: [{
        name: " ",
        description: " ",
        cubes: [{
            name: " ",
            description: " ",
            table: " ",
            measureName: " ",
            measureAttribute: " ",
            privateDimensions: [{
                name: " ",
                description: " ",
                table: " ",
                hierarchies: [{
                    name: " ",
                    description: " ",
                    levels: [{
                        name: " ",
                        attrID: " ",
                        other_attrs: [{
                            name: " ",
                            value: " "
                        }]
                    }]
                }]
            }],
            publicDimensions: [{
                referenceCoords: [],
                table: " "
            }]
        }],
        dimensions: [{
            name: " ",
            description: " ",
            table: " ",
            hierarchies: [{
                ...
            }]
        }]
    }],
    userID: UID_user,
    databaseID: UID_connection,
}
```

Listing 5.2: Estructura del documento *Datawarehouse*

Con los datos presentados anteriormente en formato JSON se muestra un ejemplo de cómo se almacenará la definición de un DW. Dentro del esquema se guarda una lista de cubos y una lista de dimensiones compartidas. A su vez cada cubo guarda su lista

de dimensiones privadas, y una lista de referencias a dimensiones compartidas. En las dimensiones privadas se guardará la lista de jerarquías, y a su vez cada jerarquía tiene su lista de niveles. En las dimensiones públicas se guardará la referencia a la dimensión compartida (coordenadas en el JSON). Las dimensiones compartidas (ubicadas fuera de los cubos) seguirán la misma estructura que cualquier dimensión privada de un cubo. Por último, las consultas generadas por el usuario se guardarán en la colección queries y sus documentos tienen la estructura presentada en 5.3.

```
UID_datawarehouse: {
  datawarehouseID: "MunNYyGhHB78G4XsOzLV",
  indexCube: 0,
  name: "Consulta Ejemplo",
  query: "{"levels":[
    {"hierarchy":null,"level":null,"attrId":false,"attrs":[0],"index":0},
    {"hierarchy":null,"level":null,"attrId":false,"attrs":[0],"index":1},
    {"hierarchy":null,"level":null,"attrId":false,"attrs":[0],"index":2},
    {"hierarchy":null,"level":null,"attrId":false,"attrs":[0],"index":3},
    {"hierarchy":null,"level":null,"attrId":false,"attrs":[0],"index":4},
    {"hierarchy":null,"level":null,"attrId":false,"attrs":[0],"index":5}
    ],
    "indexAttrStarts":[0,2,4,6,8,10],
    "children":[],"selected":true,"path":[]
  }"
}
```

Listing 5.3: Estructura del documento *Consulta*

5.2.3. Conexión con la base del usuario

Para la implementación de la conexión con la base de datos del usuario se optó por utilizar la biblioteca *open source* Knex.js [19]. Esta contiene un constructor de consultas SQL para PostgresSQL, MySQL, MariaDB, SQLlite3, Oracle y Amazon Redshift, en sintaxis JavaScript para Node.

El componente que realiza la conexión con la base de datos del usuario, se inicializa mediante los parámetros de configuración de conexión de la base del usuario, y genera una conexión a la base mediante el uso de la biblioteca Knex. Se presenta en 5.4 el código asociado a la creación de la conexión Knex para una base de datos de ejemplo.

```
const knex = require('knex')({
  client: 'dbms',
  connection: {
    host : '127.0.0.1',
    user : 'your_database_user',
    password : 'your_database_password',
    database : 'myapp_test'
  }
```

```
});
```

Listing 5.4: Conexión Knex

5.3. Consultas pyDatalog

Una vez que el usuario finalice la definición de un DW, la aplicación permitirá pasar a la interfaz de Consultas. En dicha interfaz el usuario puede ejecutar dos tipos de consultas: operaciones de *roll-up* sobre los cubos definidos y ejecutar métricas de calidad sobre el DW, visualizando los resultados y el código pyDatalog generado para realizar dichas consultas.

Para prevenir la llegada a esta etapa con un DW mal definido, se decidió desarrollar un módulo de validación, en el que se verifica que la definición del DW sea correcta y en caso contrario, lista los problemas encontrados.

Las validaciones implementadas fueron las siguientes:

- Cubos con el mismo nombre: No pueden existir en la definición de un DW dos o más cubos con el mismo nombre.
- Dimensiones públicas con el mismo nombre: No pueden existir en la definición de un DW dos o más dimensiones con el mismo nombre.
- Dimensiones privadas con el mismo nombre: No pueden existir en la definición dos o más dimensiones privadas con el mismo nombre, dentro de un mismo cubo.
- Dimensión privada y pública referenciada con el mismo nombre: No pueden existir en la definición una o más dimensiones privadas y una o más dimensiones públicas con el mismo nombre, dentro del mismo cubo.
- Jerarquías con el mismo nombre: No pueden existir en la definición dos o más jerarquías con el mismo nombre, dentro de la misma dimensión.
- Niveles con el mismo nombre: No pueden existir en la definición dos o más niveles con el mismo nombre, dentro de la misma jerarquía.
- Atributo identificador coincidente: El nivel más bajo de todas las jerarquías dentro de la misma dimensión debe tener el mismo atributo identificador.
- Existencia de atributo identificador: El atributo identificador del nivel más bajo de cada dimensión, debe existir entre los atributos de la tabla asociada al cubo.
- Tablas asociadas coincidentes: La tabla asociada a cualquiera de las dimensiones pertenecientes a un cubo, no debe coincidir con la tabla asociada a dicho cubo.

Una vez que la definición del DW cumpla con las validaciones mencionadas, se le permitirá al usuario acceder a la interfaz de Consultas/Calidad. La implementación de dicha interfaz cumple con los casos de uso 12 (generación de código pyDatalog utilizado en las operaciones 4.3), 13 (ejecución de consultas *roll-up* 4.4), 14 (guardar consultas A.10) y 15 (ejecución de métricas de calidad 4.5).

Para realizar la ejecución de las operaciones de *roll-up* y las métricas de calidad, desde Node.js es necesario ejecutar un subproceso en un intérprete de Python, ya sea ejecutando un código generado dinámicamente o transfiriendo la lógica al código escrito en lenguaje Python. Se genera inicialmente el código pyDatalog referente a la definición del DW y el cubo base para luego generar el código asociado a las operaciones de *roll-up* indicadas o métricas ejecutadas.

Dada una operación de *roll-up*, el código pyDatalog desplegado por la aplicación en el resultado incluye el código generado asociado a la definición del DW y las operaciones de *roll-up* previas, sin embargo el código ejecutado es únicamente el asociado a la operación de *roll-up* correspondiente.

En las siguientes secciones se describen las decisiones a nivel de implementación, tomadas para cumplir con las funcionalidades requeridas.

5.3.1. Tecnologías utilizadas

Las tecnologías que fueron utilizadas para la construcción y ejecución de las consultas en lenguaje pyDatalog son las siguientes:

- **Python [15]:** Lenguaje utilizado para implementar, mediante la biblioteca pyDatalog [25], las consultas al DW definido por el usuario
- **Node.js [9]:** Entorno de ejecución donde se ejecutan, a partir de un código Python desarrollado por la aplicación, las consultas definidas.

5.3.2. Implementación del Data Warehouse

Para la ejecución de operaciones de *roll-up* y métricas de calidad es necesario primero generar el código pyDatalog para el DW definido por el usuario. A continuación se describe la implementación en pyDatalog para el DW presentado en la sección 3.1.

El primer paso de la implementación consiste en establecer la conexión a la base de datos y crear una sesión utilizando SQLAlchemy, para proporcionar recursos de consulta potentes en bases de datos relacionales. La sesión es un espacio de trabajo local para los objetos obtenidos de una conexión de base de datos [25]. En el código B.1 se establece la conexión con la base de datos y se genera la sesión.

En el próximo paso se crean las clases correspondientes a las dimensiones y hechos definidas, en este caso: *Time*, *Price*, *Shop*, *Location*, *Author*, *Book* y *Sales*. Estas clases heredan las propiedades de pyDatalog y SQLAlchemy. Los atributos se definen a partir del esquema de las tablas de la base de datos. En el código B.2 se definen las clases para las dimensiones y la tabla de hechos mencionadas.

Luego se crea una serie de predicados accesibles desde las consultas de *roll-up* que chequean la relación de jerarquía, en el código B.3 se presentan los predicados de jerarquía

para nuestro caso de estudio. Por último, en el código B.4 se realizan a modo de ejemplo dos consultas pyDatalog, una para obtener el cubo base, y otra con la operación *roll-up* agrupando los elementos de la dimensión *Time* del nivel *Date* al nivel *Month*.

5.3.3. Operaciones de roll-up

Una vez generado el código pyDatalog del DW y de su cubo base, el usuario puede generar las vistas de cubo asociadas a las operaciones de *roll-up*.

En esta sección se abarcan todos los aspectos referentes a la implementación para la ejecución en pyDatalog de las operaciones de *roll-up* sobre el DW definido.

Selección de la operación

Como se observa en la figura 5.1, la interfaz para la selección de la operación de *roll-up* a ejecutar consta de dos componentes: el panel lateral y el árbol de consultas.

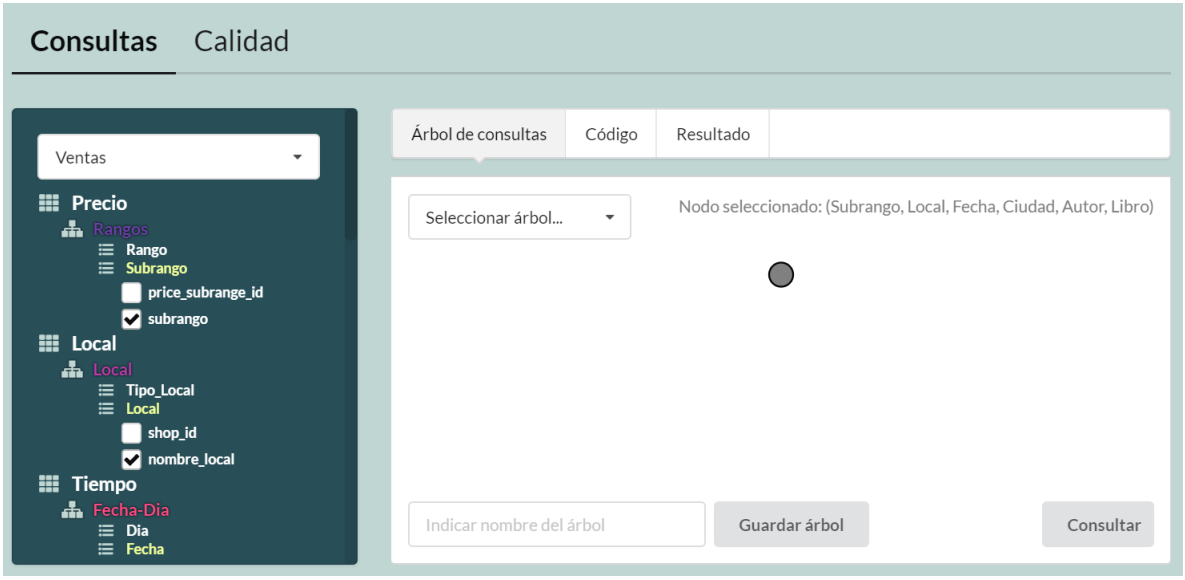


Figura 5.1: Árbol de consultas inicial y panel lateral.

En el árbol de consultas, cada nodo representa una vista de cubo seleccionado en el panel lateral (en este caso Ventas) y se corresponde con una serie de operaciones de *roll-up*. Inicialmente el árbol consta de un solo nodo, el cual corresponde al cubo base formado por el nivel más bajo de cada dimensión del cubo.

En el panel lateral se muestra la definición del cubo (dimensiones, jerarquías y niveles). Los niveles de la vista del cubo asociado al nodo actual aparecen resaltados y se permite seleccionar qué campos se quieren mostrar en el resultado luego de ejecutar la consulta.

Desde el panel se permite seleccionar una jerarquía habilitada para realizar la operación *roll-up* según el nodo actual (vista de cubo). Al seleccionar una jerarquía, se genera un nuevo nodo que representa la vista de cubo resultante de ejecutar *roll-up* por dicha jerarquía y automáticamente el nodo actual pasará a ser el nodo resultante de la operación.

En la figura 5.2 se muestra un árbol de consultas con varios nodos. En las aristas se indica la dimensión por la que se hace *roll-up*. Si posicionamos el puntero sobre la arista se muestra además la jerarquía por la que se hace el *roll-up*.

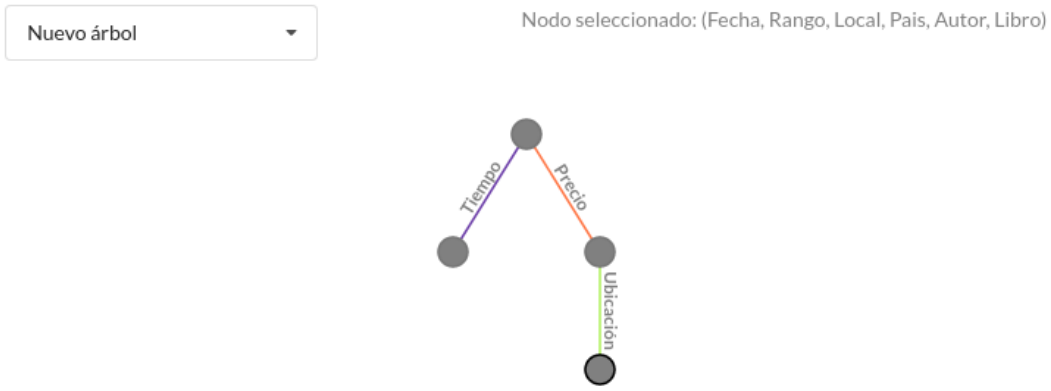


Figura 5.2: Árbol de consultas con varios nodos

Dado un nodo seleccionado en el árbol de consultas, se puede ejecutar pulsando en el botón "Consultar". Luego de terminar la consulta en pyDatalog, el nodo se vuelve blanco y en las pestañas de resultado y código generado, mostradas en la figura 5.3, se carga el resultado correspondiente y el código pyDatalog utilizado para ejecutar la consulta. Si nodos anteriores al nodo a ejecutar no están ejecutados se ejecutan en orden hasta el nodo que el usuario eligió ejecutar.

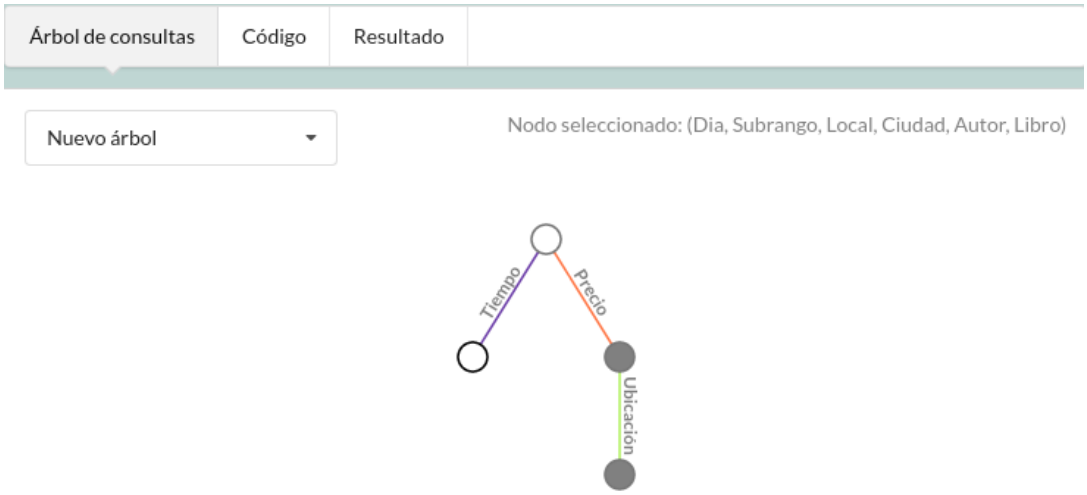


Figura 5.3: Árbol luego de ejecutar una consulta.

Para la vista de árbol se utiliza una librería llamada D3.js [22] que se utiliza generalmente en la manipulación de SVG dentro del DOM. La herramienta se puede utilizar mediante las funciones de cálculos de D3.js con la manipulación de DOM propia de React, como en nuestro caso o transfiriéndole directamente el control a D3.js.

Generación de consultas

Para ejecutar la consulta se genera el código correspondiente en Python y se le asigna un identificador numérico. Cada consulta de *roll-up* utiliza como partida la vista de cubo de la consulta anterior a través del identificador numérico. El código generado también necesita información de la conexión, de la definición del DW y el identificador de la jerarquía por la que se realiza la operación, para poder ser ejecutado.

Para el caso particular de la consulta inicial la vista de cubo de partida sería el mismo cubo base y no es necesario un identificador de jerarquía.

Se utiliza la forma de template string de JavaScript para escribir el código de forma de minimizar lo más posible la diferencia respecto al resultado final.

La consulta generada selecciona, además de los identificadores de las tablas de dimensiones correspondientes a los niveles de la consulta, todos los atributos que se definieron para esos niveles, y que el usuario posteriormente puede elegir, ver u ocultar.

Ejecución de consultas

La ejecución se utiliza para obtener las vistas de cubos. Antes de ejecutar la primer consulta es necesario ejecutar el código inicial en el que se realiza la conexión con la base de datos y se definen las tablas en pyDatalog, entre otras cosas.

Para la ejecución se utiliza una librería llamada PythonShell [23], que simplifica la utilización de child_process (librería estándar de NodeJS para crear subprocesos en ejecución) cuando el subproceso se trata de la ejecución de un script escrito en Python. Se utiliza una misma instancia del subproceso por sesión mientras no se cambie de cubo o de árbol de consultas. De este modo, para ejecutar una consulta no es necesario ejecutar nuevamente todas las consultas anteriores, pues las vistas de cubos correspondientes ya están en la memoria del subproceso.

La ejecución de las consultas inicialmente era muy costosa, por lo que se tomaron dos medidas para la mejora del rendimiento: las referencias de las tablas son procesadas en orden ascendente de tamaño, y cada vista de cubo es cargada en memoria en un arreglo (utilizado de forma similar a una memoria caché) y luego se crea un nuevo predicado de la vista de cubo asociado a ese arreglo para hacer referencia en las posteriores consultas. En la sección 6.3 se presentarán en profundidad las correcciones realizadas y los mejoras obtenidas en el rendimiento al ejecutar consultas.

Código pyDatalog y resultado

El código generado resultado de la consulta es de utilidad para que el usuario pueda verificar cómo se realizó la misma, y si quisiera, para que pueda ejecutarla manualmente en un interprete de Python 3. Para la visualización del código se utilizó la librería highlight.js [24], que permite resaltar código escrito en muchos lenguajes, entre ellos Python.

El código generado es el código necesario para realizar la consulta seleccionada, las consultas anteriores de las que depende y el código inicial (declaración de importaciones de bibliotecas externas, declaración de conexiones a la base de datos del usuario y la sesión actual del engine) para realizar todas las consultas. Esto significa que si se realizaron otras consultas que no dependen de la consulta seleccionada, no serán desplegadas en el código generado. Un ejemplo de código generado se muestra en el anexo B.1.

En la pestaña de resultado, se muestra el resultado obtenido de ejecutar la vista de cubo asociada al nodo seleccionado. La vista de cubo es desplegada como una tabla que tiene las columnas ordenadas de modo que el identificador de cada nivel va seguido de los atributos asociados al correspondiente nivel. Las columnas siempre están ordenadas de modo que los niveles correspondientes a una dimensión siempre van en la misma posición relativa a los de otras dimensiones. El usuario puede elegir qué columnas de atributos quiere que se muestren para cada nivel. Por defecto, se oculta el identificador del nivel, pero puede mostrarse.

5.3.4. Métricas de calidad

En esta sección se detalla la implementación realizada, la cual fue de dos de las métricas de calidad definidas en la sección 3.2.2, con la posibilidad de extender la aplicación agregando otras métricas. Las métricas implementadas son la de sumarizabilidad y de totalidad, cuyos cálculos son obtenidos a través de consultas pyDatalog.

Sumarizabilidad

La métrica de sumarizabilidad implementada brinda una comparación celda a celda de la medida para dos vistas de cubo con la misma granularidad. Si existe una medida para una vista pero no para la otra, también lo informa. Si las dos vistas de cubo no son iguales es seguro que no se cumple la sumarizabilidad, pero si son iguales no es posible asegurar que se cumpla.

La métrica retorna para cada celda, los atributos identificadores, la medida correspondiente de cada vista de cubo en caso de existir y en caso contrario, un valor nulo, y el resultado de la comparación de las medidas, que en caso de existir y coincidir es 1, sino 0.

Asumiendo que fueron realizadas dos consultas que refieren a dos vistas de cubo distintas con la misma granularidad, asociadas a los predicados `cubeView1` y `cubeView2`, y a modo de simplificar, estos predicados sólo tienen un atributo y un valor. La definición del predicado de la consulta en pyDatalog sería el código mostrado en 5.5.

