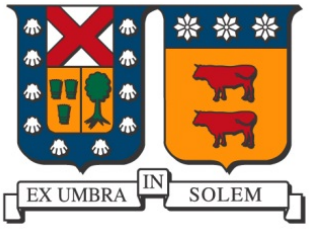
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
SANTIAGO - CHILE



“PROPUESTA DE METODOLOGÍA PARA EL DISEÑO LÓGICO DE BASES DE DATOS MULTIMODELO”

DANIELA ANDREA SÁNCHEZ NIZZA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: José Luis Martí Lara
Profesor Correferente: Ricardo Salas Letelier

agosto - 2025

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Propuesta de Metodología para el Diseño Lógicos de Bases de datos Multimodelos

Nombre del candidato(a): Daniela Andrea Sánchez Nizza

Carrera / Grado: Ingeniería Civil en Informática

Campus: Santiago San Joaquin **;** **Departamento**: Departamento de Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, José Luis Martí Lara, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 27/08/2025 ; **Firma:** _____

Estudiante o Candidato(a):

Fecha: 27/08/2025 ; **Firma:** _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

A mi mamá y papá, a mi memi y mi tata, gracias por ser mis mayores fans.

AGRADECIMIENTOS

Quiero expresar mi profundo agradecimiento a mis profesores de matemática, Ximena Flores, Olga Saiz, Michela Cuomo y Aldo Zambrano. Sin ustedes jamás habria entrado a una ingeniería ni pasado las mates a la primera.

También agradecer a mis amigos y en particular a mi pareja, por su apoyo incondicional, compañía y ayuda. Sin ustedes la carrera se habría vuelto muy fome y un suplicio.

A mi familia, quienes me han apoyado incondicionalmente siempre, en lo que sea que he querido estos años.

En penúltimo lugar, quiero agradecer a mi profesor guía José Luis Martí, gracias por tenerme paciencia incluso cuando no entendía que hacer o cuando estaba muy encasillada en una forma.

Finalmente, quiero agradecer al profesor que más me marcó en la universidad, Marcelo Mendoza, gracias a sus clases me empezaron a gustar las bases de datos. Que viaje! fui ayudante muchos semestres de bases de datos, también uno de ellos como ayudante de bases de datos avanzadas y al final todo culminó en una memoria sobre el tema.

RESUMEN

Resumen— Este trabajo de memoria se sitúa en el area del diseño lógico de las bases de datos multimodelo. Se propone una metodología que permite transformar un modelo conceptual de datos en un modelo lógico multimodelo mediante heurísticas de diseño. El objetivo es mantener la consistencia semántica del dominio original, evaluando cómo cada decisión de modelado afecta la navegabilidad y la representación de los datos. Los resultados muestran que la metodología propuesta permite adaptarse a diferentes tecnologías sin perder la integridad y consitencia de los datos del modelo inicial. La relevancia de este enfoque radica en que ofrece una base sólida para futuros trabajos en modelado y diseño de bases de datos multimodelo, facilitando la transición desde conceptos abstractos hacia estructuras de almacenamiento concretas y efectivas.

Palabras Clave— Modelado de datos, bases de datos multimodelo, bases de datos, diseño lógico, UML, ArangoDB, ETL, SurrealDB.

ABSTRACT

***Abstract*—** This thesis work is situated in the area of logical design for multimodel databases. It proposes a methodology that enables the transformation of a conceptual data model into a multimodel logical model through design heuristics. The objective is to preserve the semantic consistency of the original domain, evaluating how each modeling decision affects data navigability and representation. The results show that the proposed methodology can adapt to different technologies without compromising the integrity and consistency of the initial data model. The relevance of this approach lies in providing a solid foundation for future work in the modeling and design of multimodel databases, facilitating the transition from abstract concepts to concrete and effective storage structures.

***Keywords*—** Data modeling, multimodel databases, databases, logical design, UML, ArangoDB, ETL, SurrealDB.

GLOSARIO

ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*): Conjunto de propiedades que garantizan la fiabilidad de las transacciones en bases de datos.

AGE (*AgensGraph Extension*): Extensión para PostgreSQL que añade soporte para bases de datos de grafos.

AQL (*ArangoDB Query Language*): Lenguaje de consulta específico para ArangoDB.

BD (Base de Datos): Sistema organizado para almacenar, gestionar y consultar datos.

BI (*Business Intelligence*): Conjunto de herramientas para el análisis estratégico de datos empresariales.

CAP (*Consistency*, *Availability*, *Partition tolerance*): Teorema que describe los compromisos en sistemas distribuidos.

CRUD (*Create*, *Read*, *Update*, *Delete*): Operaciones básicas en la manipulación de datos.
DBMS (*Database Management System*): Sistema de gestión de bases de datos.

ERP (*Enterprise Resource Planning*): Sistema de planificación y gestión integral de recursos empresariales.

I/O (*Input/Output*): Operaciones de entrada y salida de datos en un sistema.

***NewSQL*:** Sistemas de bases de datos que combinan escalabilidad tipo NoSQL con propiedades ACID.

NoSQL (*Not Only SQL*): Categoría de bases de datos no relacionales, orientadas a datos semiestructurados y alta escalabilidad.

OLAP (*Online Analytical Processing*): Tecnología para análisis multidimensional de datos.

RDBMS (*Relational Database Management System*): Sistema de gestión de bases de datos relacional, basado en tablas.

UTFSM: Universidad Técnica Federico Santa María.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

En la actualidad, las organizaciones deben enfrentar el desafío de representar y gestionar datos complejos provenientes de múltiples fuentes y estructuras. Las bases de datos multimodelo surgen como una solución a este problema, al permitir la integración de distintos paradigmas de almacenamiento —como el relacional, documental, de grafos y clave-valor— en un mismo sistema. No obstante, la ausencia de metodologías consolidadas para su diseño lógico ha dificultado su uso correcto y su adopción efectiva.

El presente trabajo propone una metodología replicable que, a partir de un modelo conceptual UML, permite asignar de forma fundamentada un paradigma de almacenamiento a cada componente del dominio, preservando su consistencia semántica.

El documento se organiza en cinco capítulos principales. El primer capítulo corresponde a la contextualización del problema: introduce el concepto de bases de datos, su origen, evolución y detalla el problema a resolver, incluyendo sus causas, efectos y los objetivos del trabajo. El segundo capítulo presenta el marco conceptual, donde se aborda el surgimiento de las bases de datos multimodelo, sus tipos y el estado del arte respecto al diseño lógico. El tercer capítulo está dedicado al diseño de la solución, presentando la metodología propuesta para transformar un modelo conceptual UML en un modelo lógico multimodelo, junto con el plan de trabajo para su validación. El cuarto capítulo desarrolla la solución, aplicando la metodología sobre un caso de estudio real (el dataset IMDb), analizando cualitativamente el modelo resultante e implementándolo en dos gestores de bases de datos multimodelo.

Se espera que este trabajo sirva como guía para diseñadores e ingenieros de datos que enfrenten el desafío de transformar modelos abstractos en arquitecturas de almacenamiento heterogéneas, manteniendo la integridad y el significado original del dominio.

Capítulo 1: Contextualización del problema

A lo largo del tiempo, la forma en que se almacenan, organizan y consultan los datos ha experimentado una evolución constante, impulsada por los cambios en las necesidades tecnológicas y organizacionales. El capítulo establece el contexto técnico y conceptual de esta memoria, recorriendo el surgimiento de las bases de datos, su evolución y las limitaciones que presentan frente a escenarios modernos caracterizados por alta heterogeneidad y volumen. Se expone, además, por qué el diseño lógico en entornos multimodelo se ha convertido en un desafío vigente y relevante para el modelado de datos.

1.1. Las bases de datos tradicionales y la crecida exponencial de la información

Una base de datos es un repositorio digital para almacenar, gestionar y proteger colecciones organizadas de datos. Su objetivo es permitir el almacenamiento eficiente de información, así como su consulta y modificación por parte de distintos usuarios y sistemas [Berg *et al.*, 2012].

Este concepto comenzó a tomar forma en la década de los 60, cuando en respuesta a la masificación de los computadores surgió la necesidad de manejar los grandes volúmenes de datos de las empresas de forma estructurada y eficiente [Foote, 2021]. Es así como nacieron los primeros modelos de almacenamiento de datos: el modelo jerárquico y el modelo de red. Ambos se basaban en estructuras rígidas y acceso navegacional, es decir, para consultar los datos era necesario conocer la ruta exacta entre registros y la estructura interna del sistema. Pese a estas limitaciones, estos modelos marcaron el inicio de la gestión de datos computacional moderna [Berg *et al.*, 2012].

Las primeras bases de datos modernas surgieron en la década de 1970, marcando un cambio fundamental en la forma de acceder a la información: ya no era necesario seguir rutas predefinidas entre registros, sino que se podía consultar directamente el dato deseado. Estos sistemas emergieron en un contexto donde el volumen de información a manejar era considerablemente menor al actual. En ese escenario, las bases de datos relacionales (RDBMS) se consolidaron como el estándar dominante, organizando los datos en tablas con estructuras rígidas y esquemas predefinidos [Neupane, 2024]. Lo anterior era posible debido a que los datos en esa época eran en su mayoría muy estructurados, por lo que se conseguía su almacenamiento eficiente en un modelo tabular. Estas bases se caracterizan por cumplir con las propiedades ACID (*Atomicity*, *Consistency*, *Isolation*, *Durability*), que garantizan la fiabilidad de las transacciones, asegurando que todas las operaciones se ejecuten de forma completa, consistente y resistente a fallos, incluso ante errores del sistema. Sin embargo, al pasar los años, la cantidad de datos a gestionar por parte de los sistemas informáticos aumentó exponencialmente y su naturaleza se diversificó [Foote, 2021].

Surgió así el problema del manejo efectivo, eficaz y eficiente de grandes volúmenes de datos,

provenientes principalmente del tráfico web. Este problema se ha convertido en un desafío crítico para los sistemas informáticos, siendo impulsado por el aumento del uso de dispositivos conectados, aplicaciones móviles, redes sociales, sensores de *IoT* (Internet de las Cosas), entre otros [Phiri, 2022].

Es así como las RDBMS comenzaron a mostrar limitaciones significativas de uso y rendimiento. Los cuellos de botella, los problemas de escalabilidad, flexibilidad y la pérdida en general de eficiencia y recursos se hicieron evidentes, generando la necesidad de un cambio en el cómo se manipulan y almacenan los datos.

1.2. Diversidad de datos y el desafío de definir un esquema efectivo

El crecimiento exponencial de la información no solo implicó un aumento en el volumen de datos, sino también en su diversidad de tipos. Los datos actualmente pueden ser clasificados en estructurados, semiestructurados y no estructurados, cada uno con características distintas que requieren estrategias de almacenamiento y procesamiento específicas. Forzar los datos a ajustarse a un esquema relacional rígido puede resultar ineficiente y llevar a un mal uso de recursos, problemas de rendimiento o, incluso, a la pérdida de integridad de los datos[Foote, 2021].

Esta problemática impulsó el desarrollo de bases de datos NoSQL, que introdujeron esquemas más flexibles y estructuras optimizadas para diferentes tipos de datos [Kelly, 2022]. Estas bases de datos permitieron a los sistemas informáticos manejar grandes volúmenes de información de manera más eficiente y adaptativa, favoreciendo un escalamiento horizontal en lugar de uno vertical, que suele ser más costoso. Además, se introdujo el concepto de bases de datos distribuidas, donde los datos se almacenan en múltiples servidores geográficamente separados, aumentando la tolerancia a fallos y mejorando el rendimiento.

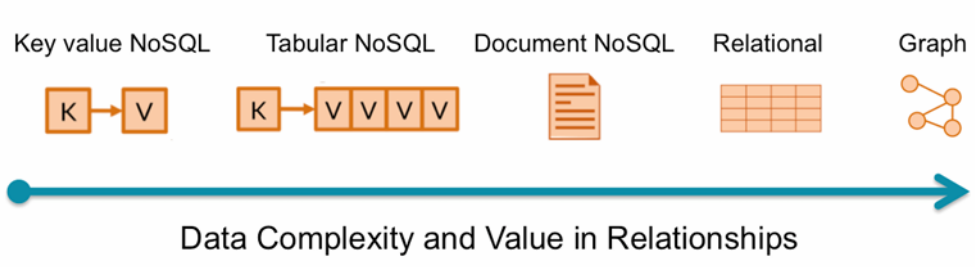


Figura 1: Complejidad de datos y relaciones

Fuente: Datastax, The Multi-Model Database Cloud Applications in a Complex World [Ammar, 2022]

1.3. Bases de datos NoSQL

Las bases de datos NoSQL surgen como una solución flexible y escalable que se utiliza en diferentes modelos de datos. De esta forma, el desarrollador puede elegir el DBMS que más se ajuste a sus necesidades para sacar el máximo provecho. Una característica común en muchos motores NoSQL es el enfoque *schemaless*, es decir, la capacidad de almacenar datos sin un esquema rígido definido en la base de datos[Meier y Kaufmann, 2019]. Esto permite manejar documentos, claves o nodos con estructuras variables, lo que otorga gran flexibilidad y facilita la adaptación a cambios en los requerimientos. Sin embargo, también transfiere al nivel de aplicación la responsabilidad de mantener la coherencia de los datos, ya que el motor no la garantiza de forma nativa. Las principales bases de datos NoSQL son:

1. **Clave-valor**: Las bases de datos de clave-valor, como Redis y Amazon DynamoDB, almacenan datos como pares formados por una clave única y un valor asociado. Este modelo es útil para aplicaciones donde la velocidad de lectura/escritura es crucial y no se requiere una estructura compleja de datos, como en cachés de memoria y sesiones de usuario. [haz, 2024]

2. **Columnar**: Las bases de datos columnares, como Cassandra y HBase, almacenan datos como columnas en lugar de filas, lo que permite un acceso rápido a grandes volúmenes de datos, dado que solo las columnas necesarias se leen del disco, lo que mejora el rendimiento de las consultas. Este modelo es adecuado para análisis de grandes datos y aplicaciones que requieren realizar consultas agregadas de alto rendimiento.[Jordan, 2023]

3. **Documental**: Las bases de datos documentales, como MongoDB y Couchbase, almacenan datos en documentos (JSON, BSON, o XML), lo que les permite manejar datos jerárquicos y semiestructurados. Este modelo se ajusta a aplicaciones que requieren almacenar y consultar datos que pueden variar en estructura o que contienen anidaciones complejas, donde la flexibilidad y agilidad son cruciales.[mon, 2021]

4. **De grafos**: Las bases de datos de grafos, como Neo4j y ArangoDB, están diseñadas para gestionar datos altamente conectados mediante la representación de las relaciones entre los datos como vértices y aristas. Este modelo se ajusta a aplicaciones que requieren relaciones complejas entre datos, como redes sociales o recomendaciones, donde las relaciones son tan importantes como los datos en sí para el funcionamiento del sistema.[neo, 2015]

Cada paradigma descrito permite ser utilizado en distintos casos de uso, cada uno trae ventajas y desventajas. En la tabla 1 se comparan los diversos paradigmas de almacenamiento:

Modelo de datos	Fortalezas principales	Limitaciones técnicas	Casos de uso ideales
Clave-valor	Lectura/escritura ultrarrápida para búsquedas por clave; Alto rendimiento en operaciones CRUD; Alta escalabilidad horizontal	No permite consultas complejas eficientes; estructura interna inexistente para modelar jerarquías o relaciones	Cachés, sesiones de usuario, configuración distribuida[haz, 2024]
Columnar	Eficiente para análisis masivo de datos; bajo I/O al leer columnas específicas; Eficiente para OLAP	No apto para operaciones OLTP, datos muy mutables o para modelar relaciones complejas (ineficiente) [Groves, 2025]	OLAP, BI , reporte analítico, datos con series de tiempo de estructura fija, data warehousing
Documental	Esquema flexible; fácil modelado de jerarquías y datos semiestructurados mediante uso de documentos XML, JSON y BSON, alta escalabilidad horizontal	Difícil modelar relaciones complejas; duplicación de datos puede afectar consistencia, uso muy limitado de JOIN, requiere definir cuidadosamente índices y difícil mantenimiento [Bathla *et al.*, 2018].	Aplicaciones web modernas con estructuras variables como ecommerce, APIs REST, sistemas de análisis en tiempo real, IoT [mon, 2021]
Grafo	Alto rendimiento en relaciones complejas y ampliamente conectadas; altamente adaptables a cambios en estructuras de nodos y aristas; las relaciones son entidades de primera clase (se les trata como un tipo de dato fundamental) lo cual permite que las operaciones con ellas sean altamente eficientes	Escalabilidad horizontal limitada en algunos motores; menos madurez en herramientas analíticas; no óptimo en datos muy poco conectados [neo, 2015]	Recomendaciones, redes sociales, grafos semánticos, manejo de accesos y validación de identidad[neo, 2015]
Relacional	ACID, integridad referencial, consultas complejas con SQL	Rígido en el esquema; escalabilidad horizontal limitada	Sistemas financieros, ERP, aplicaciones tradicionales

Tabla 1: Comparación entre modelos de datos utilizados en bases de datos NoSQL
Fuente: Elaboración propia basada en [haz, 2024], [Groves, 2025], [Bathla *et al.*, 2018], [mon, 2021], [neo, 2015]

1.4. El teorema CAP y las carencias de las bases de datos NoSQL

Se mencionó anteriormente que una característica de las bases de datos es la capacidad de ser distribuidas. Por ello, están sujetas al teorema CAP (también conocido como teorema de Brewer) que corresponde a un principio fundamental en el diseño de sistemas distribuidos. CAP son las siglas de tres propiedades que un sistema distribuido puede intentar garantizar, pero el teorema afirma que solo se pueden garantizar dos de ellas al mismo tiempo. Dichas propiedades son: Consistencia (C), Disponibilidad (A) y Tolerancia a Particiones (P). Es así como las bases de datos pueden variar entre sí en la combinación de las propiedades y los mecanismos usados para llevarlas a cabo.

Aunque las bases de datos NoSQL resolvieron muchos problemas de escalabilidad y flexibilidad, carecían de ciertas garantías transaccionales y de consistencia estricta que ofrecían las bases de datos relacionales. Para abordar estas limitaciones sin renunciar al rendimiento y la escalabilidad, surgieron posteriormente nuevas aproximaciones como NewSQL y los sistemas multimodelo, los cuales buscan combinar las ventajas de NoSQL con garantías de consistencia y soporte transaccional, abriendo así el camino hacia soluciones más flexibles y robustas.