```
(sumarizability[Attr] == (Cant1, '-', 0)) <= (cubeView1[Attr] == Cant1)
(sumarizability[Attr] == ('-', Cant2, 0)) <= (cubeView2[Attr] == Cant2)
(sumarizability[Attr] == (Cant, Cant, 1)) <= (
    (cubeView1[Attr] == Cant) &
    (cubeView2[Attr] == Cant)
)
(sumarizability[Attr] == (Cant1, Cant2, 0)) <= (
    (cubeView1[Attr] == Cant1) &
    (cubeView2[Attr] == Cant2) &
    ~(Cant1 == Cant2)
)
```

Listing 5.5: Ejemplo de consulta de la métrica de sumarizabilidad

La definición anterior corresponde al predicado `sumarizability`, que en lenguaje natural podríamos expresarlo como: para cada celda tenemos una tripleta con las medidas correspondientes a las dos vistas de cubo (en caso de no existir alguna, muestra un '-') y el valor de nuestra métrica de sumarizabilidad.

Totalidad

Esta métrica chequea la totalidad tomando la tabla asociada a una dimensión y chequeando los valores nulos para el nivel seleccionado. En caso de no existir nulos en la columna que identifica a ese nivel, podemos asegurar totalidad. Esto se debe a que al no existir nulos, todos los elementos del nivel anterior al seleccionado pueden agruparse.

Se tomó la decisión de retornar una tabla con la cantidad de valores nulos y no nulos. Se retornan también los identificadores de los valores no nulos. Dada una tabla de dimensión `dimension`, un identificador del nivel `id_level`, el predicado `completeness` de pyDatalog se define en el código mostrado en 5.6.

```
(completeness['Nulos'] == len_(X)) <= (
    (dimension.id_level[X] == ID) &
    (ID == -1)
)
(completeness['No nulos'] == len_(X)) <= (
  (dimension.id_level[X] == ID) &
  (ID != -1)
)
(completeness[ID] == len_(X)) <= (
  (dimension.id_level[X] == ID) &
  (ID != -1)
)
```

Listing 5.6: Ejemplo de consulta de la métrica de totalidad

Es importante resaltar que los identificadores -1 representan a los nulos.

Capítulo 6

Experimentación

En este capítulo se presentan las pruebas realizadas para chequear el correcto funcionamiento de las consultas pyDatalog asociadas a las vistas de cubos y al cálculo de las métricas de calidad implementadas. Se presentan también consideraciones de rendimiento que surgieron al realizar las pruebas, las cuales implicaron cambios en la lógica de algunas funcionalidades con el objetivo de mejorar los tiempos de ejecución.

Las pruebas fueron realizadas utilizando el DW definido en la sección 3.1 y con un equipo con las siguientes características:

- Sistema operativo: Windows 11
- Procesador: Intel(R) Core(TM) i5-8265U CPU @ 1.60GHz 1.80 GHz
- Memoria: 8.00 GB
- Disco duro: 256 GB sólido

La base de datos utilizada para la experimentación utiliza PostgreSQL versión 12, y sus tablas tienen las siguientes cantidades de filas:

- Sales: 448
- Time: 104
- Authors: 500
- Books: 589
- Price: 14
- Shops: 50
- Geographic_location: 139

6.1. Pruebas de la página de consultas

La primer prueba realizada se hizo para la consulta pyDatalog correspondiente al cubo base. Para ello se crea una tabla auxiliar en la base de datos con el código que se muestra en 6.1 para alojar el resultado de la consulta obtenido en la aplicación.

```
create table query_without_atributes (
    time_id bigint not null,
    price_id bigint not null,
    shop_id bigint not null,
    geographic_location_id bigint not null,
    author_id bigint not null,
    book_id bigint not null,
    quantity bigint not null,
    constraint query_without_atributes_pkey primary key (time_id,
   price_id, shop_id, geographic_location_id, author_id, book_id)
)
```

Listing 6.1: Tabla para guardar resultado de una consulta.

Luego se ejecuta el nodo de la consulta inicial en la aplicación y se carga el resultado obtenido en la tabla auxiliar 6.1. Para corroborar que el resultado sea el correcto, se ejecuta una consulta SQL que devuelva el mismo resultado, pero directamente sobre la base de datos. Se chequea que efectivamente sean iguales a través de una doble inclusión.

```
select date_id, price_subrange_id, shop_id, city_id, author_id, book_id,
    quantity from sales
```

Listing 6.2: Consulta SQL para comparar con el resultado de la aplicación.

Para chequear el correcto funcionamiento de las operaciones de *roll-up*, en el árbol de consultas de la aplicación, seleccionamos uno de los nodos y ejecutamos su correspondiente consulta. Los datos obtenidos luego de ejecutar esta consulta se cargan en la tabla auxiliar 6.1, habiendo limpiado la misma previamente.

Utilizamos la misma lógica de construir una consulta SQL que dé el mismo resultado y chequeamos la doble inclusión para saber que son iguales. La consulta SQL en este caso, donde por la aplicación se hizo *roll-up* de *Date* a *Month* y de *Month* a *Trimester* por la dimensión *Time* y de *Book* a *Language* por la dimensión *Book*, sería la siguiente:

```
select trimester_id, price_subrange_id, shop_id, city_id, author_id,
   book_language_id, sum(quantity) from sales, time, books
where sales.date_id = time.date_id
and sales.book_id = books.book_id
and month_id <> -1
and trimester_id <> -1
and book_language_id <> -1
group by trimester_id, price_subrange_id, shop_id, city_id, author_id,
   book_language_id
```

Listing 6.3: Consulta SQL para comparar con el resultado de la aplicación.

Al realizar estas comparaciones, se verificó que se obtienen los mismos resultados mediante consultas pyDatalog dentro de la aplicación, que directamente por la base de datos correspondiente mediante consultas SQL.

6.2. Pruebas de la página de métricas de calidad

En estas pruebas se chequea el correcto funcionamiento de las métricas de calidad implementadas en la aplicación: Sumarizabilidad y Totalidad. Para cada una se crea una tabla auxiliar en la base de datos para alojar el resultado de la consulta en la aplicación. Las definiciones SQL de las tablas se muestran en los códigos 6.4 y 6.5.

```
create table sumarizability (
    time_id bigint not null,
    price_id bigint not null,
    shop_id bigint not null,
    geographic_location_id bigint not null,
    author_id bigint not null,
    book_id bigint not null,
    quantity1 bigint not null,
    quantity2 bigint not null,
    metric bigint not null,
    constraint sumarizability_pkey primary key (time_id, price_id,
   shop_id, geographic_location_id, author_id, book_id)
)
```

Listing 6.4: Tabla para guardar el resultado de aplicar la métrica de sumarizabilidad.

```
create table totality (
    dimension_id text not null primary key,
    counter bigint not null
)
```

Listing 6.5: Tabla para guardar el resultado de aplicar la métrica de totalidad.

Como ejemplo de consulta de Sumarizabilidad, tomamos la vista de cubo correspondiente a las operaciones de *roll-up* de *Date* a *Month*, de *Month* a *Trimester* y de *Trimester* a *Semester* por la dimensión *Time*. Y una segunda vista de cubo, obtenida de hacer *roll-up* de *Date* a *Month*, de *Month* a *Bimester* y de *Bimester* a *Semester* por la dimensión *Time*. El resultado obtenido al ejecutar por la aplicación la Sumarizabilidad para esas dos vistas de cubo, lo cargamos en la tabla auxiliar 6.4

Para comparar utilizamos la consulta SQL mostrada en 6.6 y como antes, chequeamos la doble inclusión para saber que son iguales.

```
select x1.*, x2.quantity2, case when x1.quantity1 = x2.quantity2 then 1
   else 0 end from (
  select semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id, sum(quantity) as quantity1 from sales, time
  where sales.date_id = time.date_id
  and month_id <> -1
  and trimester_id <> -1
  and semester_id <> -1
  group by semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id
```

```
) as x1, (
  select semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id, sum(quantity) as quantity2 from sales, time
  where sales.date_id = time.date_id
  and month_id <> -1
  and bimester_id <> -1
  and semester_id <> -1
  group by semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id
) as x2
where x1.semester_id = x2.semester_id
and x1.price_subrange_id = x2.price_subrange_id
and x1.shop_id = x2.shop_id
and x1.city_id = x2.city_id
and x1.author_id = x2.author_id
and x1.book_id = x2.book_id
union
select x1.*, 0 from (
  select semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id, sum(quantity) as quantity1, -1 quantity2 from sales, time
  where sales.date_id = time.date_id
  and month_id <> -1
  and trimester_id <> -1
  and semester_id <> -1
  group by semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id
) as x1
where (semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id) not in (
  select semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id from sales, time
  where sales.date_id = time.date_id
  and month_id <> -1
  and bimester_id <> -1
  and semester_id <> -1
  group by semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id
)
union
select x2.*, 0 from (
  select semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id, -1 quantity1, sum(quantity) as quantity2 from sales, time
  where sales.date_id = time.date_id
  and month_id <> -1
  and bimester_id <> -1
  and semester_id <> -1
  group by semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id
) as x2
where (semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id) not in (
  select semester_id, price_subrange_id, shop_id, city_id, author_id,
   book_id from sales, time
  where sales.date_id = time.date_id
  and month_id <> -1
```

```
52   and trimester_id <> -1
53   and semester_id <> -1
54   group by semester_id, price_subrange_id, shop_id, city_id, author_id,
      book_id
55 )
```

Listing 6.6: Consulta SQL para comparar con el resultado de la aplicación.

Al realizar la comparación de los resultados obtenidos por la aplicación con los de la consulta SQL 6.6 se obtiene que los mismos coinciden.

Para probar la métrica de Totalidad seguimos el mismo procedimiento. Luego de sustituir los '-' por -1 del resultado obtenido de ejecutar la métrica por la aplicación, se carga en la tabla auxiliar 6.5 y se compara con el resultado obtenido en la consulta SQL mostrada en 6.7

```
1 select 'Nulos' Trimestre, count(trimester_id) from time
2 where trimester_id = -1
3 union
4 select 'No nulos' Trimestre, count(trimester_id) from time
5 where trimester_id <> -1
6 union
7 select trimester_id::text Trimestre, count(trimester_id) from time
8 where trimester_id <> -1
9 group by trimester_id
```

Listing 6.7: Consulta SQL para comparar con el resultado de la aplicación.

Al realizar la comparación de los resultados obtenidos por la aplicación con los de la consulta SQL 6.7 se obtiene que los mismos coinciden.

6.3. Consideraciones de rendimiento

Originalmente las consultas demoraban mucho por lo que tuvimos que investigar el funcionamiento interno de la ejecución de las consultas en PyDatalog, lo que nos llevó a reordenar los miembros de las consultas de modo que los miembros estuvieran ordenados por la cantidad de registros que se chequean, de menor a mayor. De este modo el tiempo de ejecución de las consultas de la página de consultas pasó a ser razonable.

La cantidad de registros de cada tabla se obtiene una vez que el usuario realiza una conexión exitosa contra una de sus conexiones definidas. En ese momento el sistema realiza una cantidad de consultas a la base de datos para obtener diversos datos como por ejemplo: nombres de tablas, nombres de columnas, y cantidad de registros por tabla. Estos datos son almacenados en un estado global de la aplicación y es utilizado en todas las operaciones y consultas que se realizan al momento de generar el código pyDatalog. En el código 6.8 se presenta un ejemplo de consulta sobre una base de datos.