1.5. Criterios de diseño lógico en bases de datos NoSQL

El diseño lógico de una base de datos se refiere a la forma en que se modelan los datos y sus relaciones de acuerdo con las necesidades de información del sistema, sin considerar aún aspectos físicos como la replicación, distribución o almacenamiento. En el caso de las bases de datos NoSQL, que pueden adoptar distintos modelos, el diseño lógico implica adaptar la estructura de los datos a las operaciones esperadas, con criterios diferentes a los del modelo relacional tradicional. A continuación, se presentan criterios relevantes para el diseño lógico en bases de datos NoSQL:

- **Diseño orientado a consultas (*Query-Driven Design*)**: Consiste en diseñar el esquema de datos de tal forma que favo- rezca el desempeño de consultas frecuentes u operaciones de alto valor para el contexto en el que se utilizarán, lo anterior puede provocar la duplicación y/o anidamiento de datos. Estas consultas y operaciones se pueden extraer de las historias de usuario que describen las necesidades del sistema, como también pueden extraerse del contexto en el que estará inserta la base de datos [Pore, 2019].
- **Identificación de unidades de agregación**: A diferencia del modelo relacional que modela entidades y relaciones de forma separada, en NoSQL es común agrupar datos relacionados dentro de una misma unidad (documento, registro, nodo, etc.). El diseño lógico debe determinar qué información se consulta y actualiza de forma conjunta, para definir agregados coherentes [Jadon, 2024].

- **Equilibrio entre normalización y desnormalización**: Aunque en NoSQL es frecuente usar desnormalización para mejorar el rendimiento de lectura, el modelo lógico debe justificar cuándo se duplica información y cómo se mantiene la consistencia de los datos. No se trata de eliminar completamente la normalización, sino de encontrar un equilibrio de acuerdo a las necesidades del sistema [Jordan, 2023] [Pore, 2019].

- **Cohesión semántica**: Las estructuras de datos deben tener significado claro y estar organizadas de modo que cada elemento cumpla una función definida dentro del modelo. Se deben evitar estructuras ambiguas o sobrecargadas que mezclen conceptos o niveles de abstracción para evitar complejizar la mantención y prever errores semánticos o de validación en el sistema [Sadalage y Fowler, 2013].

- **Claridad en las relaciones implícitas y explícitas**: Aunque NoSQL no siempre usa claves foráneas o relaciones formales, el diseño lógico debe especificar cómo se representan las asociaciones entre datos (por claves embebidas, referencias cruzadas, caminos de grafo, etc.) [Johnston, 2022].

- **Consistencia semántica**: Aunque el sistema pueda tolerar consistencia eventual a nivel físico, el diseño lógico debe mantener una organización clara y consistente de la información. Esto significa que la forma en que se representan los datos debe corresponder fielmente con los conceptos del dominio del sistema, sin ambigüedades ni contradicciones. Las entidades deben estar bien definidas, las relaciones entre ellas deben entenderse fácilmente, y las estructuras utilizadas deben mantenerse uniformes en todo el modelo [Padhy *et al.*, 2011].

- **Adaptabilidad al cambio**: El modelo debe prever posibles cambios futuros en el dominio (nuevas propiedades, tipos de relaciones, reglas de negocio) y estructurarse de forma que dichos cambios puedan incorporarse sin rediseñar completamente el modelo lógico [Scherzinger *et al.*, 2013].

A pesar de los avances en el desarrollo de tecnologías orientadas a mejorar la escalabilidad, flexibilidad y rendimiento de los sistemas de almacenamiento de datos por medio de las bases de datos NoSQL y *NewSQL*, en años recientes ha surgido una necesidad creciente: integrar múltiples modelos de datos en una misma plataforma sin recurrir a arquitecturas complejas y fragmentadas. Lo anterior, principalmente por las dificultades de mantenimiento y costos que significa. En este contexto emergen las bases de datos multimodelo, las cuales buscan unificar distintos modelos de almacenamiento dentro de un solo sistema, facilitando la gestión de datos heterogéneos y reduciendo la dependencia de arquitecturas políglotas [Goodfellow, 2025]; además, al estar todo en un único lugar, resulta más fácil manipular datos almacenados con distintos paradigmas de almacenamiento.

Sin embargo, a pesar de su promesa, las bases de datos multimodelo enfrentan desafíos significativos que aún no han sido abordados de manera sistemática. En particular, no existe una metodología consolidada que guíe la transformación desde un modelo conceptual hacia un modelo lógico multimodelo, asegurando que se mantenga la consistencia semántica del

dominio [Bimonte *et al.*, 2023]. Este vacío metodológico dificulta la validación del modelo de datos y la toma de decisiones informadas sobre qué paradigmas emplear en cada parte del sistema.

Además, dado que las bases multimodelo surgen alrededor del año 2013, aún existe un grado de desconocimiento en torno a sus capacidades reales y a las implicancias de su implementación.

1.6. El problema a resolver

En el diseño de bases de datos multimodelo no hay una metodología de diseño lógico universalmente aceptada, solo existen propuestas de criterios de diseño lógico, pero son dispersas, poco probadas e incompletas en algunos casos [Bimonte *et al.*, 2023]. No se ha llegado a un conjunto aplicable de criterios que por medio de una metodología, permitan guiar el proceso de selección y diseño en contextos reales. Es más, esta problemática de realizar un esquema que balancee requerimientos del sistema como velocidad, facilidad de consulta y su mantención en el tiempo se detalla en múltiples papers relacionados a las bases de datos multimodelo, papers con menos de 5 años de antigüedad.

A continuación, se detallan las causas y los efectos de este problema:

Causas

1. **Diversidad de Requisitos y Escenarios de Uso:** Los diferentes casos de uso requieren distintos modelos y estrategias para manipular datos, lo que complica la definición de un conjunto único de criterios para elegir una base de datos multimodelo. Los criterios existentes se contradicen entre sí dependiendo del contexto [Lu y Holubová, 2019].

2. **Aparición Algo Reciente de las Bases de Datos Multimodelo:** Como las bases de datos multimodelo son una tecnología relativamente nueva, hay poca investigación y casos de estudio que proporcionen guías prácticas o criterios basados en experiencias previas.

3. **Fragmentación en la Evolución de la Tecnología de Bases de Datos:** La evolución de las bases de datos se ha centrado en modelos de datos específicos. Esta fragmentación ha llevado a enfoques y arquitecturas difíciles de integrar, lo que dificulta el desarrollo de criterios unificados.

Efectos

1. **Adopción de bases de datos ineficientes:** La falta de criterios claros puede llevar a las organizaciones a elegir bases de datos multimodelo que no son adecuadas para sus

necesidades específicas. Esto puede resultar en un mal uso de los recursos y dificultad para escalar.

2. **Mayor complejidad en la gestión de datos:** La falta de una guía clara o criterios para validar un modelo puede llevar a diseños ineficientes que son difíciles de mantener y optimizar.

3. **Riesgo de tomar decisiones basadas en tendencias y no en necesidades reales:** La falta de criterios claros puede llevar a las organizaciones a tomar decisiones basadas en tendencias y no en un análisis detallado de sus necesidades, llevando a elegir tecnologías inadecuadas.

1.7. Objetivo del Trabajo

El objetivo general consiste en desarrollar y validar una metodología replicable para el diseño lógico de bases de datos multimodelo, basada en heurísticas de diseño, que permitan decidir cómo representar cada parte de los datos según su estructura y su uso.

Esta metodología guía la transformación desde un modelo conceptual en notación UML hacia un modelo lógico multimodelo, implementado en distintos motores. Utiliza criterios para representar cada parte del dominio según su estructura y uso, para asegurar que el diseño lógico resultante sea fiel al significado original del dominio y adaptable a distintos entornos tecnológicos.

Lo anterior, para proponer una herramienta sistemática que permita a diseñadores de bases de datos multimodelo transformar modelos conceptuales complejos en representaciones lógicas coherentes, sin perder información del dominio en la transformación, facilitando su implementación en entornos heterogéneos y su evolución futura.

Como objetivos específicos se consideran:

- Analizar las características técnicas de distintos gestores de bases de datos multimodelo, evaluando los modelos que soportan, su integración entre paradigmas y su adecuación a distintos escenarios, con el fin de comprender las posibilidades y limitaciones de cada uno para tomar decisiones de asignación informadas en el modelo lógico.

- Identificar y sistematizar un conjunto de al menos cinco heurísticas de diseño lógico, extraídas del estado del arte y del análisis propio, que orienten la transformación de modelos conceptuales a modelos lógicos en bases de datos multimodelo, para apoyar la toma de decisiones fundamentadas durante el diseño lógico, asegurando coherencia con el dominio y facilitando su aplicación en contextos similares.

- Diseñar un proceso metodológico de transformación desde un modelo conceptual hacia un modelo lógico multimodelo, utilizando las heurísticas identificadas como guía para asignar un paradigma de almacenamiento a cada componente del dominio, para

ofrecer un proceso paso a paso que permita aplicar las heurísticas de manera estructurada, manteniendo la consistencia del dominio.

- Implementar el modelo lógico obtenido en un caso de estudio real, empleando dos gestores de bases de datos multimodelo. Para validar que la metodología puede adaptarse a diferentes motores con sus respectivas particularidades, preservando la consistencia semántica del dominio.

1.8. Alcance de la solución

Este trabajo se enfoca exclusivamente en el diseño lógico de bases de datos multimodelo, partiendo desde un modelo conceptual expresado como diagrama de clases UML. La propuesta metodológica desarrollada considera únicamente aspectos lógicos del diseño, sin abordar decisiones físicas como replicación, particionamiento o configuración de infraestructura. La validación empírica se realiza mediante un único caso de estudio y su implementación se limita a dos gestores multimodelo.

Capítulo 2: Marco conceptual

En este capítulo se aborda el nacimiento de las bases de datos multimodelo, las arquitecturas existentes, el estado del arte del modelado lógico y las dimensiones semánticas que deben considerarse. Todo ello con el propósito de fundamentar la construcción de una metodología que permita transformar un modelo conceptual en un modelo lógico multimodelo coherente y semánticamente consistente.

2.1. El nacimiento de las bases de datos multimodelo

A medida que las organizaciones comenzaron a manejar diferentes tipos de datos en un mismo contexto, se hizo evidente que ninguna solución única podía abordar todas sus necesidades [Liu *et al.*, 2018]. En los primeros años de NoSQL, cada sistema estaba especializado en un modelo de datos particular, lo que obligaba a las empresas a utilizar múltiples gestores de bases de datos, incrementando la complejidad operativa y la carga de trabajo en integración de datos.[Mihai, 2020] Esta arquitectura políglota implicaba desafíos adicionales de compatibilidad, integración de datos y uso de interfaces, incrementando la carga de trabajo tanto del sistema como del usuario.

Para superar estos desafíos, se requieren sistemas que permitan soportar diversos modelos de datos en una única plataforma, que sean escalables y que permitan que los datos “conversen” entre sí sin importar su tipo[Liu *et al.*, 2018].

Esta integración permite no solo reducir la complejidad técnica, sino también disminuir los costos operativos asociados al mantenimiento, licenciamiento y capacitación de personal [Goodfellow, 2025], ofreciendo una alternativa más eficiente frente a las arquitecturas políglotas.

En este contexto, surgieron las bases de datos multimodelo como la siguiente fase en la evolución de los gestores de datos. Estas bases de datos permiten combinar diversos tipos de modelos en un único servidor integrado, organizando diferentes tipos de datos en estructuras que optimicen y faciliten su uso. Cabe mencionar que algunos motores multimodelo permiten realizar transacciones que abarcan múltiples modelos de datos de forma consistente, mientras que otros limitan las operaciones transaccionales a un único modelo por vez, lo que representa un factor clave al evaluar sus capacidades reales para casos de uso complejos [Lu, 2017].

La mayoría de las bases de datos multimodelo existentes nacen como una evolución de las bases de datos SQL y NoSQL [Mihai, 2020], adaptándose para soportar múltiples modelos de datos. También existen alternativas diseñadas desde cero, como ArangoDB, que fue creada específicamente para ser una base de datos multimodelo.

Según [Oracle, 2019], una base de datos multimodelo debe cumplir con las siguientes características:

1. Dar soporte a más de un modelo de datos e, idealmente, a un único lenguaje de consultas.
2. Asegurar que todos los modelos estén disponibles mediante un mecanismo común, evitando a los desarrolladores el esfuerzo de lidiar con interfaces de lógica distinta.
3. Una capa unificada de persistencia de datos, que permite a los diferentes modelos de datos compartir el mismo *backend* de almacenamiento y un *framework* común.
4. Estar diseñada para soportar diversos tipos de aplicaciones y necesidades para empresas de manera ágil.
5. Ser la alternativa más eficiente y conveniente a largo plazo en términos de recursos para ser adoptada en una empresa.

2.2. Arquitecturas internas de BD multimodelo

Al investigar diferentes bases de datos multimodelos se han identificado dos tipos de enfoques fundamentales:

- **Integración total**: Como en ArangoDB, donde todos los modelos comparten un mismo motor de almacenamiento y un mismo lenguaje de consulta (AQL), facilitando interoperabilidad y consistencia interna.
- **Integración parcial**: Como MarkLogic o Azure Cosmos DB, que encapsulan motores específicos (documentos, grafos, relacional) y requieren lenguajes y/o APIs distintas pero ofrecen una capa unificada para el acceso.

La primera opción reduce la latencia en operaciones cruzadas, pero puede implicar compromisos en optimización por tipo de dato. La segunda permite especialización extrema por modelo, pero complica el mantenimiento y pruebas.

Al elegir una arquitectura, también se debe considerar la evolución futura del sistema, ya que los enfoques de integración total suelen ser más fáciles de escalar y mantener a largo plazo, mientras que los parciales pueden requerir ajustes finos al crecer la complejidad.

2.3. Bases de datos multimodelo existentes

Actualmente existen diversas bases de datos multimodelo que soportan variados modelos de datos. La mayor parte de las bases de datos multimodelo existentes nacieron como NoSQL y con el tiempo comenzaron a incorporar múltiples paradigmas de almacenamiento, transformándose así en multimodelo. Además, dentro de las bases de datos multimodelo existentes, por lo general, se combinan 2 o 3 modelos de almacenamiento distintos; son escasos los

ejemplos de gestores que incluyan 4 o más. A continuación se detallan algunas de las bases de datos multimodelo más populares [Software, 2025].

Oracle

Comenzó como un RDBMS y es ampliamente utilizado por organizaciones que requieren integridad transaccional, disponibilidad alta y soporte robusto para cargas críticas.

Oracle ha incorporado soporte para ambientes multimodelo dentro de su núcleo relacional: permite trabajar con documentos JSON, XML, grafos (a través de Oracle Spatial and Graph), y columnas orientadas a analítica (mediante Oracle In-Memory). Todo esto se integra bajo un único motor, sin sacrificar las garantías transaccionales, y ofreciendo un lenguaje SQL unificado para todos los modelos [Oracle, 2019].

Sin embargo, su arquitectura monolítica y los costos de licenciamiento pueden representar barreras para organizaciones pequeñas o con recursos limitados.

Oracle también destaca por sus herramientas avanzadas de replicación, particionamiento, gestión de usuarios, auditoría, recuperación ante desastres y monitoreo en tiempo real.

Redis

Redis (Remote Dictionary Server) es una base de datos NoSQL orientada a clave-valor, diseñada para operar completamente en memoria, lo que le permite alcanzar tiempos de respuesta muy bajos. Aunque se presenta como un almacén clave-valor, admite estructuras de datos avanzadas como listas, conjuntos, *hashes* y *streams*, lo que lo hace apto para casos de uso más allá del simple *caching*. No obstante, su enfoque en memoria implica que puede no ser adecuado para almacenar grandes volúmenes persistentes de datos sin mecanismos externos de respaldo. Redis puede trabajar con otros modelos usando módulos, por ejemplo RedisGraph para grafos o RedisJSON para documentos, transformándola en una base de datos multimodelo. Su arquitectura lo hace ideal para aplicaciones de baja latencia como colas de mensajes, almacenamiento de sesiones, *ranking* en tiempo real y *caching* distribuido [Redis, 2024].

OrientDB

OrientDB comenzó en 2010 como una base de datos pensada para ser multimodelo, es orientada a documentos, grafos, objetos y estructuras relacionales [Group, 2024].

Gracias a su motor unificado, OrientDB permite realizar consultas que combinan modelos documentales y de grafos en una misma operación, sin necesidad de sincronización entre sistemas. Esto la convierte en una solución eficiente para aplicaciones que requieren

una representación rica de relaciones, con baja latencia y buena escalabilidad horizontal [Lomakin y Garulli, 2023].

Así, OrientDB consolidó su posición como una de las primeras plataformas multimodelo nativas, ofreciendo un entorno flexible y potente para representar y consultar datos complejos dentro de una única base de datos coherente.

ArangoDB

ArangoDB se diseñó desde sus inicios como un gestor capaz de soportar múltiples modelos de datos, incluyendo documentos (JSON), grafos y clave-valor. El objetivo era proporcionar flexibilidad en el almacenamiento y consulta de diferentes tipos de datos, eliminando la necesidad de utilizar múltiples bases de datos especializadas [Business, 2025].

Este gestor utiliza su propio lenguaje de consulta, AQL (ArangoDB Query Language), similar a SQL pero capaz de manejar operaciones tanto de grafos como de documentos. Este enfoque unificado facilita a los desarrolladores trabajar con datos de diferentes modelos sin tener que aprender varios lenguajes de consulta o cambiar de contexto [ArangoDB, 2025].

A diferencia de muchas otras bases de datos multimodelo, ArangoDB permite transacciones ACID entre sus modelos de datos, lo que asegura la integridad de las operaciones y datos. Además, es altamente escalable, ofreciendo balanceo de carga automático para manejar grandes volúmenes de datos, lo que le permite adaptarse a las necesidades de escalabilidad del usuario.

ArangoDB ha evolucionado como una solución versátil en las bases de datos multimodelo, proporcionando soporte flexible para varios modelos de datos a través de una plataforma unificada.