```
SELECT relname as table_name,reltuples as size
FROM pg_class C
LEFT JOIN pg_namespace N ON (N.oid = C.relnamespace)
JOIN information_schema.tables ON (information_schema.tables.table_name
   = C.relname)
WHERE table_catalog = ? AND nspname NOT IN ('pg_catalog', '
   information_schema') AND relkind='r'
```

Listing 6.8: Consulta para obtener cantidad de registros en las tablas de la base de datos

El siguiente problema que se presentó fue que al realizar la consulta de sumarizabilidad no era posible hacer que su ejecución finalizara. Entendemos que lo que sucedía era que el orden de complejidad de la consulta había aumentado respecto a las consultas anteriores. Como no podíamos disminuir el orden de cómputo, supusimos que tal vez la operación que era costosa en tiempo de ejecución era la consulta a base de datos, por lo que la sustituimos por consultas a memoria de la aplicación, que consideramos que podría ser más liviana. Efectivamente haciendo ese cambio, la consulta de sumarizabilidad concluyó y además el tiempo fue sensiblemente menor.

Como en las consultas anteriores también se podía realizar esta sustitución, lo llevamos a cabo, y de este modo conseguimos que todas las consultas se ejecutaran casi de forma inmediata, exceptuando la consulta del cubo base que al no hacerse en base a otra consulta, se realiza de la manera original. Las consultas a memoria se realizaron a través de un nuevo predicado el cual será detallado en B.2.

Capítulo 7

Conclusiones y trabajos a futuro

En este capítulo se presentan las conclusiones obtenidas luego de culminar el proyecto, así como algunos aspectos que consideramos importantes para desarrollar a futuro, que quedaron por fuera del alcance del mismo.

7.1. Conclusiones

En el capítulo inicial del proyecto 1.1 se definió como objetivo principal la implementación de una herramienta que permita evaluar mediante métricas de calidad, aspectos de la calidad de un DW particular. Fue posible cumplir con el objetivo planteado, obteniendo los siguientes resultados:

- Se desarrolló una herramienta que permite al usuario establecer una conexión con su base de datos y definir un DW junto con sus cubos asociados.
- La herramienta permite al usuario ejecutar operaciones de *roll-up* sobre los cubos definidos, con la posibilidad de visualizar los resultados y obtener el código pyDatalog asociado utilizado para la ejecución de dichas operaciones.
- Se definieron y formalizaron métricas de calidad que permiten la evaluación de la calidad del DW: Sumarizablidad, Totalidad y Jerarquía N:1.
- Se implementaron en la herramienta las métricas de Sumarizabilidad y Totalidad, de forma que el usuario pueda ejecutarlas para su DW, obteniendo los resultados y el código pyDatalog asociado.

El desarrollo se realizó sobre tecnologías modernas y muy utilizadas, lo que permite un fácil entendimiento del código fuente para que otro desarrollador pueda trabajar sobre el mismo. Se desarrolló una aplicación de escritorio que tiene como una de sus características la flexibilidad de elección de motor de bases de datos permitiendo la conexión a PostgresSQL, MYSQL y OracleDB, entre otros. Además permite la ejecución de la herramienta en cualquier sistema operativo, sin necesidad de realizar desarrollo nativo.

El sistema funciona con sesiones de usuario, donde cada uno puede tener sus propias conexiones. En cada conexión el usuario puede definir el DW asociado, y para cada uno se pueden guardar las operaciones *roll-up* realizadas.

Algunos de los principales desafíos que tuvimos que abordar fueron la definición formal de métricas de calidad asociadas a la calidad del DW y la mejora de performance de las consultas realizadas en pyDatalog.

7.2. Trabajo a futuro

Durante el desarrollo del proyecto fueron detectadas ciertas mejoras que podrían agregarse a la herramienta implementada y algunos aspectos que quedaron por fuera del alcance del mismo. Las mejoras detectadas son las siguientes:

- Implementación de la métrica de calidad Jerarquía N:1
- Formalización e implementación de otras métricas de calidad.
- Incluir al contexto como parte de la definición del DW y la evaluación de las métricas de calidad implementadas. Para resolver esto, se podría agregar una interfaz donde el usuario pueda subir una ontología que represente el contexto, y tenga una forma de mapearlo con la definición del DW.

Referencias bibliográficas

[1] http://datalog.sourceforge.net/datalog.html. [Web accedida el 12-11-2021].

[2] https://www.electronjs.org/. [Web accedida el 12-6-2021].

[3] https://es.reactjs.org/. [Web accedida el 12-6-2021].

[4] https://react.semantic-ui.com/. [Web accedida el 12-6-2021].

[5] https://www.javascript.com/. [Web accedida el 12-6-2021].

[6] https://html.com/. [Web accedida el 12-6-2021].

[7] https://www.w3.org/Style/CSS/Overview.en.html. [Web accedida el 12-6-2021].

[8] https://sass-lang.com/. [Web accedida el 12-6-2021].

[9] https://nodejs.org/es/. [Web accedida el 12-6-2021].

[10] https://es.redux.js.org/. [Web accedida el 12-6-2021].

[11] https://www.chromium.org/Home. [Web accedida el 12-6-2021].

[12] https://v8.dev/. [Web accedida el 12-6-2021].

[13] https://webassembly.github.io/spec/core/. [Web accedida el 12-6-2021].

[14] https://tc39.es/ecma262/. [Web accedida el 12-6-2021].

[15] https://www.python.org/. [Web accedida el 12-6-2021].

[16] https://www.docker.com/. [Web accedida el 12-6-2021].

[17] https://firebase.google.com/?hl=es. [Web accedida el 12-6-2021].

[18] https://www.sqlalchemy.org/. [Web accedida el 19-02-2022].

[19] http://knexjs.org/. [Web accedida el 12-6-2021].

[20] https://firebase.google.com/docs/auth. [Web accedida el 12-6-2021].

[21] https://firebase.google.com/docs/firestore. [Web accedida el 12-6-2021].

[22] https://d3js.org/. [Web accedida el 12-6-2021].

[23] https://www.npmjs.com/package/python-shell. [Web accedida el 12-6-2021].

[24] https://highlightjs.org/. [Web accedida el 12-6-2021].

[25] https://sites.google.com/site/pydatalog/3—datalog-and-data-integration.

[26] Batini, C., and Scannapieco, M. Data and information quality, 1 ed. Data-Centric Systems and Applications. Springer International Publishing, Basel, Switzerland, June 2016.

[27] Carpani, F. CMDM: un método conceptual para la especificación de bases multidimensionales. Master's thesis, Facultad de Ingeniería. Instituto de Computación – PEDECIBA, 2000.

[28] Hurtado, C. A., and Mendelzon, A. O. Reasoning about Summarizability in Heterogeneous Multidimensional Schemas. In Database Theory — ICDT 2001 (Berlin, Heidelberg, 2001), J. Van den Bussche and V. Vianu, Eds., Lecture Notes in Computer Science, Springer, pp. 375–389.

[29] Hurtado, C. A., and Mendelzon, A. O. Olap dimension constraints. In Proceedings of the Twenty-first ACM SIGMOD-SIGACT (New York, NY, USA: ACM, 2002), pp. 169–179.

[30] Inmon, W. H. Building the data warehouse, 3 ed. John Wiley & Sons, New York, apr 2002.

[31] Sanz, C. Context based Data Quality Rules for Multidimensional Data. PhD thesis, PEDECIBA Informática. Instituto de Computación. Facultad de Ingeniería. UdelaR. Trabajo en curso.

[32] Tarín Morales, F. Optimización de programas Datalog basada en estrategias. Master's thesis, Universitat Politècnica de València, 7 2011.

[33] Vaisman, A., and Zimanyi, E. Data warehouse systems: design and implementation, 2014 ed. Data-Centric Systems and Applications. Springer, Berlin, Germany, Sept. 2014.

ANEXOS

Anexo A

Casos de uso

01 - Crear sesión

Identificador: 01
Nombre: Crear sesión
Objetivo: Dar de alta una nueva sesión en el sistema
Actores: Usuario
Precondiciones: No se ha iniciado sesión
Descripción: Un usuario desea crear una nueva sesión para utilizar el sistema.
Flujo normal: 1. Usuario selecciona la opción de crear nueva sesión. 2. Sistema despliega formulario con los siguientes campos: - Correo asociado (requerido), Contraseña (requerido) 3. Usuario completa los campos y selecciona la opción de aceptar. 4. Sistema redirige al usuario a la pantalla de configuración de conexión. 5. Fin del caso de uso.
Flujos alternativos: 3A. Nombre de sesión ya existe en el sistema: 3A.1. Sistema despliega alerta que el nombre de sesión ya existe. 3A.2. Volver al paso 3.
Poscondiciones: Se carga la sesión para los datos ingresados.

Tabla A.1: Crear sesión

02 - Iniciar sesión

Identificador: 02
Nombre: Iniciar sesión
Objetivo: Abrir la sesión de un usuario en el sistema
Actores: Usuario
Descripción: Un usuario desea ingresar a su sesión para utilizar el sistema.
Flujo normal: 1. Usuario selecciona la opción de iniciar sesión. 2. Sistema despliega formulario con los siguientes campos: - Nombre de sesión (requerido) - Contraseña (requerido) 3. Usuario completa los campos y selecciona la opción de ingresar. 4. Sistema inicia la sesión y redirige al usuario a su pantalla principal. 5. Fin del caso de uso.
Flujos alternativos: 3A. Datos ingresados incorrectos: 3A.1. Sistema despliega alerta que los datos son incorrectos. 3A.2. Volver al paso 3.
Poscondiciones: Se carga la sesión para los datos ingresados.

Tabla A.2: Iniciar sesión

03 - Cerrar sesión

Identificador: 03
Nombre: Cerrar sesión
Objetivo: Cerrar la sesión de un usuario en el sistema
Actores: Usuario
Precondiciones: Se ha iniciado sesión
Descripción: Un usuario desea cerrar su sesión en el sistema.
Flujo normal: 1. Usuario selecciona la opción de cerrar sesión. 2. Sistema cierra la sesión y redirige al usuario al inicio de sesión. 3. Fin del caso de uso.
Poscondiciones: Se cierra la sesión actual en el sistema

Tabla A.3: Cerrar sesión

05 - Conectarse a base de datos

Identificador: 05
Nombre: Conectarse a base de datos
Objetivo: Conectarse a la base de datos seleccionada
Actores: Usuario
Precondiciones: Se ha configurado alguna conexión
Descripción: Un usuario quiere conectarse a una conexión de base de datos configurada, el sistema establece la conexión.
Flujo normal: 1. Usuario selecciona la conexión y presiona el botón conectar. 2. Sistema establece la conexión a la base de datos. 3. Fin del caso de uso.
Flujos alternativos: 2A. No se pudo establecer la conexión: 2A.1. Sistema despliega el mensaje de error correspondiente 2A.2. Fin del caso de uso
Poscondiciones: Se establece una conexión a base de datos

Tabla A.4: Conectarse a base de datos

06 - Borrar conexión a base de datos

Identificador: 06
Nombre: Borrar conexión a base de datos
Objetivo: Eliminar una conexión a base de datos
Actores: Usuario
Precondiciones: Se ha configurado alguna conexión
Descripción: Un usuario quiere eliminar una conexión de base de datos existente, el sistema elimina la conexión y todas las definiciones asociadas a la misma.
Flujo normal: 1. Usuario selecciona la opción de borrar una conexión. 2. Sistema despliega la lista de conexiones asociadas a la sesión. 3. Usuario selecciona la conexión y presiona el botón borrar. 4. Sistema despliega un mensaje de confirmación, alertando al usuario que se borrará todo el progreso de la definición del DW asociado. 5. Usuario confirma que desea borrar la conexión . 6. Sistema elimina la conexión seleccionada . 7. Fin del caso de uso.
Flujos alternativos: 5A. Usuario cancela la operación: 5A.1. Vuelve al paso 7
Poscondiciones: Se elimina una conexión a base de datos

Tabla A.5: Borrar conexión a base de datos

08 - Definir Cubo

Identificador: 08
Nombre: Definir Cubo
Objetivo: Definir un cubo con su tabla de hechos y medida
Actores: Usuario
Descripción: Un usuario define la tabla de hechos y medida correspondiente.
Flujo normal: 1. Usuario selecciona la opción para definir un Cubo. 2. Sistema despliega los campos correspondientes: - Nombre del cubo (requerido) - Descripción (opcional) - Medida (requerido) - Tabla asociada (requerido) 3. Usuario completa los datos y selecciona la opción de guardar. 4. «incluye» Definir Dimensión. 5. Fin del caso de uso.
Flujos alternativos: 5A. Usuario desea agregar otra dimensión: 5A.1. Vuelve al paso 4
Poscondiciones: Se guarda el cubo en la definición del DW

Tabla A.6: Definir Cubo

09 - Definir Dimensión

Identificador: 09
Nombre: Definir Dimensión
Objetivo: Definir una dimensión del DW
Actores: Usuario
Precondiciones: Se ha creado un cubo
Descripción: Un usuario define una dimensión del DW, con su correspondiente tabla asociada de la base de datos
Flujo normal: 1. Usuario selecciona la opción de definir Dimensión. 2. Sistema despliega los campos para definir la dimensión: - Nombre (requerido) - Tabla asociada de la base de datos (requerido) 3. Usuario completa los datos y selecciona la opción de guardar. 4. «incluye» Definir Jerarquía. 5. Fin del caso de uso.
Flujos alternativos: 5A. Usuario desea agregar otra jerarquía: 5A.1. Vuelve al paso 4
Poscondiciones: Se guarda la dimensión en la definición del DW

Tabla A.7: Definir Dimensión

10 - Definir Jerarquía

Identificador: 10
Nombre: Definir Jerarquía
Objetivo: Definir jerarquías del DW
Actores: Usuario
Precondiciones: Se ha creado una dimensión
Descripción: Un usuario define una jerarquía dada la dimensión actual.
Flujo normal: 1. Usuario selecciona la opción de definir Jerarquía. 2. Sistema despliega los campos para definir la jerarquía: - Nombre (requerido) - Descripción (opcional) 3. Usuario completa los datos y selecciona la opción de guardar. 4. «incluye» Definir Nivel. 5. Fin del caso de uso.
Flujos alternativos: 5A. Usuario desea agregar otro nivel: 5A.1. Vuelve al paso 4
Poscondiciones: Se guarda la jerarquía en la definición del DW

Tabla A.8: Definir Jerarquía

11 - Definir Nivel

Identificador: 11
Nombre: Definir Nivel
Objetivo: Definir nivel dentro de una jerarquía del DW
Actores: Usuario
Precondiciones: Se ha creado una jerarquía
Descripción: Un usuario define un nivel dada la jerarquía actual.
Flujo normal: 1. Usuario selecciona la opción de definir Nivel. 2. Sistema despliega los campos para definir el nivel: - Nombre (requerido) - Foreign key (requerido) - Nivel padre (requerido) - Otros atributos (opcional) 3. Usuario completa los datos y selecciona la opción de guardar. 4. Fin del caso de uso.
Flujos alternativos: 3A. Usuario selecciona un nivel padre que ya tiene un hijo asociado: 3A.1. El sistema le dice al usuario que el nivel padre tiene un hijo y no puede tener más de uno. 3A.2. Vuelve al paso 2.
Poscondiciones: Se guarda el nivel en la definición del DW

Tabla A.9: Definir Nivel

14 - Guardar consultas

Identificador: 14
Nombre: Guardar consultas
Objetivo: Guardar las consultas generadas
Actores: Usuario
Precondiciones: Se generó una consulta en pyDatalog.
Descripción: Se guardan los parámetros de la consulta generada por el usuario y su correspondiente código en pyDatalog.
Flujo normal: 1. Usuario selecciona la opción de guardar la consulta ejecutada. 2. Sistema guarda los parámetros de la consulta y su código en pyDatalog. 3. Fin del caso de uso.
Poscondiciones: Se guardó una consulta en el sistema para el DW.

Tabla A.10: Guardar consultas

Anexo B

Código pyDatalog

B.1. Código pyDatalog generado para una consulta

```
import sqlalchemy
from sqlalchemy import *
from sqlalchemy.ext.declarative import *
from sqlalchemy.orm import *
from pyDatalog import pyDatalog

######## Engine ########
engine = create_engine('postgresql+psycopg2://postgres:
    postgres@localhost:5432/datalog_test', echo=False)

######## Declarative base ########
Base =  declarative_base(cls=pyDatalog.Mixin, metaclass=pyDatalog.
    sqlMetaMixin)

######## Session #########
Session = sessionmaker(bind=engine)
session = Session()
Base.session = session
Base.metadata.bind = engine
```

Listing B.1: Conexión a la BD y sesión

```
######## Clases de dimensiones ########

class Tabtime(Base):
    __table__ = Table('time', Base.metadata, autoload=True,
    autoload_with=engine)
    def __repr__(self):
        return "%s" % self.date_id

class Tabprice(Base):
    __table__ = Table('price', Base.metadata, autoload=True,
    autoload_with=engine)
    def __repr__(self):
        return "%s" % self.price_subrange_id

class Tabshops(Base):
    __table__ = Table('shops', Base.metadata, autoload=True,
    autoload_with=engine)
    def __repr__(self):
        return "%s" % self.shop_id

```

```
class Tabgeographic_location(Base):
    __table__ = Table('geographic_location', Base.metadata, autoload=
   True, autoload_with=engine)
    def __repr__(self):
        return "%s" % self.city_id

class Tabauthors(Base):
    __table__ = Table('authors', Base.metadata, autoload=True,
   autoload_with=engine)
    def __repr__(self):
        return "%s" % self.author_id

class Tabbooks(Base):
    __table__ = Table('books', Base.metadata, autoload=True,
   autoload_with=engine)
    def __repr__(self):
        return "%s" % self.book_id

######## Clases de hechos ########

class Tabsales(Base):
    __table__ = Table('sales', Base.metadata, autoload=True,
   autoload_with=engine)

    def __repr__(self): # specifies how to display the employee
        return "Total: %s %s %s" % self.date_id, self.price_subrange_id,
    self.shop_id, self.city_id, self.author_id, self.book_id

######## Creacion de esquema ########

Base.metadata.create_all(engine)
```

Listing B.2: Carga de dimensiones y hechos

```
######## Crear predicados de jerarquias #########

pyDatalog.create_terms('XlowerLevel, XupperLevel, Z, hierDatedayDateDay,
    hierTrimestreyearDateMonth,hierTrimesteryearMonthTrimester,
   hierTrimesteryearTrimesterSemester,hierTrimesteryearSemesterYear,
   hierBimesteryearDateMonth,hierBimesteryearMonthBimester,
   hierBimesteryearBimesterSemester,hierBimesteryearSemesterYear,
   hierRangesSubrangeRange, hierShopShopTypeshop,
   hierLocationCityCountry,hierLocationCountryContinent,
   hierAuthorageAuthorAgerange, hierAuthorgenreAuthorGenre,
   hierBooklanguageBookLanguage, hierBookgenreBookSubgenre,
   hierBookgenreSubgenreGenre')

hierDatedayDateDay(XlowerLevel,XupperLevel) <=
    (Tabtime.date_id[Z] == XlowerLevel) &
    (Tabtime.date_weekday[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierTrimesteryearDateMonth(XlowerLevel,XupperLevel) <=
    (Tabtime.date_id[Z] == XlowerLevel) &
```

```
    (Tabtime.month_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierTrimesteryearMonthTrimester(XlowerLevel,XupperLevel) <=
    (Tabtime.month_id[Z] == XlowerLevel) &
    (Tabtime.trimester_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierTrimesteryearTrimesterSemester(XlowerLevel,XupperLevel) <=
    (Tabtime.trimester_id[Z] == XlowerLevel) &
    (Tabtime.semester_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierTrimesteryearSemesterYear(XlowerLevel,XupperLevel) <=
    (Tabtime.semester_id[Z] == XlowerLevel) &
    (Tabtime.year[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierBimesteryearDateMonth(XlowerLevel,XupperLevel) <=
    (Tabtime.date_id[Z] == XlowerLevel) &
    (Tabtime.month_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierBimesteryearMonthBimester(XlowerLevel,XupperLevel) <=
    (Tabtime.month_id[Z] == XlowerLevel) &
    (Tabtime.bimester_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierBimesteryearBimesterSemester(XlowerLevel,XupperLevel) <=
    (Tabtime.bimester_id[Z] == XlowerLevel) &
    (Tabtime.semester_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierBimestreyearSemesterYear(XlowerLevel,XupperLevel) <=
    (Tabtime.semester_id[Z] == XlowerLevel) &
    (Tabtime.year[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierRangesSubrangeRange(XlowerLevel,XupperLevel) <=
    (Tabprice.price_subrange_id[Z] == XlowerLevel) &
    (Tabprice.price_range_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierShopShopTypeshop(XlowerLevel,XupperLevel) <=
    (Tabshops.shop_id[Z] == XlowerLevel) &
    (Tabshops.shop_type[Z] == XupperLevel) &
```

```
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierLocationCityCountry(XlowerLevel,XupperLevel) <=
    (Tabgeographic_location.city_id[Z] == XlowerLevel) &
    (Tabgeographic_location.country_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierLocationCountryContinent(XlowerLevel,XupperLevel) <=
    (Tabgeographic_location.country_id[Z] == XlowerLevel) &
    (Tabgeographic_location.continent_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierAuthorageAuthorAgerange(XlowerLevel,XupperLevel) <=
    (Tabauthors.author_id[Z] == XlowerLevel) &
    (Tabauthors.author_agerange_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierAuthorgenreAuthorGenre(XlowerLevel,XupperLevel) <=
    (Tabauthors.author_id[Z] == XlowerLevel) &
    (Tabauthors.author_gender[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierBooklanguageBookLanguage(XlowerLevel,XupperLevel) <=
    (Tabbooks.book_id[Z] == XlowerLevel) &
    (Tabbooks.book_language_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierBookgenreBookSubgenre(XlowerLevel,XupperLevel) <=
    (Tabbooks.book_id[Z] == XlowerLevel) &
    (Tabbooks.book_subgenre_id[Z] == XupperLevel) &
    (XlowerLevel != -1) &
    (XupperLevel != -1)

hierBookgenreSubgenreGenre(XlowerLevel,XupperLevel) <=
    (Tabbooks.book_subgenre_id[Z] == XlowerLevel) &
    (Tabbooks.book_genre_id[Z] == XupperLevel) &
    (XlowerLevel != -1) & (XupperLevel != -1)
```

Listing B.3: Predicados de jerarquía

```
pyDatalog.create_terms('Xcant, Xsales, Xtime, Xprice, Xshops,
   Xgeographic_location, Xauthors, Xbooks, cubeView0, Timedate_id,
   Timedate_value, Priceprice_subrange_id, Priceprice_subrange_value,
   Shopsshop_id, Shopsshop_name, Geographic_locationcity_id,
   Geographic_locationcity_name, Authorsauthor_id, Authorsauthor_name,
   Booksbook_id, Booksbook_name')

######## Dia - Mes ########

```

```
pyDatalog.create_terms('cubeView1, fastCubeView1, Timemonth_id,
    Timemonth_value')

(cubeView1[Timemonth_id, Timemonth_value, Priceprice_subrange_id,
    Priceprice_subrange_value, Shopsshop_id, Shopsshop_name,
    Geographic_locationcity_id, Geographic_locationcity_name,
    Authorsauthor_id, Authorsauthor_name, Booksbook_id, Booksbook_name]
    == sum_(Xcant, for_each=Timedate_id)) <= (
     (fastCubeView0[Timedate_id, Timedate_value, Priceprice_subrange_id,
    Priceprice_subrange_value, Shopsshop_id, Shopsshop_name,
    Geographic_locationcity_id, Geographic_locationcity_name,
    Authorsauthor_id, Authorsauthor_name, Booksbook_id, Booksbook_name]
    == Xcant) &
     (Tabtime.date_id[Xtime] == Timedate_id) &
     (Tabtime.month_id[Xtime] == Timemonth_id) &
     (Tabtime.month_value[Xtime] == Timemonth_value) &
     (hierTrimestreYearDateMonth(Timedate_id,Timemonth_id))
)

cache1 = (cubeView1[Timemonth_id, Timemonth_value,
    Priceprice_subrange_id, Priceprice_subrange_value, Shopsshop_id,
    Shopsshop_name, Geographic_locationcity_id,
    Geographic_locationcity_name, Authorsauthor_id, Authorsauthor_name,
    Booksbook_id, Booksbook_name] == Xcant).data
(fastCubeView1[Timemonth_id, Timemonth_value, Priceprice_subrange_id,
    Priceprice_subrange_value, Shopsshop_id, Shopsshop_name,
    Geographic_locationcity_id, Geographic_locationcity_name,
    Authorsauthor_id, Authorsauthor_name, Booksbook_id, Booksbook_name]
    == Xcant) <= ((Timemonth_id == 1) & (Timemonth_value == 1) & (
    Priceprice_subrange_id == 1) & (Priceprice_subrange_value == 1) & (
    Shopsshop_id == 1) & (Shopsshop_name == 1) & (
    Geographic_locationcity_id == 1) & (Geographic_locationcity_name ==
    1) & (Authorsauthor_id == 1) & (Authorsauthor_name == 1) & (
    Booksbook_id == 1) & (Booksbook_name == 1) & (Xcant == 1) & (Xcant !=
     Xcant))
for x in cache1:
  fastCubeView1[x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8], x
    [9], x[10], x[11]] = x[12]

print((cubeView1[Timemonth_id, Timemonth_value, Priceprice_subrange_id,
    Priceprice_subrange_value, Shopsshop_id, Shopsshop_name,
    Geographic_locationcity_id, Geographic_locationcity_name,
    Authorsauthor_id, Authorsauthor_name, Booksbook_id, Booksbook_name]==
    Xcant))

######## Mes - Trimestre ########

pyDatalog.create_terms('cubeView2, fastCubeView2, Timetrimester_id,
    Timetrimester_value')

(cubeView2[Timetrimester_id, Timetrimester_value, Priceprice_subrange_id
    , Priceprice_subrange_value, Shopsshop_id, Shopsshop_name,
    Geographic_locationcity_id, Geographic_locationcity_name,
    Authorsauthor_id, Authorsauthor_name, Booksbook_id, Booksbook_name]
    == sum_(Xcant, for_each=Timemonth_id)) <= (
```

```
27      (cubeView1[Timemonth_id, Timemonth_value, Priceprice_subrange_id,
        Priceprice_subrange_value, Shopsshop_id, Shopsshop_name,
        Geographic_locationcity_id, Geographic_locationcity_name,
        Authorsauthor_id, Authorsauthor_name, Booksbook_id, Booksbook_name]
        == Xcant) &
28      (Tabtime.month_id[Xtime] == Timemonth_id) &
29      (Tabtime.trimester_id[Xtime] == Timetrimester_id) &
30      (Tabtime.trimester_value[Xtime] == Timetrimester_value) &
31      (hierTrimestreYearMonthTrimestre(Timemonth_id,Timetrimester_id))
32  )
33
34  print((cubeView2[Timetrimester_id, Timetrimester_value,
        Priceprice_subrange_id, Priceprice_subrange_value, Shopsshop_id,
        Shopsshop_name, Geographic_locationcity_id,
        Geographic_locationcity_name, Authorsauthor_id, Authorsauthor_name,
        Booksbook_id, Booksbook_name] == Xcant))
```

Listing B.4: Consultas roll-up

B.2. Código pyDatalog del predicado para realizar consultas a memoria local

Las consultas a memoria se realizaron a través de un nuevo predicado que utiliza como base el arreglo devuelto por la propiedad `data` de consultar sobre el predicado original de la vista de cubo. Si llamamos `cubeViewData` al arreglo con el resultado de la consulta, la definición del nuevo predicado con una sola dimensión y sin atributos distintos del identificador, es la del código B.5.

```
1  (fastCubeView[lvl_id] == Xcant) <= (
2      (lvl_id == 1) &
3      (Xcant == 1) &
4      (Xcant != Xcant))
5  for x in cubeViewData:
6      fastCubeView[x[0]] = x[1]
```

Listing B.5: Predicado para consultar a memoria local en lugar de a base de datos

La primer sentencia no agrega hechos, pero define el predicado `fastCubeView` vacío si el arreglo `cubeViewData` no tiene elementos, ya que si no se hace esto, `fastCubeView` queda indefinido y da un error a la hora de consultar.

Anexo C

Configuración y manual de usuario

C.1. Instalación y configuración

Para el correcto funcionamiento de la herramienta es necesario tener acceso a internet. Pudiendo así realizar funciones básicas como el inicio de sesión, conexión a bases de datos externas, y recuperar las definiciones y consultas guardadas.

Se deben tener instalados los siguientes softwares: npm: (última versión), node (versión 12 o mayor), python (versión 3 o mayor)

C.1.1. Configuración de la base de datos

La aplicación utiliza una base de datos Firebase, en la cual se guardan los usuarios, junto con sus conexiones, definiciones de DW y consultas realizadas.

En caso de querer cambiar las credenciales asociadas a dicha base de datos, es necesario acceder a https://console.firebase.google.com/ y crear un nuevo proyecto. Una vez creado este proyecto, se debe crear una aplicación web la cual representará al *framework*.

Figura C.1: Firebase crear aplicación.

Luego de registrar la aplicación nos encontramos en el paso “Agregar el SDK de Firebase”, el sistema desplegará un conjunto de configuraciones, estas deben ser guardadas ya que serán necesarias para finalizar la configuración de las nuevas credenciales.