PostgreSQL

A diferencia de ArangoDB y OrientDB, PostgreSQL comenzó como un sistema de bases de datos relacional y de código abierto, que con el tiempo comenzó a incorporar otros paradigmas de almacenamiento de datos. Aunque su núcleo sigue siendo relacional, PostgreSQL incorpora de forma nativa soporte para documentos (a través de JSON y JSONB), grafos, vectores, funciones para series de tiempo y extensiones que permiten ampliar sus capacidades hacia modelos más flexibles y no estructurados.

Esta evolución ha sido impulsada por la necesidad de manejar datos cada vez más heterogéneos sin sacrificar la robustez del modelo relacional. PostgreSQL tiene un potente sistema de tipos, permitiendo mediante extensiones mezclar diversos tipos de datos y paradigmas en consultas complejas [IBM, 2021]. Con respecto a la concurrencia, se gestiona mediante el uso de MVCC (Control de concurrencia de múltiples variantes). En la práctica, esto significa

que las lecturas no bloquean las escrituras y las escrituras no bloquean las lecturas.

PostgreSQL ofrece capacidades de indexación específicas para columnas JSONB, soporte para búsquedas full-text, extensiones para manejo de grafos (pgRouting, AGE), y herramientas para particionamiento temporal[dbd,]. Así, es una solución altamente flexible y eficiente para aplicaciones que requieren consultar datos almacenados mediante múltiples paradigmas sin renunciar a la potencia del modelo relacional.

Con más de 35 años de desarrollo, es una base de datos ampliamente utilizada que sigue creciendo y evolucionando.

SurrealDB

SurrealDB es un sistema multimodelo de código abierto lanzado en 2022, que integra datos documentales, de grafos, clave-valor y relacionales bajo un único motor. Su lenguaje de consulta, SurrealQL, combina sintaxis declarativa inspirada en SQL con operaciones multimodelo específicas, como RELATE para grafos. Arquitectónicamente, destaca por su flexibilidad, pudiendo ejecutarse como binario standalone (usando RocksDB), en memoria o en entornos distribuidos. Soporta un modelo *schemaless*, pero también permite definir esquemas explícitos. Entre sus ventajas están su bajo consumo de recursos y despliegue sencillo, aunque como tecnología emergente aún presenta limitaciones en consultas analíticas avanzadas, optimización en alta concurrencia y ecosistema de integraciones. Estas áreas representan líneas activas de desarrollo por parte de la comunidad [SurrealDB, 2022].

En conjunto, SurrealDB se posiciona como una solución innovadora y minimalista dentro del panorama de bases de datos multimodelo, destacándose por combinar flexibilidad, simplicidad y capacidad expresiva en un motor compacto y de rápido despliegue. Este gestor no es muy famoso, pero se incluye en esta memoria por su versatilidad y su novedad.

Las bases de datos multimodelo aún enfrentan desafíos en estandarización, soporte de transacciones cruzadas, optimización avanzada y herramientas de monitoreo, áreas que siguen en evolución activa.

2.4. Estado del arte en modelado lógico aplicado a sistemas multimodelo

El diseño lógico en bases de datos multimodelo es un problema complejo en la ingeniería de datos, ya que implica definir estructuras que integren distintos modelos (relacional, documental, grafos, columnares, etc.) dentro de un mismo esquema. A diferencia de los sistemas relacionales tradicionales, que cuentan con metodologías consolidadas, el diseño lógico en entornos multimodelo aún carece de un marco metodológico unificado. En esta sección se describen los enfoques propuestos por la literatura para abordar esta problemática.

Logical design of multi-model data warehouses (2023)

Bimonte et al. (2023)[Bimonte *et al.*, 2023], define una metodología experimental para definir lineamientos de diseño lógico en *Data warehouses*. Establece que las bases de datos multimodelo pueden lidiar eficientemente con los problemas que conlleva el manejo de *big data* preservando la variedad, volumen y velocidad; lo anterior es beneficial para el almacenamiento de datos en data warehouses. Es así, como los autores usan un caso de estudio adaptando su estructura conceptual de UniBench mediante tablas de hechos (representan a los datos cuantificables a analizar) y dimensiones (representan a los atributos descriptivos que contextualizan los hechos), a partir de este esquema preliminar se realiza una adaptación para tres enfoques lógicos: relacional, documental y de grafos.

El enfoque multimodelo se basa en comparar, para cada tipo de elemento multidimensional, distintas alternativas de modelado lógico. En el caso de modelos documentales, se exploran esquemas denormalizados y fragmentados o *shattered* (colecciones separadas para hechos y dimensiones, unidas por referencias). Para grafos, se analizan esquemas planos y atajos, diferenciados por la presencia de aristas transversales que mejoran el rendimiento de consulta. En el modelo relacional, se utiliza el esquema estrella como referencia base.

Estas alternativas fueron evaluadas mediante un conjunto de consultas que miden el rendimiento, la complejidad de formulación de las consultas, el uso de almacenamiento y la complejidad del proceso ETL. A partir de los resultados, los autores extraen las siguientes conclusiones con respecto al modelado:

Entre los hallazgos más relevantes, destacan que:

- El modelo relacional es superior en términos de almacenamiento y simplicidad del ETL, especialmente para jerarquías simples y no estrictas. Es decir, se puede concluir que es conveniente usar un modelo relacional cuando hay jerarquías fijas, el modelo está normalizado y no hay ambigüedades, la eficiencia compensa la falta de flexibilidad.
- El modelo de grafos es el único que permite ejecutar eficientemente consultas recursivas (como jerarquías de tipo *knows* entre usuarios). Es decir, para relaciones recursivas, caminos entre variables y navegaciones es eficiente el uso de grafos, porque no es necesario especificar el número de saltos entre dos nodos.
- Los documentos JSON ofrecen mayor flexibilidad para representar elementos opcionales, niveles desconocidos o estructuras evolutivas. Es decir, los documentos JSON pueden utilizarse para datos qcuya estructura cambia con el tiempo o tienen campos opcionales sin alterar los demás registros en la BD.

A unified representation and transformation of multi-model data using category theory (2022)

Otro trabajo relevante es el de [Koupil y Holubová, 2022] quienes definen una forma unificada de representar y transformar datos provenientes de distintos modelos de almacenamiento (relacional, documental, clave-valor, columnar y grafo) usando *teoría de categorías*, una rama de las matemáticas que permite describir estructuras y transformaciones de forma general.

A través de esta teoría, los autores proponen un modelo lógico intermedio unificado, capaz de abstraer elementos comunes de los distintos modelos de datos. Además, definen reglas formales de transformación (funtores) que permiten pasar del modelo unificado hacia un modelo lógico.

Aplican su modelo mediante la herramienta MM–cat, que permite transformar automáticamente un esquema ER/UML a distintas formas físicas, y discuten su aplicabilidad a consultas conceptuales, definición de instancias y evolución de esquema.

Si bien este *paper* no se basa en el diseño lógico como tal, y más bien en cómo crear una representación común para los distintos modelos de datos con el fin de ir al diseño físico, ofrece elementos útiles para derivar criterios de diseño lógico en bases de datos multimodelo, que permiten evaluar, desde una perspectiva semántica y estructural, cuál modelo lógico resulta más adecuado para representar una determinada entidad o conjunto de datos, antes de decidir su implementación en un motor físico:

- **Orientación a agregaciones**: Modelos como documental y clave-valor son ideales para representar estructuras autocontenidas que se consultan como una unidad.
- **Profundidad estructural y jerarquía**: El modelo documental permite estructuras anidadas arbitrarias lo que lo hace ideal para representar datos complejos con múltiples niveles de agrupación, a diferencia del relacional o grafo.
- **Homogeneidad de estructuras internas**: Mientras que los modelos relacional, columnar y de grafos tienden a requerir homogeneidad en los tipos de datos, el modelo documental permite mayor flexibilidad, incluyendo listas heterogéneas y atributos opcionales.
- **Representación de relaciones**: Los modelos agregados, como el documental, tienden a utilizar relaciones implícitas mediante anidamiento de datos (es decir, incluyendo un objeto dentro de otro); mientras que los modelos normalizados (relacional, grafo) utilizan referencias explícitas (*foreign keys*, aristas) para representar vínculos entre entidades.

Estos criterios permiten tomar decisiones fundamentadas sobre cómo estructurar lógicamente la información desde un punto de vista teórico.

A Unified Metamodel for NoSQL and Relational Databases (2021)

Por último, Fernández-Candel et al. (2021)[Candel *et al.*, 2022] proponen un metamodelo llamado U-Schema, implementado en EMF/Ecore, junto a una estrategia de extracción que facilita la interoperabilidad y migración entre modelos diferentes, diseñado para representar de forma unificada la estructura de las bases de datos relacionales y NoSQL (documentales, clave-valor, columnares y de grafos). El modelo actúa como una capa lógica intermedia que permite extraer, representar y transformar esquemas. También está orientado a facilitar la interoperabilidad, migración y visualización de esquemas entre distintos paradigmas de almacenamiento.

El paper introduce una estrategia de extracción de esquemas (incluso en sistemas schemaless), y define formalmente cómo representar entidades, relaciones, estructuras anidadas y variaciones dentro de colecciones o modelos heterogéneos. No trata, directamente, sobre bases de datos multimodelo, pero sí es útil para establecer criterios de diseño lógico, pues propone una manera de representar de forma unificada los esquemas de bases de datos relacionales y NoSQL.

A partir de este enfoque, se pueden extraer criterios de diseño lógico:

- **Soporte explícito de variaciones estructurales:** El modelo lógico debe registrar explícitamente las diferencias estructurales que existen entre las instancias de una misma entidad, en lugar de ignorarlas o forzarlas a una estructura rígida.
- **Separación entre entidades y relaciones:** El diseño debe distinguir claramente entre estructuras que representan entidades y aquellas que representan relaciones (por ejemplo, foreign keys o aristas con atributos).
- **Representación diferenciada de agregaciones y referencias:** Es necesario identificar cuándo una estructura debe ser agregada dentro de otra y cuándo debe representarse mediante una referencia externa.
- **Neutralidad tecnológica:** El diseño lógico debe ser independiente de los motores específicos, de forma que el mismo modelo pueda ser proyectado sobre diferentes tecnologías (por ejemplo, MongoDB, Cassandra, Neo4j) sin redefinición estructural.
- **Trazabilidad semántica entre representaciones:** El modelo debe permitir mapear y seguir una misma entidad a través de diferentes representaciones físicas, manteniendo su significado y estructura lógica.

En conjunto, estos trabajos ofrecen enfoques complementarios: (i) guías empíricas y cuantitativas (Bimonte et al.), (ii) un marco formal unificado (Koupil & Holubová) y (iii) una arquitectura de interoperabilidad y extracción (Fernández-Candel et al.). Los criterios extraídos servirán como base para modelar el caso de estudio seleccionado, aplicándolos en los dos motores elegidos y evaluando su aplicabilidad práctica en la metodología propuesta.

Además de los trabajos mencionados, se revisaron otros trabajos que aportan contribuciones menores centradas en aspectos específicos, como optimización de consultas, análisis de rendimiento o adaptaciones para motores particulares. Sin embargo, la mayoría de estos enfoques carecen de generalización metodológica y no abordan de forma integral los desafíos de interoperabilidad, consistencia semántica o neutralidad tecnológica en el diseño lógico multimodelo. Esta limitación general del campo refuerza la necesidad de avanzar hacia metodologías prácticas que permitan guiar decisiones de modelado lógico en escenarios heterogéneos y realistas.

Cabe destacar que, aunque cada enfoque aporta elementos valiosos, ninguno ofrece aún una metodología completamente integrada y probada en contextos heterogéneos. Este vacío motiva la propuesta de la presente memoria, que busca construir y poner a prueba una metodología fundamentada en criterios de diseño lógico multimodelo.

Para ello, es fundamental identificar qué aspectos semánticos deben ser preservados al transformar un modelo conceptual en uno lógico, asegurando que las estructuras seleccionadas representen de forma fiel la información del dominio.

2.5. Dimensiones semánticas relevantes para el diseño lógico

Para que una metodología de diseño lógico pueda preservar adecuadamente la semántica del dominio, es fundamental identificar de forma explícita qué aspectos semánticos deben ser considerados en la transformación [Candel *et al.*, 2022]. A continuación, se presentan las principales dimensiones que es necesario tener en cuenta al crear un modelo lógico multimodelo.

- **Cardinalidad:** Describe cuántas instancias de una entidad pueden estar asociadas a una instancia de otra entidad. Las cardinalidades (1:1, 1:N, N:M) definen la naturaleza de las relaciones y determinan su implementación lógica. Por ejemplo, tabla intermedia, documento embebido, arista en grafo).
- **Grado:** Indica el número de entidades que participan en una relación. Mientras que las relaciones binarias (grado 2) son las más comunes, también pueden existir relaciones ternarias o de mayor grado, las cuales requieren estructuras específicas para representar adecuadamente su semántica.
- **Dependencia existencial:** Una entidad tiene dependencia existencial cuando no puede existir sin estar asociada a otra entidad. Esta propiedad puede reflejarse por ejemplo, con una entidad embebida en otra o restricciones que impidan la creación de registros huérfanos.
- **Unicidad:** Se refiere a la característica de que una entidad o relación sea única, por ejemplo, mediante un identificador en una entidad o mediante mecanismos de restricción de exclusividad, por ejemplo, evitar duplicados en claves compuestas o atributos clave.

- **Temporalidad:** Representa la necesidad de mantener versiones históricas de los datos, ya sea por razones de auditoría, trazabilidad o análisis evolutivo. Se puede implementar mediante snapshots, timestamps o estructuras temporales.
- **Herencia (Jerarquía):** Permite modelar subtipos y su relación con una entidad general (supertipo). Puede implementarse usando herencia explícita en modelos relacionales, estructuras anidadas en documentos o etiquetas de tipo en grafos o clave-valor.
- **Agregación / Composición:** Se refiere a relaciones de pertenencia fuerte o débil entre entidades. La agregación implica una relación estructural sin dependencia vital, mientras que la composición implica que la entidad compuesta no tiene sentido sin la existencia de la otra. Su implementación varía entre claves foráneas o embebido documental.
- **Categorización:** Consiste en organizar instancias en grupos o categorías, como tipos, géneros o etiquetas. Esto permite clasificaciones semánticas, puede representarse como entidades referenciadas, campos enumerados o atributos embebidos.

Estas dimensiones actúan como referentes conceptuales para preservar la integridad semántica del dominio durante el proceso de modelado lógico. El proceso se detallará en los siguientes capítulos.

Capítulo 3: Diseño de la solución

A partir de los enfoques revisados en el estado del arte, se identificaron criterios clave para orientar el diseño lógico en bases de datos multimodelo. Sin embargo, como se discutió previamente, ninguno de estos enfoques ofrece una metodología práctica completamente integrada ni adaptada a la implementación en motores específicos. En este contexto, se propone a continuación una metodología propia que reúne y aporta a estos criterios, proporcionando una guía clara y justificable para transformar modelos conceptuales en modelos lógicos multimodelo aplicables en escenarios reales.

3.1. Fundamento de los criterios propuestos

Los criterios definidos en esta metodología fueron elaborados a partir de una doble fuente: (i) los lineamientos y hallazgos extraídos del estado del arte revisado en el capítulo anterior, y (ii) un análisis propio basado en principios generales de ingeniería de datos, considerando aspectos estructurales (como jerarquía, agregación, heterogeneidad) y semánticos (como roles, dependencia, historial y conectividad). La combinación de ambos enfoques permite contar con una guía que no solo recoge lo reportado en la literatura, sino que lo adapta y complementa para ser aplicable de forma práctica en contextos reales, con el objetivo de orientar decisiones de modelado lógico multimodelo de manera justificable.

3.2. Propuesta de metodología

Para esta metodología, se asumirá que se cuenta de antemano con un modelo conceptual construido usando un diagrama de UML del contexto de los datos, puesto que el objetivo es pasar de un modelo conceptual a un modelo lógico. Se entiende por modelo lógico la asignación de cada componente del dominio (entidad o relación) a un modelo de almacenamiento determinado (relacional, documental, grafo, clave-valor o columnar). Esta asignación se realiza sin entrar aún en detalles específicos de implementación o definición física en un motor de base de datos particular.

Cabe destacar que un mismo componente puede ser representado por más de un paradigma de almacenamiento válido, dependiendo de las características del dominio y de las capacidades del motor seleccionado. Esta metodología propone una guía orientativa que busca ser justificable y flexible, en lugar de establecer reglas únicas de transformación.

En el siguiente punto se abordarán las etapas de la metodología de modelado lógico multimodelo.

Etapas de la metodología de diseño lógico multimodelo

1. **Etapa 1: Análisis del contexto y requisitos del sistema:** Antes de transformar el modelo conceptual, es necesario comprender el contexto en el que la base de datos será utilizada. Esto implica identificar los tipos de consultas que se espera realizar, la naturaleza de las operaciones prioritarias (lectura, escritura, navegación), y los requisitos no funcionales relevantes, como rendimiento, escalabilidad o mantenibilidad.

2. **Etapa 2: Identificación de componentes:** Se deben listar todas las clases de entidad y relaciones del modelo conceptual, considerando sus atributos, cardinalidades, frecuencia de acceso, repeticiones y anidaciones.

3. **Etapa 3: Evaluación de alternativas de modelado:** Es decir, ver cómo puede representarse cada componente según distintos modelos de datos.

4. **Etapa 4: Asignación de modelos de almacenamiento:** Asignar modelos de almacenamiento, escogiendo el paradigma más adecuado para cada componente según los criterios definidos.

A continuación se desarrollan en detalle las etapas 3 y 4, dado que requieren una mayor cantidad de pasos y consideraciones específicas para su correcta ejecución. En contraste, las etapas 1 y 2 corresponden a procesos más acotados, cuyo alcance ya quedó descrito previamente, por lo que no requieren mayor desagregación.

Etapa 3: Evaluación de alternativas de modelado: Por cada clase de entidad y cada relación es necesario responder, según corresponda, cómo puede representarse en base a las alternativas posibles:

Para clases de entidad:

- Tiene estructura variable o campos opcionales? → **Documental**
  Los documentos permiten representar registros heterogéneos sin requerir un esquema rígido, otorgando flexibilidad a cambios[Candel *et al.*, 2022].

- Presenta jerarquías o estructuras anidadas profundas? → **Documental o Grafo**
  El modelo documental admite anidaciones naturales, mientras que el modelo de grafos es preferible cuando se requiere navegar entre niveles de jerarquía [Koupil y Holubová, 2022].

- Tiene estructura homogénea y estable? → **Relacional**
  El modelo relacional ofrece eficiencia y claridad para estructuras bien definidas, especialmente con integridad referencial y tipos de datos fijos [Bimonte *et al.*, 2023].

- Se consulta siempre junto a otra entidad específica? → **Documental**
  Anidando los datos se evita la necesidad de realizar joins u operaciones costosas para cruzar datos [MongoDB, 2021].

- Esta clase de entidad participa en múltiples relaciones importantes con otras entidades o cumple un rol central en el dominio? → **Grafo**
  El modelo de grafos facilita las consultas sobre nodos centrales y relaciones múltiples, optimizando el rendimiento en contextos altamente conectados [Anuyah *et al.*, 2024].

- Se realizan frecuentemente consultas que agrupan o realizan operaciones por atributos específicos? → **Columnar**
  La representación columnar está optimizada para operaciones analíticas, como agregaciones por columnas, que aprovechan su almacenamiento orientado a columnas [Meier y Kaufmann, 2019].

- Normalmente se accede a esta clase de entidad directamente a partir de un identificador, sin necesidad de recorrer relaciones? → **Clave-valor**
  El modelo clave-valor ofrece búsquedas directas con baja latencia, siendo útil para accesos simples y repetitivos [Meier y Kaufmann, 2019].

- Se necesita consultar rangos de valores (por ejemplo, fechas o puntuaciones) o acceder repetidamente a ciertas columnas? → **Columnar**
  El modelo columnar ofrece un escaneo eficiente para consultas analíticas sobre subconjuntos específicos de atributos [Bathla *et al.*, 2018].

- Esta clase de entidad puede tener múltiples versiones a lo largo del tiempo o su estructura cambia entre instancias? → **Documental o Columnar**
  Los documentos permiten estructuras flexibles, mientras que los modelos columnares son eficaces para versiones temporales [Groves, 2025] [mon, 2021].

- Esta clase de entidad representa información complementaria que solo cobra sentido en el contexto de su relación con otras dos entidades? → **Grafo**
  La información complementaria podría ser el vínculo entre dos entidades relacionadas y almacenarse allí [Candel *et al.*, 2022].

- La entidad necesita conservar historial o versiones de sus registros en el tiempo? → **Columnar o Documental**
  Si el historial se consulta para análisis o reportes por fechas, preferir columnar. Si cada versión tiene estructura flexible o debe mantenerse junto con el registro principal, preferir documental [Groves, 2025] [mon, 2021].

  La subclase de entidad contiene pocos atributos que la diferencien de la superclase? → **No modelar como entidad separada**
  Cuando un subtipo no introduce nuevos atributos ni relaciones independientes, puede omitirse como entidad separada. En su lugar, su rol debe representarse a través de la relación que lo conecta con otras entidades (ej. arista en grafo). Esto mantiene la jerarquía semántica sin fragmentar la estructura ni introducir complejidad innecesaria [Candel *et al.*, 2022].

Para relaciones: Por cada relación identificada en el modelo conceptual, se considera la estructura y función que cumple dentro del dominio. Existen varias formas de representar relaciones en entornos multimodelo:

- Representa una relación recursiva o navegable en múltiples saltos 1:N o N:M? → **Grafo (arista)**
  Los modelos de grafos permiten representar y recorrer eficientemente relaciones au- torreferenciales o de profundidad variable mediante navegación por aristas [Bimonte *et al.*, 2023].

- Es una relación 1:1 con fuerte dependencia semántica? → **Documental (doumento anidado)**
  Cuando una entidad secundaria no tiene sentido sin su entidad principal, puede ser embebida en ella [Koupil y Holubová, 2022].

- Es una relación N:M entre entidades independientes? → **Grafo (arista) o Relacional (tabla intermedia)**
  Si se requiere una navegación dinámica entre entidades es conveniente usar grafos; si se prioriza consulta estructurada o integridad referencial, es mejor una tabla relacional [Koupil y Holubová, 2022].

- Hay una relación implícita a través de identificadores compartidos o es una relación 1:1, 1:N o N:M? → **Relacional, Documental, Grafo, Clave valor (Referencias cruzadas por ID)**
  En casos donde múltiples entidades están relacionadas mediante claves compartidas, no es necesario modelar explícitamente la relación. La trazabilidad se conserva si los ID se mantienen como referencias entre documentos, tablas o nodos [Candel *et al.*, 2022].

- Requiere trazabilidad, referencias cruzadas o es una relación 1:N o 1:1? → **Relacional, Documental, Grafo, Clave valor (Asignar ID compartido en las clases de entidad)**
  Para mantener coherencia e interoperabilidad entre paradigmas, se recomienda el uso de claves compartidas o identificadores comunes que permitan rastrear elementos [Candel *et al.*, 2022].

- Es una clase de asociación con atributos propios entre entidades o una relación N:M? → **Relacional (tabla) o Grafo (arista con propiedades)**
  Dependiendo del tipo de entidades involucradas, puede ser conveniente colocar los datos de la clase en una tabla relacional o bien almacenar la información en la arista entre las entidades. Esto permite mantener los detalles contextuales de la interacción entre entidades [Bimonte *et al.*, 2023].

- Existen jerarquías o relaciones de herencia entre entidades? → **Documental, Relacional o Grafo**
  Cuando hay subtipos con estructuras similares, se puede usar documentos con un

campo discriminador (‘type‘) para contener todas las variantes. Si los subtipos son muy distintos o deben mantenerse independientes, es preferible usar relaciones 1:1 en un modelo relacional para simularla. También puede modelarse con una propiedad tipo o creando nodos específicos por subtipo en un grafo, con una arista que los vincula al supertipo en el caso de que se priorice la facilidad de navegación entre los subtipos pensando en navegación de múltiples saltos [Chillón *et al.*, 2021].

Etapa 4: Asignación de modelos de almacenamiento: asignar a cada componente del dominio un modelo lógico correspondiente según las respuestas obtenidas.

En los casos donde se identifiquen múltiples opciones viables para representar un mismo componente, la selección debe considerar el sistema en su conjunto. No existe una única opción correcta, ya que la decisión dependerá de cómo se equilibran los requisitos funcionales, los patrones de consulta, las restricciones tecnológicas y los objetivos de rendimiento. Es responsabilidad del diseñador determinar qué aspectos priorizar y qué compromisos asumir. Por ello, a continuación se presentan criterios generales a considerar que permten elegir la alternativa más adecuada en cada situación particular.

- **Complejidad operativa:** Priorizar alternativas que minimicen el esfuerzo requerido en términos de consultas, mantenimiento y evolución futura del sistema, evitando estructuras que generen dependencias frágiles o redundancias innecesarias [Candel *et al.*, 2022]. Se sugiere Documental en el caso de priorizar el minimo esfuerzo al hacer consultas y relacional para el mantenimiento.
- **Adecuación al objetivo del sistema:** Seleccionar la representación que mejor se ajuste a los requisitos clave del sistema. Por ejemplo, si se prioriza la navegabilidad entre entidades, es preferible optar por un modelo de grafos frente a una estructura relacional, según los criterios discutidos previamente [Pore, 2019].
- **Método de consulta dominante:** Si los datos se consultan frecuentemente como unidad, favorecer modelos embebidos; Si se acceden por claves simples o mediante agregaciones, considerar modelos clave-valor o columnar [Bimonte *et al.*, 2023].
- **Frecuencia de lectura vs escritura:** Para lecturas frecuentes con pocos cambios, preferir modelos documentales; para escrituras o actualizaciones parciales frecuentes, priorizar estructuras relacionales o columnar [Hecht y Jablonski, 2011].
- **Redundancia aceptable vs mantenibilidad:** Evitar duplicaciones a menos que se justifiquen para mejorar el rendimiento; si la consistencia entre duplicados es crítica, evitar embebido [Sadalage y Fowler, 2013].
- **Conectividad del componente** Si el componente se relaciona con muchas otras entidades o se requiere navegación compleja, favorecer grafos frente a otras alternativas [Bimonte *et al.*, 2023].

- **Compatibilidad con el motor seleccionado:** Elegir representaciones compatibles con los modelos nativos u optimizados del gestor de base de datos seleccionado [Sadalage y Fowler, 2013]. Por ejemplo, si es necesario usar PostgreSQL, al haber sido creada inicialmente como base de datos relacional, cuenta con mayor soporte en ese modelo de almacenamiento que en grafos.
- **Interoperabilidad y trazabilidad:** En contextos multimodelo, si una entidad aparece en más de un modelo, asegurar su identificación mediante claves comunes [Candel *et al.*, 2022].
- **Generalidad del dominio:** Considerar cómo se conectan los demás elementos del dominio y la facilidad de un paradigma u otro de encajar [Chillón *et al.*, 2021]. Por ejemplo, si una entidad está relacionada en su mayoría con entidades que se modelan como nodos, conviene considerarla como tal también.

Los criterios de decisión provienen tanto del análisis del estado del arte como de propuestas propias basadas en el comportamiento estructural y semántico de los datos. Esta metodología es extensible a otros dominios siempre que se cuente con un modelo conceptual previo y se documente claramente el proceso de asignación.

Nota sobre flexibilidad: Cabe destacar que esta metodología no pretende ser prescriptiva ni única, sino servir como guía flexible. La asignación de paradigmas puede variar según prioridades del proyecto (rendimiento, mantenibilidad, costos, equipo disponible) y siempre requiere un análisis contextualizado.

En definitiva, se busca ofrecer una herramienta práctica que permita a diseñadores e ingenieros aplicar principios teóricos en contextos reales, contribuyendo así al avance en el campo del modelado lógico multimodelo.

3.3. Plan de trabajo para la implementación

Las siguientes etapas fueron definidas considerando tanto los objetivos identificados en el capítulo 1 como la necesidad de aplicar la metodología a un escenario práctico real. Se busca no solo diseñar un modelo lógico multimodelo, sino también evaluar su aplicabilidad, robustez y flexibilidad en entornos concretos.

1. **Selección de un caso de estudio:** Se elige un conjunto de datos reales con variedad estructural, de forma que sea posible aplicar distintos modelos de almacenamiento.
2. **Modelado conceptual del caso de estudio:** Se modela el dominio en notación UML, representando entidades, atributos, relaciones, y constructos avanzados como jerarquías, composiciones o relaciones temporales.

3. **Transformación al modelo lógico multimodelo:** A partir del modelo conceptual, se aplica la metodología propuesta para asignar a cada componente del dominio (entidad o relación) el paradigma de almacenamiento más adecuado (relacional, documental, grafo, clave-valor o columnar).

4. **Análisis cualitativo del diseño lógico:** Se evalúa el modelo lógico obtenido revisando si se conservaron correctamente las 8 dimensiones semánticas. Se identifican fortalezas, ambigüedades y compromisos asumidos en el diseño. Luego se implementará el modelo en dos motores de bases de datos multimodelo distintos para comprobar la eficacia del modelo final, comparando el desempeño de ambos motores.

Capítulo 4: Desarrollo de la solución

En este capítulo se aplica la metodología de diseño lógico multimodelo al caso de estudio del dataset IMDb, detallando el proceso de análisis, transformación del modelo conceptual, decisiones de modelado tomadas y su implementación en dos gestores de bases de datos multimodelo: ArangoDB y SurrealDB.

4.1. Selección de un caso de estudio

Como caso de estudio se utilizo *The IMDb dataset* [IMDB, 2020], un conjunto de archivos con información detallada acerca de películas, series y videojuegos. IMDb fue seleccionado como caso de estudio debido a su estructura rica y heterogénea, que combina entidades de diferentes naturalezas (películas, personas, roles, calificaciones, versiones) y relaciones complejas, como jerarquías, asociaciones N:M y atributos multivalorados. Esta complejidad lo convierte en un escenario ideal para evaluar las capacidades y limitaciones de un diseño lógico multimodelo. En la tabla 2 se describe el contenido de cada archivo del *dataset* y, posteriormente, se detalla la descripción de sus campos.

Archivo	**Descripción**	**Campos principales**
`title.basics.tsv`	Información básica de títulos de películas, series y cortos.	`tconst`, `titleType`, `primaryTitle`, `originalTitle`, `isAdult`, `startYear`, `endYear`, `runtimeMinutes`, `genres`
`name.basics.tsv`	Datos biográficos de personas relacionadas con títulos (actores, directores, escritores, etc.).	`nconst`, `primaryName`, `birthYear`, `deathYear`, `primaryProfession`, `knownForTitles`
`title.principals.tsv`	Asocia personas a títulos con sus roles específicos.	`tconst`, `nconst`, `category`, `job`, `characters`
`title.crew.tsv`	Detalla directores y guionistas por título.	`tconst`, `directors`, `writers`
`title.ratings.tsv`	Calificaciones promedio y número de votos por título.	`tconst`, `averageRating`, `numVotes`
`title.akas.tsv`	Versiones alternativas de títulos en distintos idiomas o regiones.	`titleId`, `ordering`, `title`, `region`, `language`, `types`, `attributes`

Tabla 2: Descripción de los archivos del dataset IMDb

El detalle de los campos por cada archivo se encuentra en el Anexo.

Gestores de bases de datos a utilizar

Como se explicó en la sección anterior, el dataset de IMDb permite evaluar el diseño y comportamiento del modelo lógico bajo arquitecturas con características distintas, gracias a su combinación de entidades heterogéneas, relaciones complejas y necesidades de navegación recursiva.

Es por ello, que se utilizarán los siguientes gestores de bases de datos:

1. **ArangoDB:** Se incluye por ser una base de datos multimodelo nativa que soporta modelado documental, de grafos y clave-valor. Esto permite representar documentos complejos, relaciones explícitas entre entidades y estructuras jerárquicas o recursivas dentro de un mismo sistema, utilizando un solo lenguaje de consulta (AQL). Además es una base de datos que existe hace más de 10 años (lanzada en 2012) por lo que es un referente de las bases de datos multimodelo
2. **SurrealDB:** Se selecciona porque, al igual que ArangoDB, es una base de datos multimodelo nativa, capaz de integrar modelado documental, de grafos, clave-valor y relacional bajo un único motor. SurrealDB destaca por ser una tecnología emergente (lanzada en 2022), diseñada para ofrecer simplicidad de despliegue junto con un bajo consumo de recursos.

La inclusión de ArangoDB junto a SurrealDB resulta particularmente interesante, ya que permite comparar enfoques consolidados y maduros (ArangoDB, con más de 10 años de desarrollo) con soluciones innovadoras y recientes, analizando así no solo capacidades funcionales, sino también diferencias en madurez tecnológica, ecosistema, documentación y soporte.

Dicha combinación permite no solo evaluar la aplicabilidad de la metodología propuesta en entornos contrastantes, sino también obtener *insights* sobre las diferencias prácticas en diseño, implementación y mantenimiento en tecnologías multimodelo de distinta madurez.

4.2. Modelado conceptual del caso de estudio

Para el modelo conceptual del caso de estudio se tiene una clase de entidad por cada componente estructural del dataset IMDB, considerando como tales aquellas secciones del conjunto de datos que poseen una clave identificadora única, atributos relevantes y existencia semántica propia; además, se crearon entidades extras para modelar componentes del dataset más complejos. Es así que se obtuvieron 15 entidades y 11 relaciones de diversas cardinalidades, siguiendo la notación UML. Los tipos de datos fueron asignados de forma acorde a la

documentación oficial del dataset y representan, en su mayoría, cadenas de texto, números y bool.

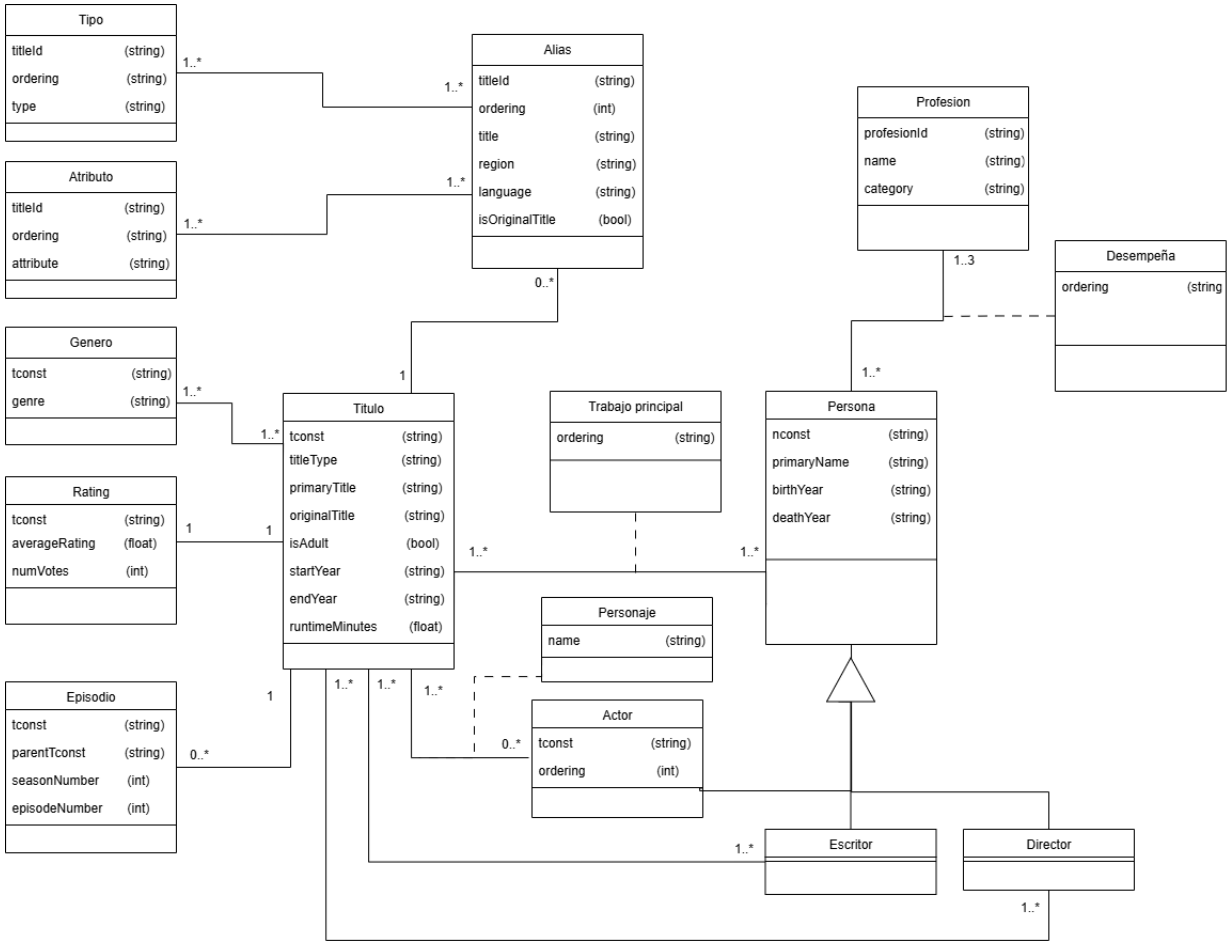


Figura 2: Modelo conceptual del caso de estudio en notación UML

Fuente: Elaboración propia

Este modelo conceptual busca preservar la semántica del dominio, manteniendo explícitas las relaciones de participación (por ejemplo, personas asociadas a títulos), la jerarquía de conceptos (persona con sub roles como Actor) y las relaciones que almacenan datos específicos, de modo que su transformación posterior a un modelo lógico multimodelo no implique pérdida de significado ni ambigüedad estructural.