```
1  // Import the functions you need from the SDKs you need
2  import { initializeApp } from "firebase/app";
3  // TODO: Add SDKs for Firebase products that you want to use
4  // https://firebase.google.com/docs/web/setup#available-libraries
5
6  // Your web app's Firebase configuration
7  const firebaseConfig = {
8    apiKey: "AIzfdsaSyA-ejemploApiKey_26hc",
9    authDomain: "ejemplo-dw.firebaseapp.com",
10   databaseURL: "https://ejemplo-dw.firebaseio.com",
11   projectId: "ejemplo-dw",
12   storageBucket: "ejemplo-dw.appspot.com",
13   messagingSenderId: "5111ej980564",
14   appId: "1:54ejemplo980564:web:163370901"
15 };
16
17 // Initialize Firebase
18 const app = initializeApp(firebaseConfig);
```

Listing C.1: Ejemplo de SDK de Firebase

Cambios en código

Para completar con el cambio de credenciales es necesario modificar el código fuente de la aplicación. Se debe sustituir el contenido de la configuración actual de la aplicación por la nueva obtenida en el punto anterior.

La ruta del file donde debe sustituirse la configuración es src/utils/firebase.js y la porción de código a sustituir está comprendido entre las líneas 4 a 10 incluidas.

C.1.2. Instalación de la aplicación

Para instalar la aplicación generando el ejecutable es necesario correr los siguientes comandos en consola, desde la raíz donde fue se descargó el código fuente, para crear el ejecutable de la aplicación con las nuevas credenciales:

- nvm use
- python install-*sistema-opertivo*.py

Donde la variable sistema-operativo puede ser linux, win o mac.

Luego de ejecutar el último comando se creará una carpeta en el mismo nivel que se encuentra la carpeta que contiene el código fuente de la aplicación, de nombre **frame-**

work_proyecto_2022. Dentro de está carpeta se generarán el ejecutable junto con los archivos necesarios para su correcto funcionamiento.

C.2. Manual de usuario

C.2.1. Configuración de usuario y conexión

Pantalla de inicio

Al ingresar a la aplicación se accede a la pantalla de Inicio (o home), donde se tienen las opciones de inicio de sesión o crear nueva sesión.

Figura C.2: Pantalla de inicio.

Crear nueva sesión

Para acceder a la pantalla de Crear nueva sesión existen dos formas:

- Desde la pantalla de inicio, seleccionando el botón de "Crear nueva sesión"
- Desde la pantalla de Iniciar sesión, accionando el botón de "Regístrate"

Una vez en la pantalla el usuario debe completar los campos asociados:

- Nombre: del usuario que desea registrarse
- Correo electrónico: dato utilizado para el inicio de la sesión
- Contraseña: dato utilizado para el inicio de la sesión

Figura C.3: Pantalla de Crear nueva sesión.

Para finalizar, se debe accionar el botón Continuar.

Iniciar sesión

Existen dos formas de acceder a la pantalla de Inicio de sesión:

- Desde la pantalla de Inicio, accionando el botón de "Iniciar sesión"
- Desde la pantalla de Crear nueva sesión, seleccionando el link de "Iniciar sesión"

Al seleccionar cualquiera de las opciones de Inicio de sesión, el usuario es redirigido a la pantalla de "Inicio de sesión".

Figura C.4: Pantalla de Inicio de sesión.

Dentro de la pantalla, el usuario debe completar los campos de texto desplegados ingresando un correo electrónico y una contraseña válidas.

Una vez completados los campos, se debe hacer clic al botón de Continuar para ingresar al sistema con el usuario identificado para los datos ingresados.

Seleccionar conexión

Una vez iniciada la sesión, se accede a la pantalla para seleccionar a que base de datos conectarse. Desde esta pantalla se puede seleccionar una conexión existente para ese usuario, editar una conexión o crear una nueva conexión.

Para seleccionar una conexión existente se debe abrir el dropdown con el texto "Tipo de base", el cual despliega todas las conexiones del usuario, y posteriormente se debe hacer click en la conexión deseada.

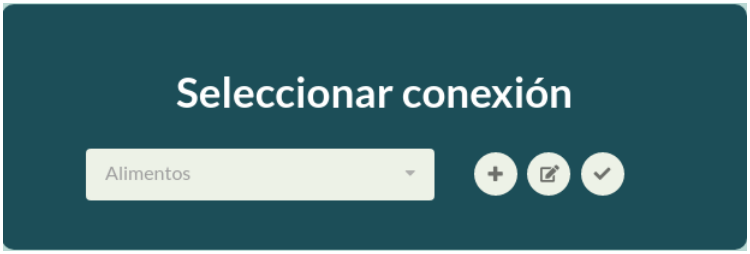


Figura C.5: Pantalla de Selección de conexión.

Una vez elegida la conexión se puede acceder a la misma presionando el botón de confirmación (tercer botón de izquierda a derecha).

Crear nueva conexión

Para crear una nueva conexión es necesario acceder a esta pantalla. Presionando el primer botón con el símbolo + (primero de derecha a izquierda).

Una vez accionado el botón de creación de conexión la aplicación nos redirigirá a la pantalla de Creación/Edición de conexión. Esta pantalla está compuesta por un formulario que debe ser completado para configurar dicha conexión.

Los campos a completar son los siguientes:

- Nombre de la conexión: nombre que identificará a la conexión
- Tipo de base: tipo de manejador para acceder a la base configurada
- Nombre de la base: nombre real por el que se define la base de datos
- Hostname: de la conexión a la base de datos