Las clases de entidad del modelo son las siguientes:

1. **Título:** Representa obras audiovisuales como películas, series o episodios. Contiene información básica como nombre, año, tipo, duración y género.
2. **Alias (Akas):** Títulos alternativos que puede tener una obra según idioma, país o contexto cultural.

3. **Ratings:** Información sobre la calificación promedio y la cantidad de votos que ha recibido una obra.
4. **Episodios:** Describe la jerarquía de episodios dentro de series o temporadas.
5. **Trabajo principal:** Clase de asociación que relaciona una persona con los títulos por los que es conocida, incluyendo un campo de orden para definir relevancia.
6. **Profesión:** Representa los distintos roles que una persona puede desempeñar en la industria (actor, director, editor, etc.), asociados mediante relaciones explícitas.
7. **Desempeña:** Clase de asociación que vincula a una persona con una o más profesiones, especificando el orden o prioridad.
8. **Tipo – Atributo:** Entidades auxiliares que permiten especificar el tipo de alias y sus atributos asociados, conservando la flexibilidad del dataset original pero sin usar arrays.
9. **Género:** Representa categorías como “drama”, “comedia” o “acción” asociadas a un título. Se modela como una relación N:M.
10. **Personas:** Individuos que participan en los títulos como actores, directores o escritores. Representa la entidad base para subtipos según el rol desempeñado.
11. **Actor:** Subclase de Persona que representa el rol específico de actuación en una obra. Se registra la posición en los créditos y el personaje interpretado.
12. **Personaje:** Entidad asociada al personaje interpretado por un actor en un título. Permite separar el personaje como concepto y reutilizarlo si es necesario.
13. **Escritor:** Subclase de Persona que representa a quienes han participado en el guion o contenido narrativo de un título.
14. **Director:** Subclase de Persona que representa a quienes dirigen la producción de una obra.

Por otro lado, las relaciones representadas en el modelo son las siguientes:

1. **Alias–Título:** Un título puede tener 0 o más alias (versiones alternativas del mismo título en diferentes regiones o idiomas), pero cada alias se asocia a un título único.
2. **Tipos–Alias:** Cada alias puede estar asociado a uno o más tipos que describen información adicional asociada (por ejemplo, puede indicar que el alias para un título esta en formato DVD) es una relación N:M.
3. **Atributo–Alias:** Similar a la relación con Tipos, cada alias puede tener uno o más atributos (por ejemplo, traducción literal, subtítulo, abreviación), asociados mediante la tabla de atributos. Es una relación N:M.

4. **Género–Título:** Un título puede estar asociado a uno o más géneros (por ejemplo, comedia, drama), y cada género puede estar asociado a múltiples títulos. Esta relación N:M permite caracterizar el contenido.

5. **Ratings–Título:** Existe una relación 1:1 entre cada título y su calificación. Ambas entidades dependen mutuamente de su significado en el dominio.

6. **Episodio–Título:** Un título puede no tener episodios (por ejemplo, si es una película), pero un episodio siempre está asociado a un título principal que lo contiene. Esta relación representa una jerarquía entre series (título) y episodios.

7. **Persona–Título (Trabajos principales):** Una persona puede ser conocida por varios títulos y un título puede estar en varios trabajos principales de una persona. Se representa como una relación N:M, donde se incluye un campo de orden para indicar la relevancia.

8. **Actor–Título (Personajes):** Un título puede tener 0 o más actores (por ejemplo, si el título no tiene personajes), mientras que cada actor puede aparecer en uno o más títulos. Se representa como una relación N:M, almacenando el nombre del personaje según corresponda.

9. **Herencia: Persona, Actor, Escritor y Director:** La superclase persona almacena la información básica de Actor, Escritor y Director.

10. **Escritor–Título:** Un escritor puede haber trabajado en uno o más títulos, y un título puede tener múltiples escritores. Es una relación N:M.

11. **Director–Título:** Similar a la anterior, un título puede tener múltiples directores y cada director puede haber dirigido varios títulos. También es una relación N:M.

12. **Persona–Profesión (Desempeña):** Una persona puede tener de una a tres profesiones, representadas como una relación N:M hacia Profesión. Se modela como clase de asociación para almacenar el orden en que estas profesiones se listan, respetando la semántica del campo ‘primaryProfession‘.

Este modelo conceptual servirá como base para la aplicación de la metodología propuesta. A partir de él se buscó mantener la consistencia semántica del dominio, incluyendo las semánticas de cardinalidad, grado, unicidad, herencia, dependencia existencial, agregación, tiempo y categorización, al pasar a un esquema lógico con distintos paradigmas de almacenamiento.

Esta fase garantiza que la transformación posterior no sea solo estructural, sino también semánticamente coherente, permitiendo evaluar de forma precisa los beneficios y desafíos del enfoque multimodelo.

En la siguiente etapa se procederá a transformar el modelo conceptual aplicando la metodología propuesta.

4.3. Transformación al modelo lógico multimodelo

Por cada elemento identificado se asignó a cada componente el paradigma de almacenamiento mas adecuado. Se responden las preguntas especificadas en la metodología, considerando que los motores a utilizar son ArangoDB y SurrealDB, a través del siguiente análisis:

4.3.1. Clases de entidad

- **Título (relacional o grafo):** Se trata de una entidad central, con atributos homogéneos y una estructura estable. Para consultas frecuentemente se usa el identificador tconst para extraer información de otras entidades relacionadas. Es por ello que puede ser modelada con enfoque relacional (tabla) o como grafo (nodo).
- **Alias (Akas) (relacional o documental):** Es una entidad subordinada a Título, sigue una estructura fija y estable. También se consulta junto a la entidad Título y puede tener múltiples versiones a lo largo del tiempo. Es por ello que puede ser modelada con enfoque relacional o documental.
- **Ratings (relacional, documental o clave valor):** Es una entidad que se consulta en relación a Título estando fuertemente asociada. También sigue una estructura fija y estable, pero es posible acceder a ella directamente mediante el identificador tconst. Es por ello que puede ser modelada con enfoque relacional, documental o clave-valor.
- **Género (relacional, documental o clave valor):** Esta entidad tiene estructura estable y se consulta junto a la entidad Título, pero es accesible mediante el identificador tconst. Es por ello que puede representarse con enfoque relacional, documental o clave-valor.
- **Episodios (relacional o grafo):** Esta entidad tiene estructura estable, se consulta junto a Título y estan altamente relacionadas. Es por ello que puede representarse con un enfoque relacional o como grafo.
- **Trabajo principal:** Esta es una clase de asociación ligada a la relación entre Persona y Título, se analizará posteriormente con las demás relaciones.
- **Profesión (relacional o documental):** Esta entidad tiene estructura estable y se consulta junto a la entidad Persona. Es por ello que puede representarse con enfoque relacional o documental.
- **Desempeña:** Esta es una clase de asociación ligada a la relación entre Persona y Profesión, se analizará posteriormente con las demás relaciones.
- **Tipo (relacional o documental):** Esta entidad tiene estructura estable que no cambia en el tiempo y por lo general se consulta junto a la entidad Título. Es por ello que puede representarse con enfoque relacional o documental.

- **Atributo (relacional o documental):** Al igual que Tipo, esta entidad tiene estructura estable y se consulta junto a la entidad Título. Es por ello que puede representarse con enfoque relacional o documental.
- **Personas (relacional, documental o grafo):** Esta entidad tiene estructura homogénea, pero también es superclase de Actor, Escritor y Director. Es por ello que puede ser representada con enfoque relacional, documental o grafo.
- **Actor (relacional, documental, grafo o no representar como entidad separada de la super clase):** Esta entidad es subclase de Persona, al igual que ella puede representarse con enfoque relacional, documental o grafo. Pero también cuenta con poca diferenciación estructural con respecto a su superclase asique puede no modelarse como entidad separada
- **Personaje:** Esta entidad esta asociada a la relación entre Actor y Título, se analizará posteriormente con las demás relaciones.
- **Escritor (relacional, documental, grafo o no representar como entidad separada de la super clase):** Similar a actor, esta entidad puede representarse con enfoque relacional, documental o grafo. Pero también es un subtipo semántico con poca diferenciación estructural asique puede no modelarse como entidad separada
- **Director (relacional, documental, grafo o no representar como entidad separada de la super clase):** Similar a actor, esta entidad puede representarse con enfoque relacional, documental o grafo. Pero también es un subtipo semántico con poca diferenciación estructural asique puede no modelarse como entidad separada

4.3.2. Relaciones

Teniendo presente que la mayor parte de las entidades tienen en común identificadores tconst y nconst está la posibilidad de utilizar una clave compartida o id común entre algunas relaciones. Se analizan las relaciones entre entidades, especificando cuando podría ser conveniente agregar un id común:

- **Alias–Título:** Es una relación 1:N fuertemente relacionada a Título, además tienen id compartido por lo que puede representarse como un documento embebido o como referencias cruzadas.
- **Tipo–Alias:** Es una relación donde tipo depende fuertemente de alias, pero además hay identificadores compartidos por lo que puede representarse como documental o como referencias cruzadas.
- **Atributo–Alias:** Es una relación donde atributo depende fuertemente de alias, pero además hay identificadores compartidos por lo que puede representarse como documental o como referencias cruzadas.

- **Género–Título:** Es una relación donde Género depende fuertemente de Título, pero además hay identificadores compartidos por lo que puede representarse como documental o como referencias cruzadas.
- **Ratings–Título:** Es una relación 1:1 y esta fuertemente relacionada a título, pero además tienen identificadores compartidos por lo que puede ser representada como documento embebido o como referencias cruzadas.
- **Episodio–Título:** Ambas entidades tienen identificadores compartidos por lo que es posible utilizar referencias cruzadas. Además es posible usar documentos embebidos dada la dependencia fuerte entre episodio y título.
- **Persona–Título (Trabajos principales):** Es una relación N:M entre entidades independientes, pero tiene una clase de asociación con atributos propios, por lo que se puede representar como tabla intermedia o como arista en grafos.
- **Actor–Título (Personajes):** Es una relación N:M entre entidades independientes, pero tiene una clase de asociación con atributos propios, por lo que se puede representar como tabla intermedia o como arista en grafos. También tiene un identificador compartido por lo que es posible usar referencias cruzadas
- **Escritor–Título:**Es una relación N:M entre entidades independientes por lo que puede modelarse mediante una arista en un grafo, pero también puede requerir trazabilidad, por lo que podría convenir asignar un ID compartido para hacer referencias cruzadas (en este caso tconst a Escritor)
- **Director–Título:**Es una relación N:M entre entidades independientes por lo que puede modelarse mediante una arista en un grafo, pero también puede requerir trazabilidad, por lo que podría convenir asignar un ID compartido para hacer referencias cruzadas (en este caso tconst a Director)
- **Persona–Profesión (Desempeña):** Es una relación N:M entre entidades independientes, pero tiene una clase de asociación con atributos propios, por lo que se puede representar como tabla intermedia o como arista en grafos.
- **Herencia Persona, Actor, Director, Escritor:** En la sección anterior se aborda el modelado de cada entidad, al ser herencia esta puede modelarse como documental, relacional o como grafo.

4.3.3. Selección final del modelo lógico para cada entidad y relación

Luego de analizar las alternativas posibles de representación para cada componente del dominio, se definió una representación principal priorizando la coherencia con el resto del modelo lógico, la compatibilidad nativa o eficiente con al menos uno de los motores seleccionados y la preservación de las dimensiones semánticas (unicidad, herencia, cardinalidad, etc.).

Clases de entidades:

- **Título → Grafos (nodo)**
  Se opta por modelar Título como un nodo debido a su rol central en el dominio y a su participación en múltiples relaciones con otras entidades clave, como Personas, Episodios y Ratings. Representarlo como nodo permite navegar eficientemente las relaciones laborales y jerárquicas (por ejemplo, desde una persona a los títulos en los que ha trabajado, o desde un título a sus episodios o reparto). Además, el uso de grafos favorece consultas recursivas, como encontrar todos los colaboradores indirectos de una persona a través de sus títulos compartidos. Esta decisión potencia la expresividad y navegabilidad del modelo, priorizando las necesidades de exploración semántica por sobre la rigidez estructural.

- **Alias (Akas) → Relacional (tabla)**
  Aunque inicialmente se podría considerar su modelado como documento embebido en Título, el volumen elevado de alias por cada título, sumado a su estructura homogénea y la necesidad potencial de consultas independientes (por idioma, región o tipo), justifican su representación como tabla relacional.

- **Ratings → Clave-Valor**
  Ratings es accedida normalmente de forma directa por `tconst`, sin requerir relaciones complejas. No hay anidaciones, por lo que su estructura puede representarse eficientemente en un modelo de clave-valor si se prioriza rendimiento en lecturas.

- **Género → Relacional (Tabla)**
  Representa clasificaciones reutilizables entre múltiples títulos. La separación permite mantener una taxonomía clara, y evita repetir los mismos géneros por documento.

- **Tipo y Atributo → Documental (anidado en Alias)**
  Aunque Tipo y Atributo pueden modelarse como tablas por su reutilización, en este caso se opta por embebidos documentales dentro de Alias para favorecer la cohesión semántica y eliminar joins.

- **Personas → Grafo (nodo)**
  Se modela Personas como nodo debido a su rol como entidad con alta conectividad, especialmente en su relación con Títulos a través de la entidad Actores. Aunque su estructura es homogénea y podría representarse relacionalmente, su representación como nodo permite una navegación eficiente desde y hacia otros elementos, permitiendo explorar rutas múltiples. Además, al emparejarse con Título como nodo, se habilita una arquitectura coherente donde Actores funciona como arista,favoreciendo consultas complejas de navegación.

- **Actor, Director, Escritor → No se modelan como entidades separadas**
  Se considera la relación que tienen con la entidad Persona. El rol (actor, director, escritor) se almacena en la arista que los conecta con Título (Actor → Actúa, Director →

Dirige, etc.). Se elige porque esta alternativa no fragmenta la estructura y considerando las entidades relacionadas simplifica y facilita la conectividad en el modelo final, minimizando la redundancia de información.

- **Episodios → Grafo (nodo)**
  Al estar subordinado a Título, la representación como nodo es conveniente en un grafo. Si bien puede usarse una tabla con clave foránea, el grafo permite navegar rápidamente entre temporadas y episodios.

- **Profesión → Relacional (Tabla)**
  Catálogo reutilizable de categorías laborales. Su separación como tabla favorece consistencia y facilita actualización independiente del resto del modelo.

Relaciones

- **Título – Alias (Akas) → Referencia mediante ID compartido (tconst)**
  Alias se encuentra almacenado como una tabla separada en el modelo relacional, pero se vincula con Título mediante el ID común. No se representa como arista ni como documento embebido ya que se prioriza su reutilización y consultas independientes (por idioma, región, etc.). Esta relación se gestiona por trazabilidad a través de tconst, sin necesidad de estructuras complejas.

- **Título – Ratings → Referencia mediante ID compartido (tconst)**
  La entidad Ratings se modela en un sistema de clave-valor y contiene métricas asociadas directamente a un Título. Al compartirse el identificador tconst, se puede acceder a los datos de forma eficiente.

- **Título--Género→ Referencia mediante ID compartido en tabla intermedia**
  La relación entre título y género se representa mediante una tabla intermedia que almacena referencias por ID, sin generar duplicación de datos en cada título y simplificando la implementación de la relación. Además facilita análisis basados en categorías (ej. todos los títulos que sean de terror)

- **Título--Episodio → Grafo (arista)**
  La relación jerárquica entre Título y Episodio se modela como una arista directa, facilita el recorrer relaciones con múltiples saltos, permitiendo recorridos jerárquicos, como identificar episodios de una serie o navegar entre temporadas.

- **Persona – Título (Trabajos principales) → Grafo (arista)**
  Esta relación representa la participación de personas en obras específicas, se modela como una arista con propiedades. Este enfoque permite recorrer el grafo eficientemente en ambos sentidos, optimizando queries como "¿Qué personas actuaron en X título?" o "¿En qué títulos ha trabajado Y persona?".

- **Actor (subclase Persona)–Título $\rightarrow$ Grafo (arista)**
  Contiene como atributo el nombre del personaje interpretado. Se modela como arista para aprovechar recorridos actorales.

- **Escritor (subclase Persona)–Título / Director (subclase Persona)-Título $\rightarrow$ Grafo (arista)**
  Similares a la relación de Actor, pero sin atributos. Su separación semántica como aristas facilita consultas como "¿Quién escribió esta película?"

- **Persona–Profesión $\rightarrow$ Referencia mediante ID compartido en tabla intermedia**
  Persona corresponde a un nodo, mientras que Profesión es una tabla, es por ello que se utiliza una tabla intermedia con los ID de ambas clases, agregando el campo ordering.

En la figura 3, se presenta un esquema que muestra los paradigmas de almacenamiento seleccionados para cada clase de entidad del modelo, junto con la representación asignada a cada relación. En el caso particular de las subclases de Persona (Actor, Director, Escritor), dado que se decidió no modelarlas como entidades independientes, sino como una propiedad dentro de las aristas que conectan Persona con Título, no se les ha asignado un color ni un contorno específico en el diagrama.

Nota adicional: Dado que actor, escritor y director quedaron como atributos de la arista en el grafo no se colorean las clases individuales en la imagen. Es importante considerar que la elección final del modelo lógico no debe interpretarse como la única posible, sino como el resultado de un proceso reflexivo orientado a priorizar la navegabilidad entre entidades clave del dominio (como Persona y Título) y a preservar la consistencia semántica del modelo conceptual. A lo largo del análisis se consideraron otras opciones viables, como una estructura basada principalmente en documentos embebidos, que habría reducido la cantidad de relaciones explícitas a costa de perder flexibilidad. También se evaluó la posibilidad de utilizar un modelo puramente basado en grafos, pero ello implicaría una pérdida de eficiencia en consultas analíticas o tabulares.

En cambio, el modelo adoptado busca un equilibrio entre la expresividad del grafo para relaciones actorales y jerárquicas, la velocidad del enfoque clave-valor, y la solidez del modelo relacional en componentes como Alias.

Esta decisión se apoya en el uso de identificadores comunes (tconst y nconst) y en el análisis explícito de cada componente según sus atributos, estructura y uso, permitiendo mantener la trazabilidad y coherencia en entornos multimodelo sin comprometer la fidelidad del dominio.

El modelo resultante es altamente heterogéneo, pero respeta las dimensiones semánticas expuestas, lo cual es el objetivo de esta metodología.

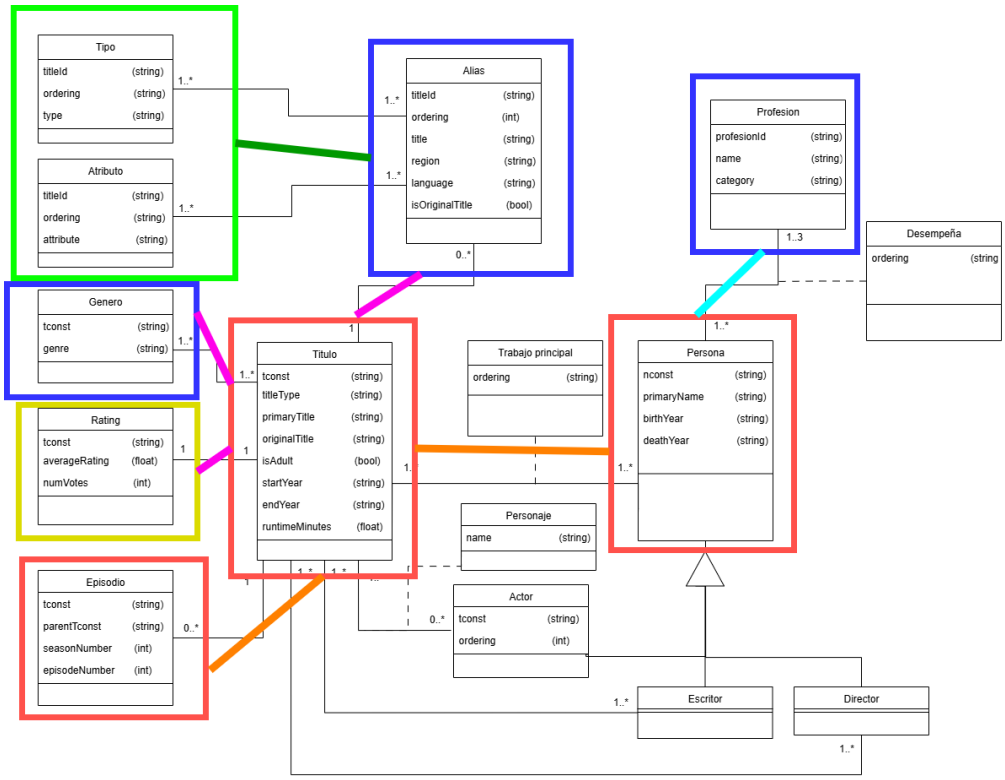


Figura 3: Esquema del caso de estudio con paradigmas de almacenamiento resultantes y relaciones

Fuente: Elaboración propia

Relacional

Grafo (nodo o arista)

Documental embebido

Clave-valor

Referencia por ID compartido

Referencia por tabla intermedia

Arista (grafo)

Embebido en entidad

4.4. Análisis cualitativo del diseño lógico

En esta etapa se busca que la transformación mantenga la consistencia semántica del modelo original. Para ello, se consideran las siguientes dimensiones y se justifica cómo se preservan en el modelo lógico multimodelo propuesto:

- **Cardinalidad:** Las relaciones 1:1 (por ejemplo, Título–Rating), 1:N (Título–Alias) y N:M (Persona–Título, Persona–Profesión, Título-Género) se modelan usando combinaciones de aristas en grafos, ids compartidos o documentos embebidos según convenga, asegurando que la multiplicidad semántica original no se pierda ni se simplifique erróneamente. También se conserva la información que aportan las clases de asociación del modelo original, como parte de la información contenida en la arista o colocándola como campo en una tabla intermedia. Es así, que se mantienen los detalles contextuales de cada participación y la expresividad de las relaciones que existen en el dominio

- **Grado:** El modelo incluye relaciones binarias (e.g., Persona–Profesión), respetando el número de entidades participantes. Relaciones de mayor complejidad (Persona-Título) se implementan como aristas enriquecidas con atributos, lo que permite mantener el significado de cada rol y evitar simplificar estructuras más de la cuenta, pudiendo perder contexto y afectar la consistencia y mantenibilidad de los datos.

- **Dependencia existencial:** Entidades como por ejemplo Episodios, Ratings o Género dependen de Título para su existencia. Esta dependencia se preserva modelando Episodios como nodos ”hijos”, Ratings como clave-valor vinculada por tconst, y género como una tabla que contiene tconst, evitando que existan registros huérfanos sin un título asociado. De esta forma las entidades débiles existen cuando existe una entidad fuerte que permita que su existencia tenga sentido

- **Temporalidad:** Aunque el modelo base no gestiona versiones históricas, es extensible para soportarlas. Por ejemplo, las aristas Persona–Profesión pueden incluir atributos temporales (como fecha de inicio y de término) si se requiere, y los motores seleccionados permiten almacenar propiedades temporales en nodos o aristas. En un caso extremo, donde se requiera gran velocidad y un registro temporal masivo, como un timestamp cada vez que se reproduce un título podría extenderse el modelo para incluir un paradigma columnar.

- **Unicidad:** Se garantiza a través de identificadores únicos como tconst (para Títulos) y nconst (para Personas), reutilizados los paradigmas como claves primarias, referencias cruzadas, tabla intermedia o identificadores de nodos. Esto asegura trazabilidad consistente entre modelos relacional, grafo, documental o clave-valor. Además los identificadores permiten conectar paradigmas distintos de almacenamiento sin complicar la lógica de implementación. La unicidad también se preserva por ejemplo, al modelar las subclases de Persona como atributos en la arista con título, debido a que se utiliza un nodo para cada persona sin duplicar información cuando tiene más de un rol (una

persona podría ser actor y escritor a la vez), favoreciendo la integridad de los datos y la mantenibilidad del sistema.

- **Herencia:** La jerarquía Persona–Actor/Director/Escritor se representa usando una única entidad Persona y diferenciando el rol en las aristas que la conectan con Título. Esto evita duplicidad de nodos y preserva la cobertura presente en el dominio de forma semánticamente clara.
- **Agregación / Composición:** En el modelo original no hay agregación ni composición de entidades, pero estructuras subordinadas como Tipo y Atributo se embeben dentro de Alias para reflejar su dependencia.
- **Categorización:** En el modelo conceptual no se utiliza categorización, ya que no existen roles compartidos entre clases de tipos distintos (por ejemplo, tanto una empresa como una persona pueden ser dueños de una propiedad, pero una empresa y una persona son entidades distintas). Las entidades como Tipo, Género o Atributo funcionan como catálogos o listas de referencia, no como interfaces abstractas. Por lo tanto, esta dimensión no tiene un impacto directo en la transformación lógica, pero se mantiene la claridad semántica en las relaciones de clasificación.

El modelo propuesto realiza un balance estratégico al incorporar distintos modelos de almacenamiento, asignando a cada componente del dominio el modelo que mejor se adapta a sus características y necesidades de uso. El modelo de grafos se utiliza para representar relaciones actorales, jerárquicas y colaborativas, optimizando la navegabilidad y consultas recursivas sobre redes de conexiones. El modelo documental embebido se aplica a entidades subordinadas que dependen fuertemente de otra (como Tipo y Atributo en Alias), favoreciendo la cohesión local y reduciendo la necesidad de uniones complejas al consultar por los detalles de un alias en específico. El modelo relacional se aplica para catálogos y listas de referencia como Género y Profesión, donde la estabilidad estructural y la reutilización son prioritarias. Finalmente, el modelo clave-valor se emplea para entidades como Ratings, donde se requieren accesos directos y rápidos sin involucrar relaciones adicionales. Este balance permite explotar las fortalezas individuales de cada paradigma, optimizando rendimiento, claridad semántica y adaptabilidad tecnológica.

Sin embargo, el modelo también presenta compromisos importantes. La combinación de múltiples paradigmas podría aumentar la complejidad operativa al ejecutar consultas o mantener integraciones entre documentos, grafos y estructuras relacionales. Modelar Actor, Director y Escritor como aristas simplifica la estructura y reduce la redundancia de datos favoreciendo la navegabilidad respecto a título, pero limita consultas específicas centradas en estos roles. Es clave destacar que este modelo no busca ser puramente relacional ni puramente de grafos, sino ofrecer una guía práctica y adaptable que balancea flexibilidad, expresividad y eficiencia, asumiendo conscientemente ciertos compromisos.

Un posible punto de ambigüedad en el modelo surge al usar documentos embebidos para entidades subordinadas como Tipo y Atributo. Se asume que su significado y contexto

quedan correctamente interpretados dentro del documento principal, lo que puede generar interpretaciones ambiguas si no se definen claramente las reglas de uso y actualización. Para lo anterior es tarea del desarrollador realizar un buen diccionario de datos para remediarlo.

Se destaca una vez más que cada elección de paradigma involucra trade-offs para el sistema final. En este caso, se decidió aumentar la complejidad operativa para aprovechar distintos tipos de almacenamiento, priorizando la preservación semántica, la expresividad y la adaptabilidad del modelo. Este enfoque permite asignar a cada componente del dominio el paradigma que mejor se ajusta a sus características, pero implica asumir compromisos explícitos en términos de mantenimiento, interoperabilidad y coordinación entre tecnologías heterogéneas, así como sacrificar parcialmente la facilidad y el rendimiento de algunas consultas. No obstante, estos compromisos son aceptados conscientemente para priorizar un modelo lógico multimodelo robusto, adaptable y semánticamente consistente.

Implementación del modelo en gestores de bases de datos

Luego de obtener el modelo lógico multimodelo en esta sección se procedió a la implementación en ArangoDB y SurrealDB. Cabe resaltar que este proceso se realizó en sistema operativo Windows 11, en un PC con 16 GB de ram y un procesador Intel Core i7 10th gen. Los archivos creados para el proceso de carga de datos en los dos motores se encuentra disponible en el siguiente enlace: https://github.com/Dani1i/Limpieza-y-carga-de-datos

4.4.1. Descarga y preparación de los datos

Para empezar, se descargaron los datos de IMDb Non-Commercial Datasets. En específico, los archivos: name.basics.tsv.gz, title.akas.tsv.gz, title.basics.tsv.gz, title.crew.tsv.gz, title.episode.tsv.gz, title.principals.tsv.gz y title.ratings.tsv.gz.

Luego de descomprimirlos, dado que el peso de los archivos en su conjunto supera los 8 GB, se decidió realizar un script de Python para procesar los datos y filtrar todo en base a los títulos que en el campo startYear tengan el año 2024. Luego del filtrado, el programa exporta en csv los siguientes archivos:

1. **titles.csv:** Entidad Título
2. **episodes.csv:** Entidad Episodio
3. **ratings.csv:** Entidad Rating
4. **genres.csv:** Entidad Género
5. **aliases.csv:** Entidad Alias, contiene dentro las entidades Tipo y Atributo dado que antes se determinó que serían documentos embebidos.

6. **professions.csv:** Entidad Profession
7. **people.csv:** Entidad Persona
8. **episode_title_edges.csv:** Arista entre Episodio y Título
9. **main_job_edges.csv:** Arista entre Persona y Título, contiene el atributo ordering para describir los trabajos principales por los cuales una persona es reconocida.
10. **role_edges.csv:** Arista entre Persona y Título, contiene la relación entre ambas entidades: ACTED_IN, DIRECTED o WROTE dependiendo del rol de la persona en la obra. Además para el caso de los actores contiene el nombre del personaje.
11. **person_profession.csv:** Tabla intermedia que relaciona Profession y persona, almacena ambos ID junto con el campo ordering que indica la relevancia de la profesión en cada persona.

Estos .csv contienen la información a cargar en las bases de datos. El código completo se encuentra disponible en GitHub.

Se tomó la decisión de modularizar los datos en archivos .csv individuales, existiendo un archivo para cada entidad y para las relaciones. De esta manera, se espera que el proceso de inserción en cada uno de los gestores sea más fácil y manejable.

4.4.2. ArangoDB

El proceso de carga de datos en ArangoDB requirió una serie de pasos. A continuación, se describen las etapas principales:

1. **Preparación del entorno:** Se levantó una instancia de ArangoDB mediante Docker utilizando la imagen oficial: arangodb/arangodb:3.11.8, exponiendo el puerto 8529 y montando la carpeta local que contenía los archivos CSV procesados. El comando utilizado fue:

   ```
   docker run -p 8529:8529 -e ARANGO_NO_AUTH=1\
   -v C:/Users/pacma/Downloads/Datas:/import\
   --name arangodb_instance -d arangodb/arangodb:3.11.8
   ```

   Este comando crea la base de datos en modo sin autorización, un modo solo recomendable para pruebas debido a que es inseguro. La base de datos de trabajo, denominada `imdb`, se creó manualmente desde la interfaz web (WebUI).

2. **Normalización de relaciones:** ArangoDB requiere un formato específico de columnas para establecer relaciones entre aristas y para sus IDs. Por ello, se creó un script en

Python que generó las columnas _from y _to para el sentido de las relaciones y también el campo _id en las entidades. Sin ello no es posible cruzar registros. Este script procesó los archivos originales asegurando que los id fueran definidos en el formato necesario y que las relaciones quedaran correctamente asociadas a los nodos de origen y destino. Además, se generaron claves compuestas en colecciones multivaluadas (como en genres.csv) para evitar conflictos de claves duplicadas durante la importación.

3. **Carga automatizada:** Se desarrolló un *script* en formato .bat para Windows que utilizó comandos docker exec para invocar arangoimport dentro del contenedor. Cada colección fue cargada individualmente, especificando:
   - Tipo de colección (document o edge).
   - Archivo CSV limpio y formateado a importar.
   - Base de datos destino (imdb).
   - Credenciales de acceso o modo sin autenticación.

En ArangoDB, los paradigmas coexisten de forma flexible. La diferenciación no requiere configuraciones explícitas al momento de cargar los datos, sino que depende del modelo lógico adoptado y de cómo se estructuran y consultan las colecciones. Esto permite integrar distintos enfoques de almacenamiento en una única base multimodelo.

El script automatizó la ejecución secuencial de importaciones, asegurando la creación automática de las colecciones y optimizando el proceso para minimizar intervención manual.

Este proceso no solo valida la aplicabilidad técnica del modelo lógico propuesto, sino que además garantiza que los principios de consistencia, trazabilidad y cohesión semántica se mantengan a lo largo de la implementación.

4.4.3. SurrealDB

El proceso de carga de datos en SurrealDB involucró una serie de pasos, comenzando por instalar la aplicación de escritorio Surrealist y, adicionalmente, SurrealDB por consola usando el instalador oficial para Windows:

```
iwr https://windows.surrealdb.com -useb | iex
```

Al tratarse de un gestor de bases de datos muy reciente, apoyarse en la documentación disponible fue crucial.

Luego, por consola se navegó hasta la carpeta de instalación indicada en el output del instalador y se inicializó el motor local con el siguiente comando:

```
./surreal start --user root --pass root rocksdb:imdb.db
```

Esta línea inicializa un motor SurrealDB local utilizando RocksDB como backend en disco, y habilita el acceso del usuario root. Posteriormente, se definieron manualmente el namespace demo y la base de datos IMDB dentro de Surrealist para habilitar el esquema de trabajo.

Se evaluaron distintas alternativas para la carga de datos:

- Primero, levantar un contenedor Docker para alojar la base de datos; sin embargo, pese al filtrado previo de los datos, la carga excedía la memoria RAM disponible y provocaba colapsos en el contenedor.
- Luego, se consideró utilizar la versión cloud, pero el plan gratuito presentaba velocidades de carga demasiado bajas y un límite de almacenamiento que probablemente no alcanzaría para contener todos los datos.
- Finalmente, se optó por crear la base de datos en disco local usando RocksDB, como se describió previamente.

Se desarrolló un script en Python que carga los datos en lotes de 500 registros, enviando comandos UPDATE y RELATE al motor SurrealDB mediante su cliente Python oficial. Dado que no existe una herramienta nativa para carga masiva ni manejo automático de errores, se incorporaron mecanismos simples de monitoreo en consola para verificar el progreso y detectar posibles fallos durante la inserción. Cabe resaltar que Surreal requiere que los ID de una tabla incluyan como prefijo el nombre de la tabla (por diseño del motor). Así, aunque el identificador del registro original sea el mismo, no es posible mapear automáticamente entre dos tablas que compartan IDs, si no se establece explícitamente una relación entre ellas en SurrealDB. Por ende, además del script principal de carga, se desarrollaron scripts adicionales que permiten generar este tipo de relaciones.

En SurrealDB, los paradigmas coexisten de forma nativa dentro del mismo motor. La diferenciación entre ellos no requiere configuraciones explícitas al momento de cargar los datos, sino que depende del modelo lógico adoptado y del uso posterior en consultas y relaciones. Esta flexibilidad permite integrar distintos enfoques de almacenamiento sobre una única base, facilitando la implementación multimodelo.