- Puerto: número de puerto de la conexión a la base de datos
- Usuario: usuario necesario para acceder a la base de datos configurada
- Contraseña: correspondiente al acceso del usuario ingresado a la base de datos

Figura C.6: Pantalla de Creación/Edición de conexión.

Una vez completado el formulario se puede realizar una prueba de la conexión presionando el botón "PROBAR". Dependiendo del resultado de la prueba el botón mostrará un icono que indicará dicho resultado. Para completar la acción, se debe presionar el botón "CONECTAR" el cuál guardará la conexión correspondiente a los datos del formulario y redirigirá al usuario a la pantalla de Definición de DW.

Editar conexión

Para modificar los datos de una conexión, primero debe seleccionarse desde el listado de conexiones y presionar el botón de edición mostrado en la siguiente imagen:

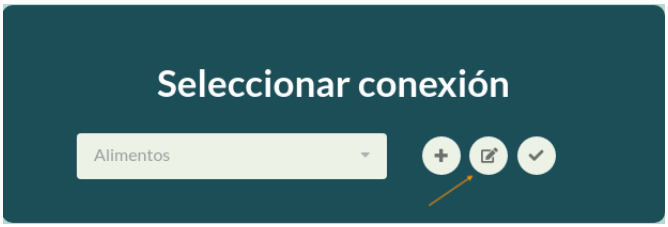


Figura C.7: Pantalla de Selección de conexión (botón editar).

Luego de seleccionar la conexión y presionar el botón, el sistema nos redirige a la pantalla de Creación/Edición de conexión, con los datos de la misma precargados en el formulario. Para guardar los cambios se debe presionar el botón "PROBAR".

C.2.2. Definición del Data Warehouse

Luego de establecer conexión con la base de datos del usuario, se redirige a la pantalla de definición del DW. Para su correcta definición, es necesario crear de forma completa al menos un esquema, cubo, dimensión, jerarquía y un nivel. Para chequear la completitud de la definición actual, el sistema realiza un conjunto de validaciones sobre ésta al intentar pasar a la pantalla de Consultas, las cuales despliegan un mensaje descriptivo al usuario con los cambios necesarios para cumplir con dicha validación.

Definir esquema

La definición de un esquema del DW consta de unicamente dos campos de texto libre: nombre del esquema (identificador) y una descripción.

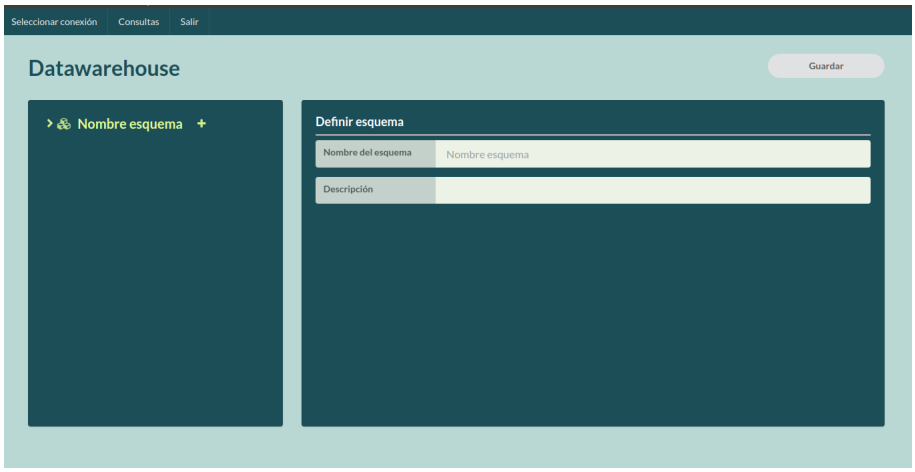


Figura C.8: Definición de esquema.

Agregar cubo o dimensión compartida

Para agregar un Cubo o una Dimensión compartida en la definición del esquema se debe presionar en el menú lateral el botón "+" a la derecha del esquema y seleccionar una de las opciones del menú desplegado.

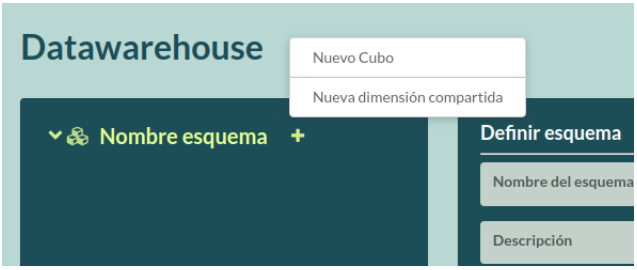


Figura C.9: Agregar Cubo o Dimensión compartida.

Definir cubo

La definición cubo consta de los siguientes campos:

- Nombre del cubo base: nombre del cubo
- Descripción: descripción del cubo
- Tabla asociada: se listan las tablas de la base de datos de la conexión actual. Esta tabla debe corresponder a la tabla de hechos del DW
- Nombre medida: nombre con el que se visualizará el atributo asociado a la medida
- Atributo asociado: lista los nombres de las columnas de la tabla seleccionada

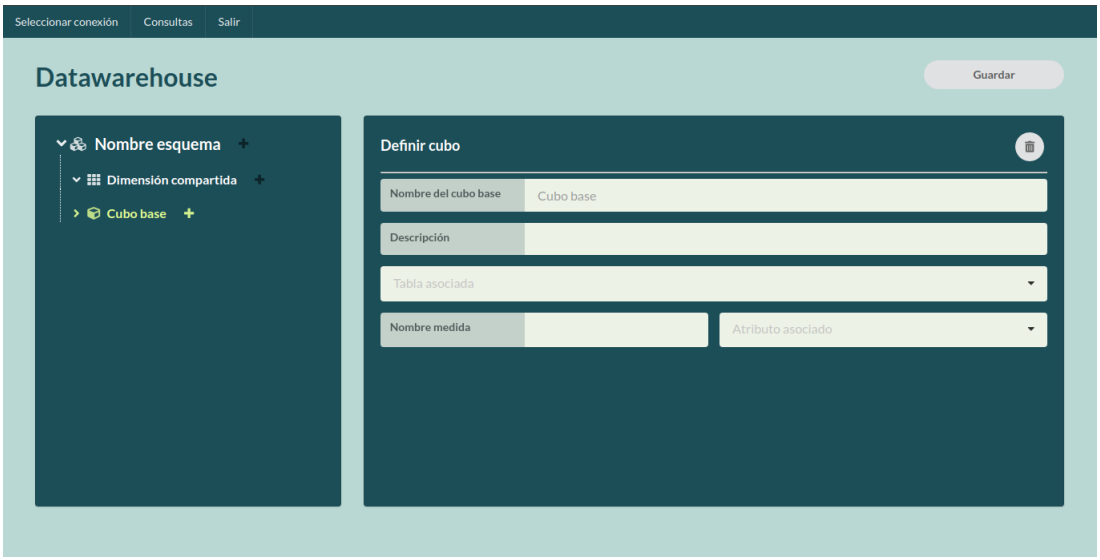


Figura C.10: Definición de cubo.

Agregar Dimensión privada o Referenciar dimensión compartida

Para agregar una dimensión privada o referenciar una dimensión compartida se debe presionar en el menú lateral el botón "+" a la derecha del cubo donde se quiere agregar.

Figura C.11: Agregar Dimensión privada o Referenciar dimensión compartida.

Definir dimensión

La definición de una dimensión consta de los siguientes campos:

- Nombre de la dimensión: nombre de la dimensión
- Descripción: descripción de la dimensión
- Tabla asociada: *dropdown* que lista las tablas existentes en la base de datos de la conexión actual. La tabla a seleccionar debe ser distinta de la tabla elegida en la definición del Cubo al que pertenece la dimensión

Dimensión privada

Es una dimensión que existe únicamente dentro del cubo al que pertenece.

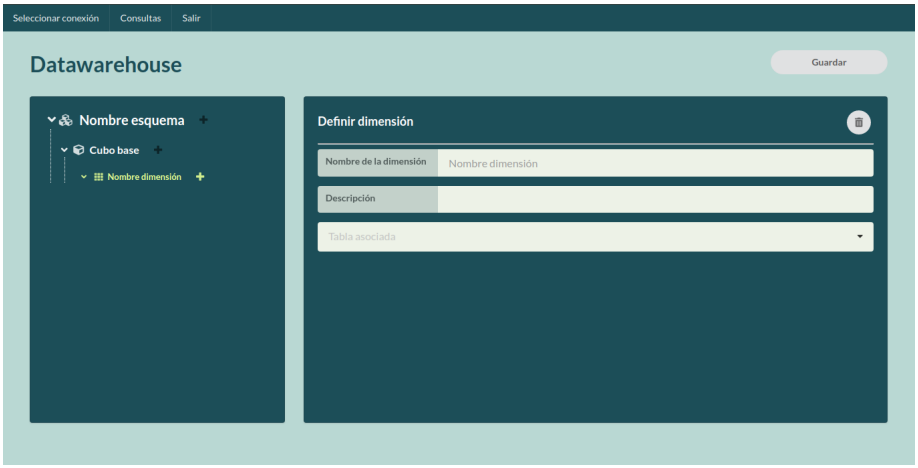


Figura C.12: Definición de dimensión.

Dimensión compartida

Es una dimensión que puede ser referenciada desde cualquier cubo.

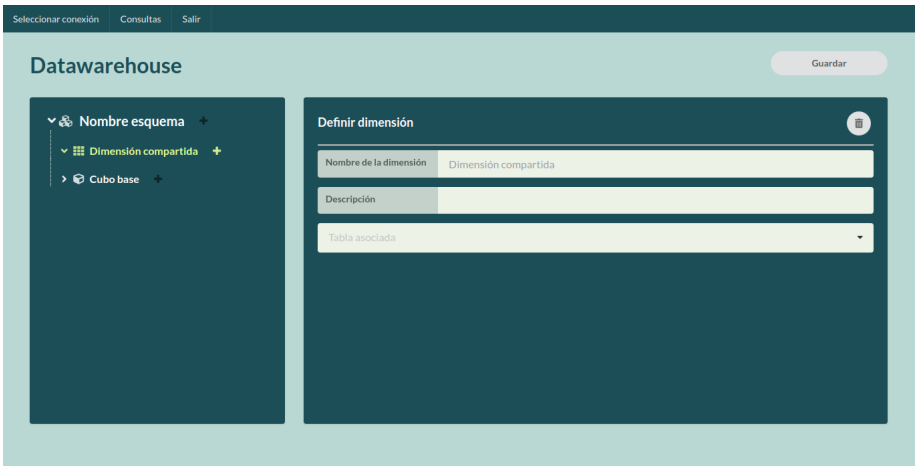


Figura C.13: Definición de dimensión compartida.

Referencia a dimensión compartida

Referencia a una dimensión compartida que fue creada fuera del cubo.

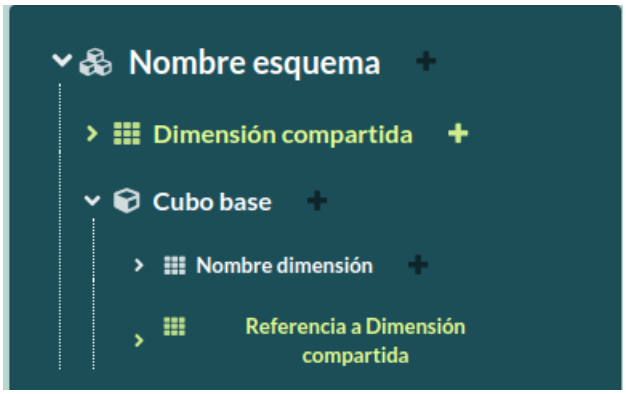


Figura C.14: Referencia a dimensión compartida.

Definir jerarquía

Para agregar una jerarquía se debe presionar en el menú lateral el botón "+" a la derecha de la dimensión donde se quiere agregar.

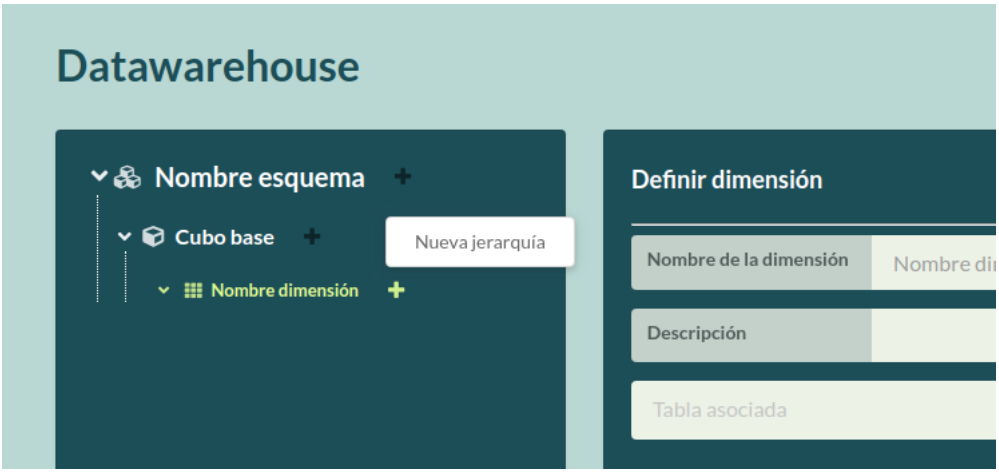


Figura C.15: Agregar Jerarquía.

La definición de una jerarquía consta de dos campos:

- Nombre de la jerarquía: Nombre de la jerarquía
- Descripción: Descripción de la jerarquía

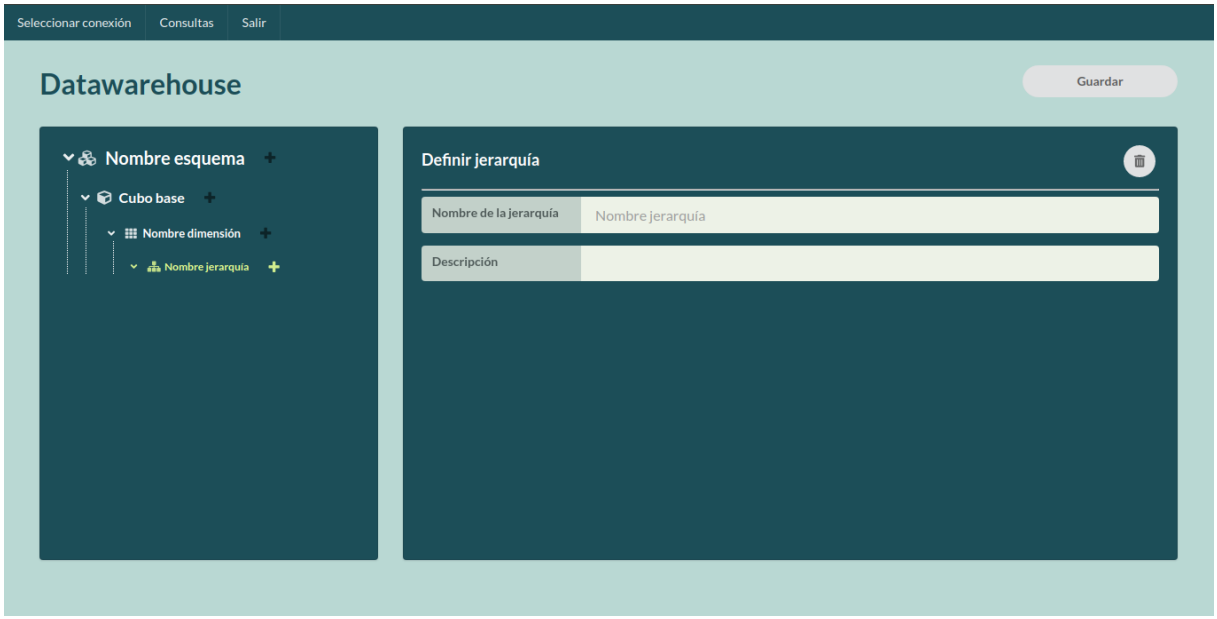


Figura C.16: Definición de jerarquía.

Definir nivel

Para agregar un nivel se debe presionar en el menú lateral el botón "+" a la derecha de la jerarquía donde se quiere agregar.

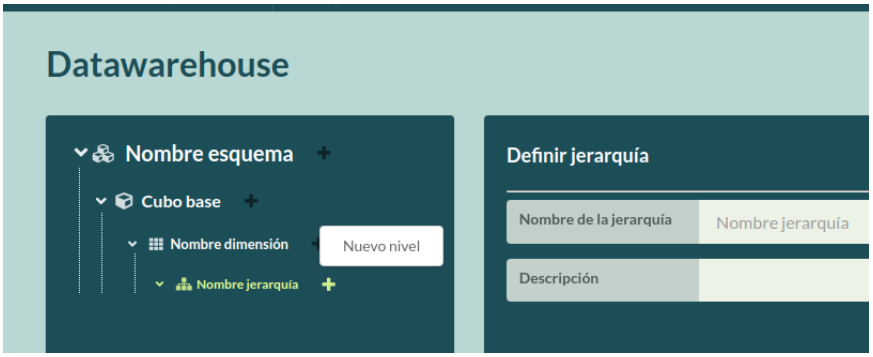


Figura C.17: Agregar nivel.

La definición de un nivel se conforma de los siguientes campos:

- Nombre del nivel: nombre nivel
- Nombre del atributo identificador: nombre del atributo identificador del nivel. En caso de ser el nivel más bajo en la jerarquía a la que pertenece, debe coincidir con el resto de los niveles de menor nivel de las otras jerarquías definidas en la misma dimensión y además, debe coincidir con un atributo en la tabla de hechos (aquella seleccionada en la definición del cubo al que pertenece la jerarquía).
- Atributo identificador: *dropdown* que lista todos los atributos existentes en la dimensión a la cual pertenece el nivel.

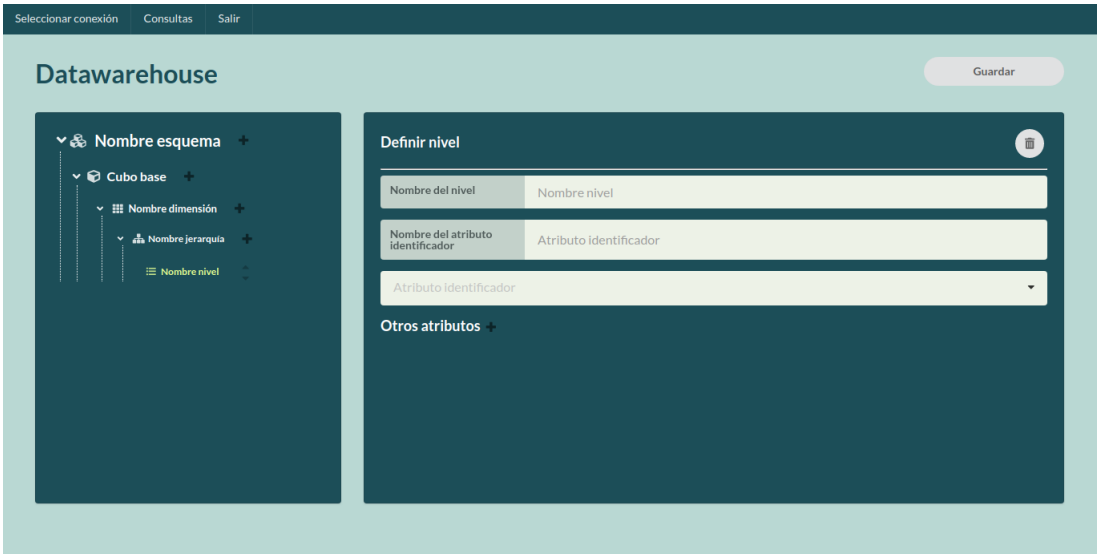


Figura C.18: Definición de nivel.

El nivel más bajo en la jerarquía es aquel que se encuentra más arriba del listado dentro de una misma jerarquía.

Ordenar nivel

Para subir o bajar un nivel en una jerarquía, se selecciona la flecha de subir o bajar según el caso, que se encuentran junto al nivel que se quiere mover, dentro del menú lateral.

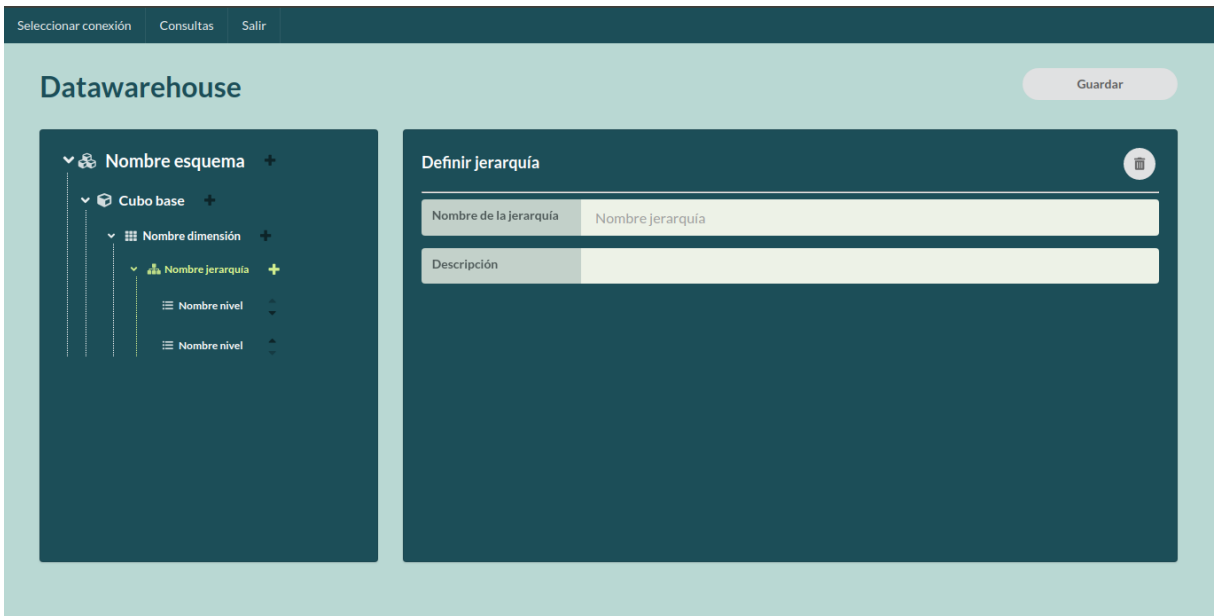


Figura C.19: Ordenar nivel.

C.2.3. Pantalla de consultas

Ejecución de operación roll-up

Como podemos ver en la siguiente figura, la pantalla de Consultas consta de dos secciones: el panel lateral y el árbol de consultas.

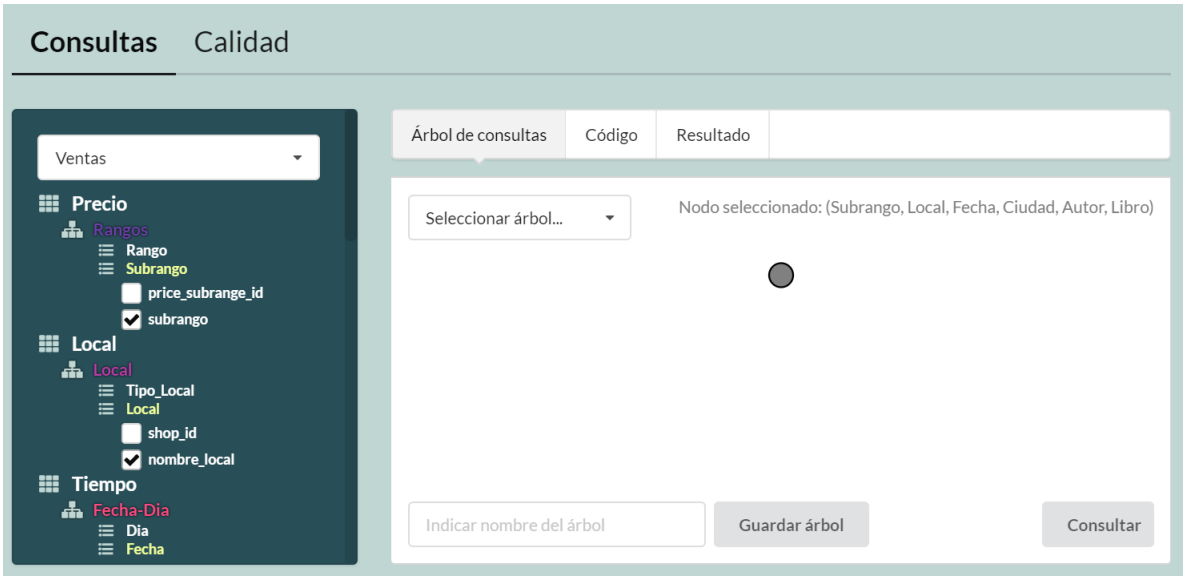


Figura C.20: Árbol de consultas inicial y panel lateral.

En el árbol de consultas, cada nodo representa una vista de cubo seleccionado en el panel lateral (en este caso Ventas) y se corresponde con una serie de operaciones de roll-up. Inicialmente el árbol consta de un solo nodo, el cual corresponde al cubo base formado por el nivel más bajo de cada dimensión del cubo.

En el panel lateral se muestra la definición del cubo (dimensiones, jerarquías y niveles). Los niveles de la vista del cubo asociado al nodo actual aparecen resaltados y se permite seleccionar qué campos se quieren mostrar en el resultado luego de ejecutar la consulta.

Desde el panel se permite seleccionar una jerarquía habilitada para realizar la operación roll-up según el nodo actual (vista de cubo). Al seleccionar una jerarquía, se genera un nuevo nodo que representa la vista de cubo resultante de ejecutar roll-up por dicha jerarquía y automáticamente el nodo actual pasará a ser el nodo resultante de la operación.

En la figura C.21 se muestra un árbol de consultas con varios nodos. En las aristas se indica la dimensión por la que se hace roll-up. Si posicionamos el puntero sobre la arista se muestra además la jerarquía por la que se hace el roll-up.

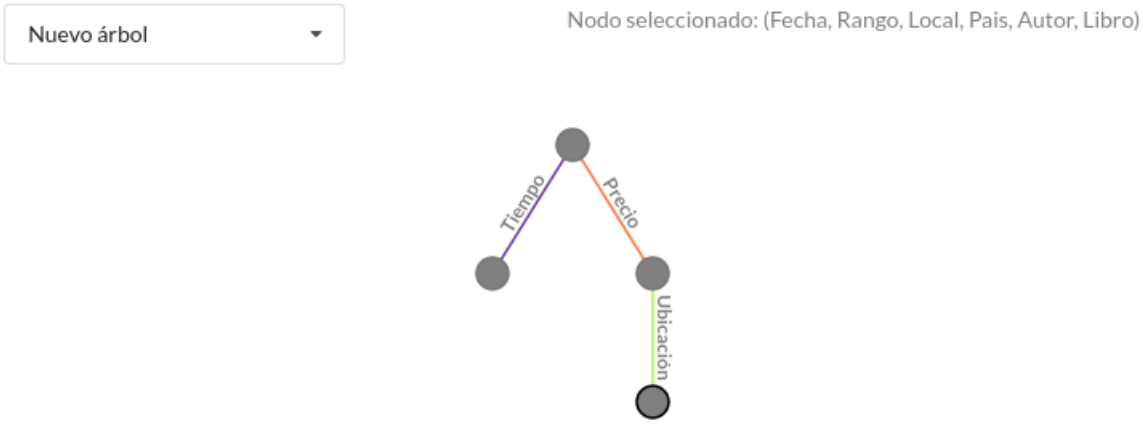


Figura C.21: Árbol de consultas con varios nodos

Dado un nodo seleccionado en el árbol de consultas, se puede ejecutar pulsando en el botón "Consultar". Luego de terminar la consulta en pyDatalog, el nodo se vuelve blanco y en las pestañas de resultado y código generado, mostradas en la figura C.22, se carga el resultado correspondiente y el código pyDatalog utilizado para ejecutar la consulta. Si nodos anteriores al nodo a ejecutar no están ejecutados se ejecutan en orden hasta el nodo que el usuario eligió ejecutar.

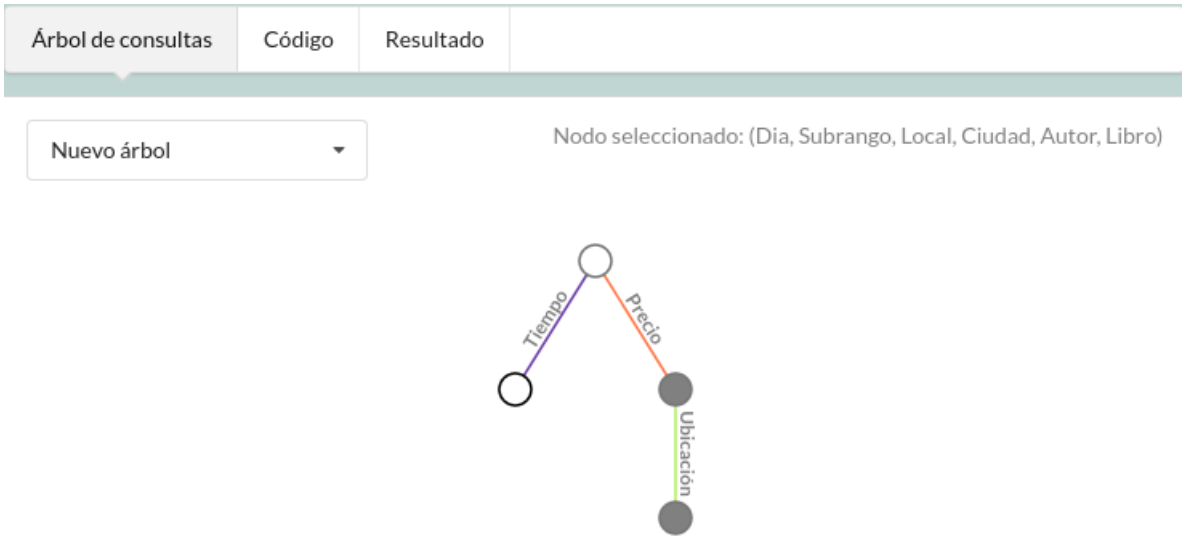


Figura C.22: Árbol luego de ejecutar una consulta.

C.2.4. Pantalla de calidad

En esta pantalla se pueden ejecutar las métricas de calidad implementadas para el DW definido, obteniendo el resultado y el código pyDatalog asociado.

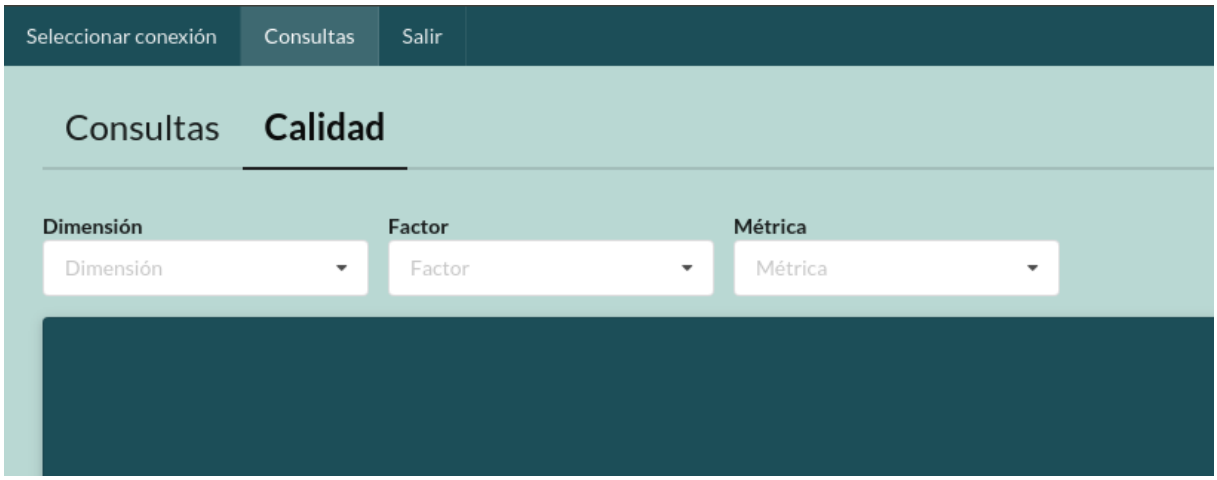


Figura C.23: Pantalla de consultas.

Se debe seleccionar primero la dimensión, luego se cargan factor correspondientes y por último se debe seleccionar la métrica que se quiere ejecutar. Luego de seleccionar la métrica, se desplegara el menú correspondiente para efectuar su ejecución.

Ejecución de métricas de calidad

Sumarizabilidad

Para el cálculo de está métrica se debe seleccionar un camino alternativo al nodo seleccionado en el árbol de consultas, es decir los niveles de las dimensiones deben coincidir.

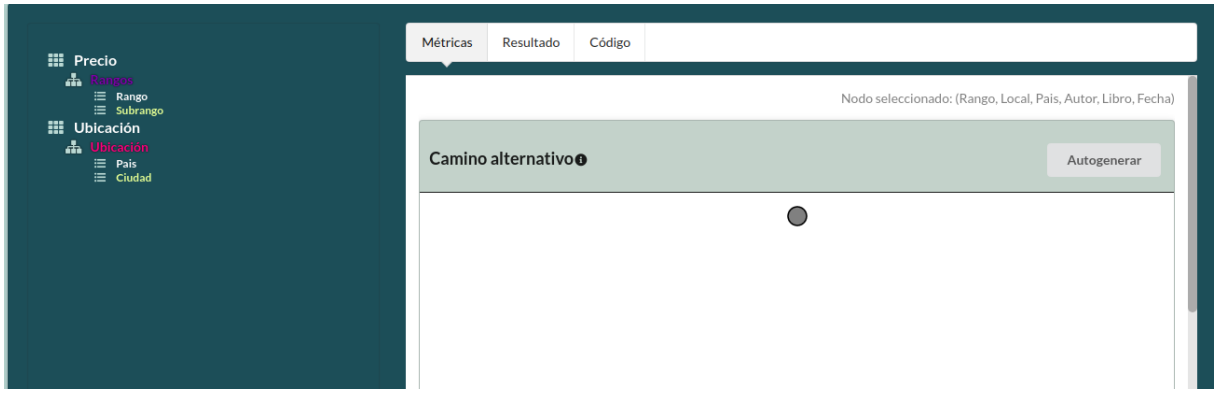


Figura C.24: Pantalla de consultas.

Con el botón autogenerar, el sistema puede construir automáticamente un camino

alternativo, en lugar de construirlo a mano por el usuario. Luego de seleccionado el camino alternativo se debe presionar el botón “ Calcular” para obtener los resultados.

Totalidad

Para el cálculo de Totalidad se debe elegir una dimensión, una jerarquía dentro de la dimensión seleccionada y luego un nivel perteneciente a esta última. Presionando el botón “ Calcular” se obtendrá el valor de la métrica sobre el nivel seleccionado.

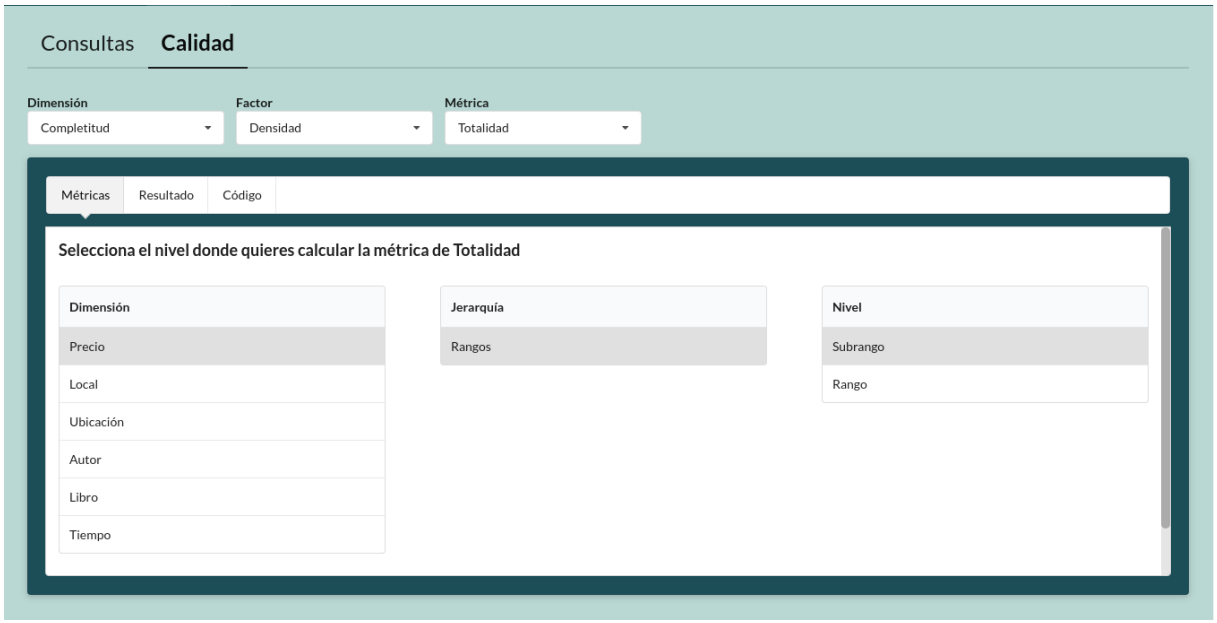


Figura C.25: Pantalla de consultas.

El cálculo de la métrica de Totalidad retorna como resultado una tabla donde se indican: la cantidad de tuplas Nulas, cantidad de tuplas no nulas y la cantidad de elementos por valor del nivel seleccionado.