Este proceso permitió consolidar un modelo multimodelo funcional en SurrealDB, integrando datos documentales, relaciones de grafo y estructuras clave-valor, preservando las dimensiones semánticas del modelo original.

Cabe mencionar que, al no existir herramientas oficiales de ingestión masiva ni optimizaciones específicas (como índices previos o carga paralela), el tiempo total de carga fue mayor al esperado, aunque aceptable para el tamaño del dataset.

Este proceso permitió validar empíricamente las capacidades y limitaciones de SurrealDB como motor multimodelo emergente, aportando evidencia práctica sobre su aplicabilidad,

flexibilidad y desafíos técnicos actuales.

4.4.4. Testeo de las bases de datos

Luego de insertar los datos, se puso a prueba el modelo implementado mediante 4 consultas que involucran datos a través de distintos paradigmas de almacenamiento:

Consulta 1: Traer los títulos en los que actuó Robert Downey Jr., junto con su rating promedio de existir (grafo, clave-valor). El resultado para ArangoDB se encuentra en la figura 4 y para SurrealDB en la figura 5.

ArangoDB:

```
FOR person IN people
  FILTER person.primaryName == "Robert Downey Jr."
  FOR edge IN role_edges
    FILTER edge._from == person._id AND edge.relation == "ACTED_IN"
    FOR title IN titles
      FILTER title._id == edge._to
      LET rating = FIRST(
        FOR r IN ratings
          FILTER r._key == title._key
          RETURN r
      )
      RETURN {
        title: title.primaryTitle,
        averageRating: rating != null ? rating.averageRating : "No rating"
      }

```

Query 18 elements 368.640 ms JSON Table

title	averageRating
The Sympathizer	6.8
The Sympathizer	6.8
The Sympathizer	6.8
Death Wish	7.9
Good Little Asian	7.4
Good Little Asian	7.4
Endings Are Hard, Aren't They?	7.6
Endings Are Hard, Aren't They?	7.6
Endings Are Hard, Aren't They?	7.6
Love It or Leave It	7.6

Figura 4: Resultado en ArangoDB para Consulta 1

SurrealDB:

```
SELECT ->acted_in->title.primaryTitle AS title,
       ->acted_in->title->calificacion
       ->rating.averageRating AS averageRating
FROM person WHERE primaryName = 'Robert Downey Jr.'
AND ->acted_in->title.primaryTitle != NONE;
```

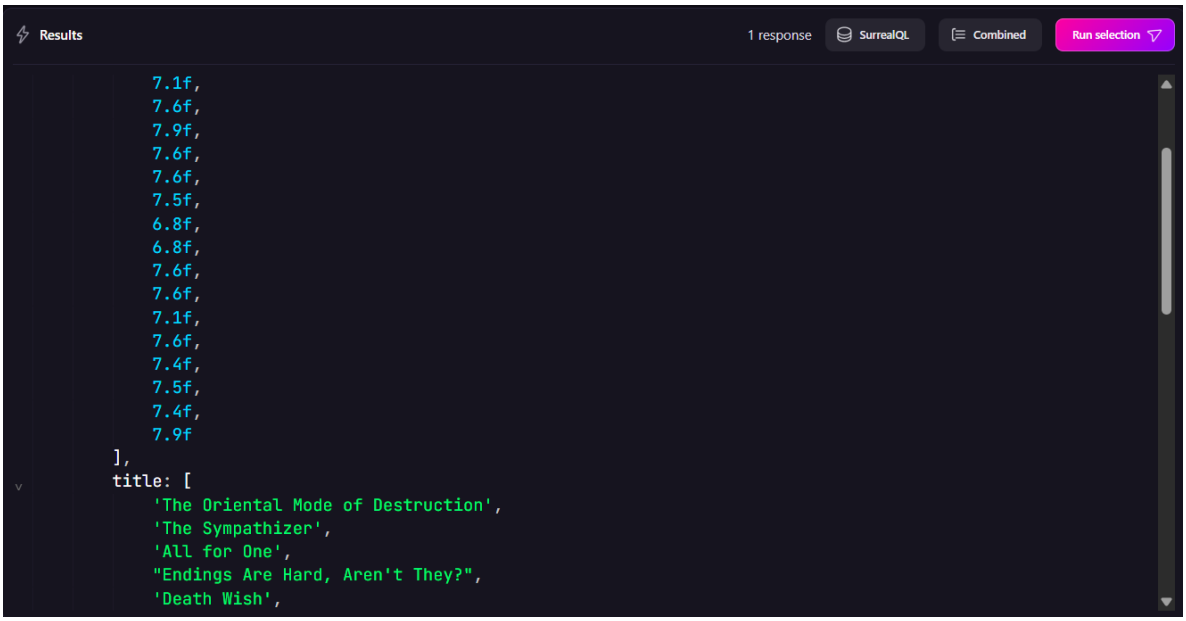


Figura 5: Resultado en SurrealDB para Consulta 1

Consulta 2: Para un título específico (Shook), traer su rating, género y personas que actuaron (clave-valor, relacional, grafo). El resultado para ArangoDB se encuentra en la figura 6 y para SurrealDB en la figura 7.

ArangoDB:

```
FOR title IN titles
  FILTER title.primaryTitle == "Shook"

  LET rating = FIRST(
    FOR r IN ratings
      FILTER r._key == title._key
      RETURN r
  )

  LET genres = (
    FOR g IN genres
      FILTER g.tconst == title._key
      RETURN g.genre
  )

  LET actors = (
    FOR edge IN role_edges
      FILTER edge._to == title._id AND edge.relation == "ACTED_IN"
      FOR person IN people
        FILTER person._id == edge._from
        RETURN person.primaryName
  )

  RETURN {
    title: title.primaryTitle,
    averageRating: rating != null ? rating.averageRating : "No rating",
    genres: genres,
    actors: actors
  }
```

Query | 2 elements | 795.516 ms | JSON | Table

title	averageRating	genres	actors
Shook	9	["Drama"]	["Nina Kiri","Amy Forsyth","Darrin Baker","Bernard White","Saamer Usmani","Pamela Sinha","Nayo Sasaki-Picou","Ho Chow","Sharjil Rasool","Scott Edgecombe"]
Shook	No rating	["Short"]	["Cheryl Canty","Phoebe Grant","Ben Irving"]

Figura 6: Resultado en ArangoDB para Consulta 2

SurrealDB:

```
SELECT primaryTitle,
       ->calificacion->rating.averageRating AS averageRating,
       ->tiene_genero->genre.genre AS genres,
       <-acted_in<-person.primaryName AS actors
FROM title
WHERE primaryTitle = "Shook";
```

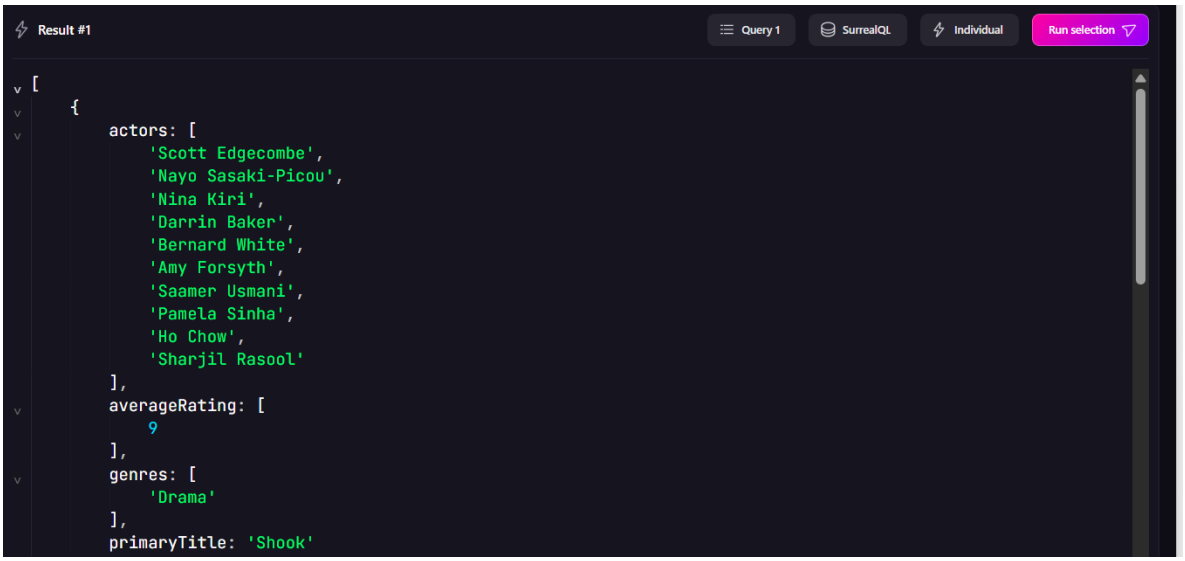


Figura 7: Resultado en SurrealDB para Consulta 2

Consulta 3: Devolver los alias y números de los episodios disponibles para el título de nombre ”Phoenix”(relacional,grafo) (Se elige por precaución limitar a 1 resultado la query porque el dataset de por si tiene algunos registros duplicados). El resultado para ArangoDB se encuentra en la figura 8 y para SurrealDB en la figura 9.

ArangoDB

```
FOR title IN titles
  FILTER title.primaryTitle == "Phoenix"
  LIMIT 1

  LET aliases = (
    FOR a IN aliases
      FILTER a.titleId == title._key
      RETURN a.title
  )

  LET episodes = (
    FOR e IN episodes
      FILTER e.parentTconst == title._key
      RETURN {
        tconst: e._key,
        seasonNumber: e.seasonNumber,
        episodeNumber: e.episodeNumber
      }
  )

  RETURN {
    title: title.primaryTitle,
    aliases: aliases,
    episodes: episodes
  }
```

Query 1 elements 729.889 ms JSON Table

title	aliases	episodes
Phoenix	["Phoenix","Phoenix","Phoenix","Phoenix","Phoenix","Phoenix","Phoenix"]	[{"tconst":"tt10112876","seasonNumber":1,"episodeNumber":1}, {"tconst":"tt10121448","seasonNumber":1,"episodeNumber":2}, {"tconst":"tt10126710","seasonNumber":1,"episodeNumber":3}, {"tconst":"tt10307012","seasonNumber":1,"episodeNumber":4}, {"tconst":"tt10307076","seasonNumber":1,"episodeNumber":5}, {"tconst":"tt10307136","seasonNumber":1,"episodeNumber":6}, {"tconst":"tt14980602","seasonNumber":1,"episodeNumber":7}]

Figura 8: Resultado en ArangoDB para Consulta 3

Surreal

```
SELECT id AS titleId,
       primaryTitle,
       ->tiene_alias->alias.title AS aliasTitles,
       <-part_of<-episode.episodeNumber AS episodes
FROM title
WHERE primaryTitle = 'Phoenix'
LIMIT 1;
```

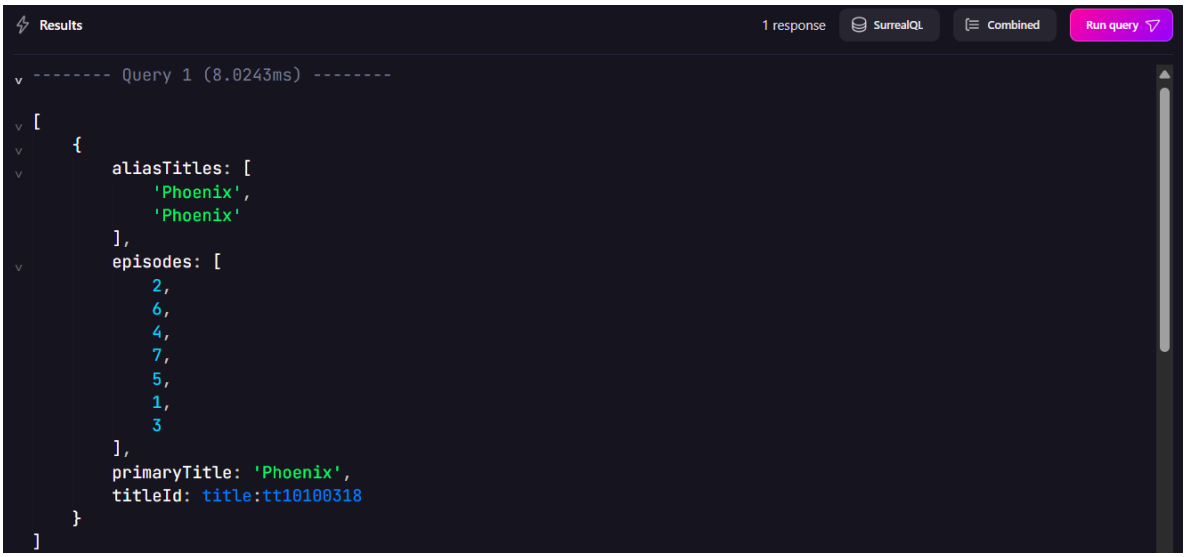


Figura 9: Resultado en SurrealDB para Consulta 3

Consulta 4: Consultar 5 títulos con más de 100 votos (numVotes) y traer las aristas de quienes escribieron el guion (clave-valor, grafo). El resultado para ArangoDB se encuentra en la figura 10 y para SurrealDB en la figura 11.

ArangoDB

```
FOR title IN titles
  LET rating = FIRST(
    FOR r IN ratings
      FILTER r._key == title._key AND TO_NUMBER(r.numVotes) > 100
      RETURN r
  )
  FILTER rating != null
  LIMIT 5

  LET writers = (
    FOR edge IN role_edges
      FILTER edge._to == title._id AND edge.relation == "WROTE"
      FOR person IN people
        FILTER person._id == edge._from
        RETURN person.primaryName
  )

  RETURN {
    title: title.primaryTitle,
    numVotes: rating.numVotes,
    writers: writers
  }
```

Query | 5 elements | 25.064 ms | JSON | Table

title	numVotes	writers
Jait Re Jait	188	["Satish Alekar","Gopal Nilkanth Dandekar","Anil Joglekar"]
Universal Groove	173	["Sandeep Panesar"]
The Killer's Game	18538	["Rand Ravich","James Coyne","Jay Bonansinga"]
Kevin Haddock	613	["Jemaine Clement","Iain Morris","Taika Waititi","Terry Gilliam","Michael Palin","Melanie Bracewell"]
Maybe It's True What They Say About Us	119	["Camilo Becerra","Sofía Gómez"]

Download | CSV | Copy To Editor

Figura 10: Resultado en ArangoDB para Consulta 4

Surreal

```
SELECT
    id AS titleId,
    primaryTitle,
    (<-wrote<-person).primaryName AS writers
FROM
    title
WHERE
    ->calificacion->rating.numVotes > 1000
    AND !!(<-wrote<-person)
LIMIT 5;
```

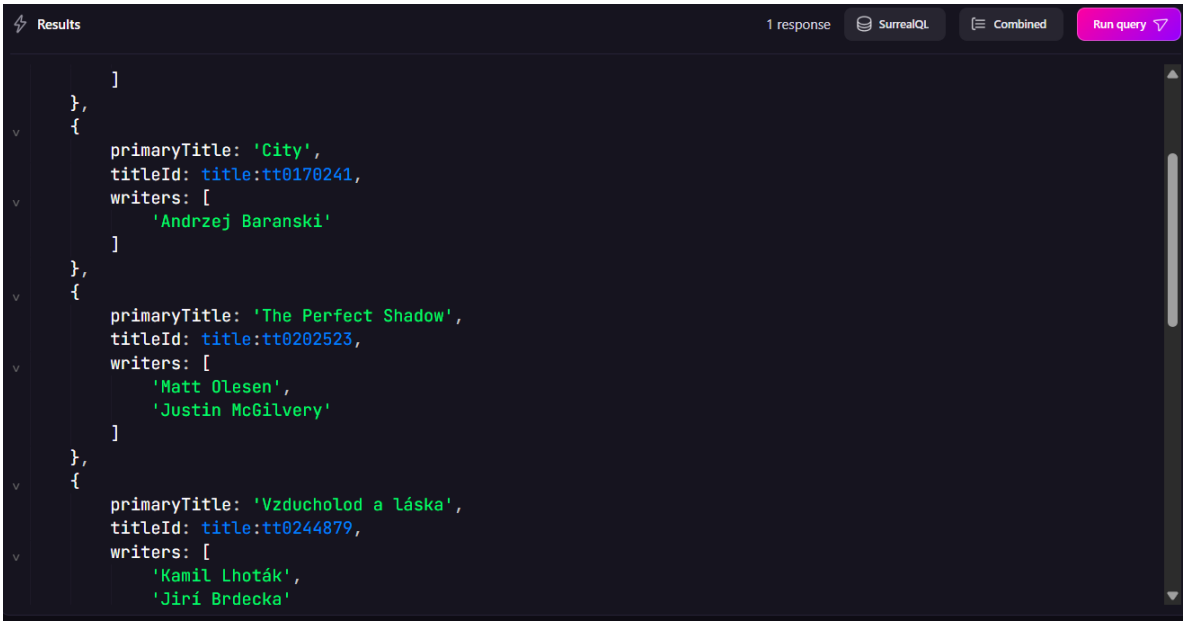


Figura 11: Resultado en SurrealDB para Consulta 4

4.5. Análisis comparativo entre ArangoDB y SurrealDB

La implementación en ArangoDB permitió explotar nativamente modelos de documentos, grafos y clave-valor. Sin embargo, se evidenció la necesidad de preparar explícitamente las colecciones como `edge collections` para habilitar funcionalidades de grafo, así como de ajustar claves e identificadores para mantener la coherencia multimodelo.

Por su parte, SurrealDB, aunque emergente, mostró una notable flexibilidad al integrar múltiples paradigmas bajo un único lenguaje declarativo. No obstante, presentó desafíos relacionados con la carga masiva de datos, la ausencia de herramientas de ingestión optimizada y la obligatoriedad de prefijar identificadores por tabla, lo que requirió desarrollar scripts adicionales.

Es importante destacar que, si bien la carga de datos resultó demandante en ambos motores, parte de las dificultades encontradas pueden atribuirse a la curva de aprendizaje asociada al manejo de cada herramienta. SurrealDB cuenta con una comunidad aún reducida en comparación con ArangoDB, por lo que fue necesario apoyarse principalmente en la documentación oficial y en los pocos recursos disponibles en línea. En términos de usabilidad, ArangoDB resultó más sencillo para la carga inicial de datos, pese a contar con un lenguaje de consultas más verboso, mientras que SurrealDB, si bien presentó problemas de desempeño, incorpora herramientas modernas como Sidekick, una interfaz con asistencia mediante inteligencia artificial para apoyar en el troubleshooting, funcionalidad ausente en ArangoDB.

En cuanto a rendimiento, ArangoDB demostró ser significativamente más robusto en la fase de carga de datos, completando el proceso en aproximadamente la mitad del tiempo requerido por SurrealDB (en torno a 30 minutos versus una hora). Esta diferencia también se reflejó en el control de errores, donde la madurez de ArangoDB, con más de una década de desarrollo, se evidenció frente a SurrealDB, que cuenta con apenas tres años desde su lanzamiento.

Sin embargo, SurrealDB destaca por su versatilidad de instalación y despliegue, ofreciendo opciones locales, en la nube, mediante contenedores Docker u otras alternativas, así como por su documentación amigable y orientada a ejemplos prácticos, incluyendo tutoriales diseñados específicamente para estudiantes universitarios. A nivel de ejecución de consultas, ArangoDB mostró un desempeño superior, resolviendo consultas aproximadamente un tercio más rápido que SurrealDB en los escenarios probados.

En resumen, SurrealDB exhibe un gran potencial como motor multimodelo emergente, incorporando funcionalidades modernas ausentes en ArangoDB, aunque aún presenta carencias atribuibles a su corta trayectoria. ArangoDB, por su parte, se posiciona como una solución más robusta y madura, con capacidades optimizadas para la gestión de grandes volúmenes de datos y un ecosistema consolidado, lo que lo hace especialmente adecuado para escenarios de producción exigentes.

Conclusiones

El modelo lógico obtenido logró equilibrar las ventajas de cada paradigma de almacenamiento: grafos para relaciones entre personas y títulos y jerárquicas, clave-valor para accesos directos, y relacional para estructuras rigidas. Tal como se ha mencionado en otras secciones, se priorizaron consultas orientadas a la navegabilidad entre títulos, episodios y personas, junto con sus trabajos principales y roles en las distintas producciones. La elección de estas prioridades responde a una definición consciente de los objetivos del diseño: en este caso, privilegiar la capacidad exploratoria sobre el dominio audiovisual por encima de consultas menos centrales, como las relacionadas a alias alternativos o atributos secundarios como profesiones.

La metodología actual admite segundas versiones del modelo lógico aplicado a un mismo caso de estudio, lo cual refuerza su carácter flexible y adaptable. En efecto, dependiendo de los objetivos específicos del diseñador de bases de datos, podrían plantearse elecciones alternativas de paradigmas, priorizando, por ejemplo, la velocidad de consultas analíticas, la redundancia controlada para mejorar el rendimiento, o incluso la simplificación estructural para favorecer mantenibilidad a largo plazo. En esta memoria, se cuidó explícitamente reducir la redundancia innecesaria y mantener un equilibrio razonable entre eficiencia y claridad semántica, sin sacrificar la integridad del dominio representado.

Un aspecto destacado es que la metodología permite conservar las dimensiones semánticas porque evita operaciones de transformación que impliquen pérdida de información: toda entidad conceptual relevante es modelada de forma explícita, o bien parte de su información se integra a estructuras relacionadas, asegurando que el significado original no se diluya ni desaparezca. En otras palabras, la metodología no sólo ofrece criterios para asignar paradigmas de almacenamiento, sino también directrices para preservar la riqueza semántica del dominio, manteniendo trazabilidad y coherencia entre los distintos niveles de representación.

Cabe mencionar que, al aplicar la metodología en gestores concretos, surgieron aprendizajes adicionales: por ejemplo, la importancia de ajustar estrategias según las limitaciones operativas de cada gestor (como la obligatoriedad de claves prefijadas en SurrealDB o la definición explícita de colecciones de aristas en ArangoDB), y la necesidad de complementar las heurísticas metodológicas con criterios prácticos que optimicen rendimiento, carga y consultas en entornos heterogéneos. Esto refuerza la idea de que el diseño lógico multimodelo no es un proceso mecánico, sino un ejercicio de ingeniería que requiere balancear teoría, objetivos de negocio, y capacidades tecnológicas reales.

Cabe destacar que la metodología presentada en esta memoria constituye una propuesta inicial, no una solución definitiva ni universal. Su objetivo es servir como guía flexible para orientar el diseño lógico en entornos multimodelo, aportando criterios y heurísticas fundamentadas que pueden ser adaptadas y evolucionadas según las necesidades específicas de cada proyecto y los avances futuros en la disciplina.

Validación de los objetivos

La propuesta metodológica desarrollada en esta memoria fue diseñada para responder a los objetivos generales y específicos definidos al inicio del trabajo. A continuación, se discute en detalle cómo se lograron validar dichos objetivos, comparando los procesos realizados en ArangoDB y SurrealDB, y reflexionando sobre el modelo lógico obtenido.

Objetivo general: desarrollar y validar una metodología replicable para el diseño lógico de bases de datos multimodelo, basada en heurísticas de diseño que permitan decidir cómo representar cada parte de los datos según su estructura y su uso.

Este objetivo se cumplió mediante la construcción de una metodología que combina heurísticas provenientes del estado del arte con un análisis propio basado en características estructurales y semánticas del dominio. La metodología fue aplicada a un caso de estudio real (IMDb dataset), implementada en dos motores multimodelo distintos, y validada empíricamente.

Objetivos específicos:

- Analizar las características técnicas de diversos gestores de bases de datos multimodelo, evaluando los tipos de modelos de datos que soportan, su integración entre paradigmas y su adecuación a distintos escenarios de uso.

  Se analizaron en detalle las capacidades, modelos soportados y limitaciones de Oracle, Redis, Postgres, ArangoDB y SurrealDB. De los cuales se seleccionaron ArangoDB y SurrealDB para la implementación, representando respectivamente un motor maduro y uno emergente. Este objetivo se abordó específicamente en la sección 2.3.

- Identificar y sistematizar un conjunto de al menos cinco heurísticas de diseño lógico, extraídas del estado del arte y del análisis propio, que orienten la transformación de modelos conceptuales a modelos lógicos en bases de datos multimodelo, para apoyar la toma de decisiones fundamentadas durante el diseño lógico, asegurando coherencia con el dominio y facilitando su aplicación en contextos similares.

  En total se identificaron 20 heurísticas o criterios de diseño en la literatura y por medio del análisis propio para orientar decisiones fundamentadas de modelado de entidades y relaciones en contextos multimodelo. Este objetivo se abordó específicamente en los capítulos 2.4, 3.1 y 3.2.

- Diseñar un proceso metodológico de transformación desde un modelo conceptual (UML) hacia un modelo lógico multimodelo, utilizando las heurísticas identificadas como guía para asignar un paradigma de almacenamiento a cada componente del dominio, para ofrecer un proceso paso a paso que permita aplicar las heurísticas de manera estructurada, manteniendo la consistencia del dominio.

A partir de las heurísticas identificadas se definió un proceso paso a paso que permite transformar modelos conceptuales UML en modelos lógicos multimodelo, preservando dimensiones semánticas relevantes y por medio de decisiones fundamentadas. Esta metodología asegura que al menos un paradigma sea asignado a cada elemento del modelo conceptual. Este objetivo se abordó específicamente en los capítulos 3.1 y 3.2.

- Implementar el modelo lógico obtenido en un caso de estudio real, empleando dos gestores de bases de datos multimodelo. Para validar que la metodología puede adaptarse a diferentes motores con sus respectivas particularidades, preservando la consistencia semántica del dominio.

  Se ejecutó la metodología sobre el dataset IMDb, cargando y consultando datos en ArangoDB y SurrealDB, demostrando su aplicabilidad práctica y su efectividad aplicada al dataset IMDB. Este objetivo se abordó específicamente en el capítulo 4.6

Trabajo futuro

A partir de los resultados obtenidos, se identifican múltiples oportunidades para extender y mejorar la propuesta metodológica desarrollada en esta memoria. Uno de los aspectos más relevantes es la posibilidad de enriquecer la metodología con pasos adicionales que permitan formalizar aún más el proceso de toma de decisiones. Por ejemplo, se podría construir un árbol de decisiones o diagrama de flujo que guíe al diseñador de bases de datos en secuencias más deterministas, priorizando criterios según el dominio, el volumen de datos, las restricciones de rendimiento, o las capacidades específicas de los motores seleccionados. Esto permitiría reducir la dependencia de la experiencia personal y facilitar la replicabilidad del proceso en otros contextos.

Además, sería valioso incorporar métricas cuantitativas para evaluar los modelos generados, no sólo en términos de preservación semántica, sino también en aspectos como rendimiento de consultas, escalabilidad y mantenibilidad, para alimentar de forma iterativa el proceso metodológico.

Otra línea clave es refinar la metodología mediante su exposición a dominios más complejos, que presenten características adicionales como datos heterogéneos en tiempo real, alta variabilidad estructural o integración de fuentes externas. Estos escenarios permitirían ajustar y robustecer las heurísticas actuales, validando su aplicabilidad en contextos de mayor exigencia.

En cuanto al ecosistema tecnológico, los resultados evidenciaron una limitación significativa: actualmente son escasos los gestores de bases de datos multimodelo que soportan de manera robusta tres o más paradigmas de almacenamiento de forma integrada. Un trabajo futuro interesante sería explorar e incluso prototipar herramientas que amplíen esta frontera, priorizando motores que no sólo combinen paradigmas como documental, grafo y clave-

valor, sino que además lo hagan mediante lenguajes de consulta intuitivos, preferentemente SQL-like, reduciendo la curva de aprendizaje para nuevos usuarios.

Otra línea clave es la mejora de las herramientas de carga e ingestión de datos: los motores evaluados carecen en su mayoría de pipelines nativos robustos para importar datos heterogéneos (CSV, TSV, JSON, entre otros) sin depender de programación adicional. Desarrollar o integrar módulos de carga de datos intuitivos y visuales sería un aporte significativo para democratizar el uso de bases de datos multimodelo, facilitando su adopción por parte de equipos no especializados en desarrollo.

Finalmente, una extensión natural sería aplicar la metodología propuesta en otros dominios (como bioinformática, redes sociales o IoT) para validar su generalidad y ajustar sus heurísticas a diferentes características de datos y contextos de uso.

Aporte de la carrera

Para el desarrollo de esta memoria se requirieron diversas competencias adquiridas a lo largo de la carrera, tanto en aspectos técnicos como metodológicos. Estas habilidades permitieron enfrentar adecuadamente los desafíos del diseño lógico multimodelo, su implementación práctica y la reflexión crítica en torno a sus implicancias. Entre las asignaturas más relevantes, se destacan:

1. **Bases de Datos:** Por su aporte en la comprensión del modelado lógico relacional y la sintaxis SQL, fundamentales para establecer comparaciones entre paradigmas y comprender otros lenguajes de consulta empleados en los motores multimodelo.
2. **Bases de datos avanzadas:** Por entregar conocimientos sobre bases de datos NoSQL, incluyendo su modelado, comportamientos, casos de uso y características técnicas. Esto fue clave para proponer y aplicar criterios de diseño lógico multimodelo.
3. **Lenguajes de programación:** Por el dominio de Python, utilizado en la creación de *scripts* para transformación y carga de datos, así como para la automatización de procesos y resolución de errores.
4. **Sistemas distribuidos:** Por proporcionar las bases para el uso de contenedores Docker, lo que facilitó el despliegue de ArangoDB en entornos controlados de prueba y para pruebas de carga de datos con SurrealDB.

Anexo

title.basics.tsv

Campo	Descripción
`tconst`	Identificador único del título (ej. tt1234567).
`titleType`	Tipo de título: movie, short, tvSeries, tvEpisode, etc.
`primaryTitle`	Título principal mostrado al público.
`originalTitle`	Título original (puede diferir del principal).
`isAdult`	1 si es contenido para adultos, 0 si no.
`startYear`	Año de estreno.
`endYear`	Año de término (solo para series).
`runtimeMinutes`	Duración en minutos.
`genres`	Lista separada por comas de géneros (ej. Drama, Comedy).

name.basics.tsv

Campo	Descripción
`nconst`	Identificador único de persona (ej. nm0000123).
`primaryName`	Nombre completo de la persona.
`birthYear`	Año de nacimiento (puede estar vacío).
`deathYear`	Año de fallecimiento (puede estar vacío).
`primaryProfession`	Lista de profesiones principales (ej. actor, director).
`knownForTitles`	Lista de títulos destacados asociados a la persona.

title.principals.tsv

Campo	Descripción
`tconst`	ID del título (película, serie, etc.).
`ordering`	Orden de aparición en los créditos.
`nconst`	ID de la persona asociada.
`category`	Rol general: actor, director, writer, etc.
`job`	Descripción más específica del trabajo (puede estar vacío).
`characters`	Lista de personajes interpretados (solo para actores).

title.crew.tsv

Campo	Descripción
`tconst`	ID del título.
`directors`	Lista de IDs de los directores (nconst), separados por comas.
`writers`	Lista de IDs de los guionistas (nconst), separados por comas.

title.ratings.tsv

Campo	Descripción
`tconst`	ID del título.
`averageRating`	Calificación promedio del título (decimal entre 0 y 10).
`numVotes`	Número total de votos recibidos.

title.akas.tsv

Campo	Descripción
`titleId`	ID del título principal al que corresponde esta variante.
`ordering`	Número que indica el orden para mostrar los títulos alternativos.
`title`	Título alternativo.
`region`	Código de país o región ISO.
`language`	Idioma del título alternativo.
`types`	Tipo del título: original, alternative, DVD, etc.
`attributes`	Información adicional: subtitulado, censurado, etc.
`isOriginalTitle`	1 si es el título original en ese idioma/región, 0 si no.

Referencias bibliográficas

[dbd,] Database of Databases — PostgreSQL — dbdb.io. https://dbdb.io/db/postgresql. [Accessed 06-05-2025].

[neo, 2015] (2015). Why Graph Databases? — neo4j.com. https://neo4j.com/why-graph-databases/. [Accessed 06-05-2025].

[mon, 2021] (2021). Document Database - NoSQL — mongodb.com. https://www.mongodb.com/es/resources/basics/databases/document-databases. [Accessed 06-05-2025].

[haz, 2024] (2024). Key-Value Store — hazelcast.com. https://hazelcast.com/foundations/data-and-middleware-technologies/key-value-store/. [Accessed 06-05-2025].

[Ammar, 2022] Ammar (2022). What on earth are graph databases? — engineering.99x.io. https://engineering.99x.io/what-on-earth-are-graph-databases-b139016dd0fa. [Accessed 28-07-2025].

[Anuyah *et al.*, 2024] Anuyah, S., Bolade, V., y Agbaakin, O. (2024). Understanding graph databases: A comprehensive tutorial and survey.

[ArangoDB, 2025] ArangoDB (2025). AQL Documentation — docs.arangodb.com. https://docs.arangodb.com/3.11/aql/. [Accessed 21-04-2025].

[Bathla *et al.*, 2018] Bathla, G., Rani, R., y Aggarwal, H. (2018). Comparative study of nosql databases for big data storage. *International Journal of Engineering & Technology*, 7(2.6):83–87.

[Berg *et al.*, 2012] Berg, K., Seymour, D. T., y Goel, R. (2012). History of databases. *International Journal of Management Information Systems (IJMIS)*, 17:29.

[Bimonte *et al.*, 2023] Bimonte, S., Gallinucci, E., Marcel, P., y Rizzi, S. (2023). Logical design of multi-model data warehouses. *Knowledge and Information Systems*, 65(3):1067–1103.

[Business, 2025] Business, C. (2025). A Brief History of ArangoDB — canvasbusinessmodel.com. https://canvasbusinessmodel.com/blogs/brief-history/arangodb-brief-history?srsltid=AfmBOorIZGRqNgU8OIke20nz4SUaCl6sUYtU_LbKvEKnJ-l5oR-E71ph. [Accessed 21-04-2025].

[Candel *et al.*, 2022] Candel, C. J. F., Ruiz, D. S., y García-Molina, J. J. (2022). A unified metamodel for nosql and relational databases. *Information Systems*, 104:101898.

[Chillón *et al.*, 2021] Chillón, A. H., Hoyos, J. R., Garcia-Molina, J., y Ruiz, D. S. (2021). Discovering entity inheritance relationships in document stores. *Knowledge-Based Systems*, 230:107394.

[Foote, 2021] Foote, K. D. (2021). A Brief History of Database Management - DATAVERSITY — dataversity.net. https://www.dataversity.net/brief-history-database-management/. [Accessed 28-07-2025].

[Goodfellow, 2025] Goodfellow, J. (2025). Multi-Model Databases: A Modern Approach to Data Management — progress.com. https://www.progress.com/blogs/multi-model-databases-a-modern-approach-to-data-management. [Accessed 28-07-2025].

[Group, 2024] Group, C. M. D. (2024). Database of Databases — OrientDB — dbdb.io. https://dbdb.io/db/orientdb. [Accessed 21-04-2025].

[Groves, 2025] Groves, M. (2025). Columnar database use cases and examples.

[Hecht y Jablonski, 2011] Hecht, R. y Jablonski, S. (2011). Nosql evaluation: A use case oriented survey. En *2011 International Conference on Cloud and Service Computing*, pp. 336–341.

[IBM, 2021] IBM (2021). ¿Qué es PostgreSQL? | IBM — ibm.com. https://www.ibm.com/mx-es/topics/postgresql. [Accessed 06-05-2025].

[IMDB, 2020] IMDB (2020). Non-commercial-datasets. https://developer.imdb.com/non-commercial-datasets/. [Accessed 28-07-2025].

[Jadon, 2024] Jadon, A. (2024). Understanding Aggregate Data Models in NoSQL Simplified 101 - Learn | Hevo — hevodata.com. https://hevodata.com/learn/aggregate-data-models-in-nosql/. [Accessed 28-07-2025].

[Johnston, 2022] Johnston, W. (2022). NoSQL Data Modeling - Redis — redis.io. https://redis.io/blog/nosql-data-modeling/. [Accessed 07-05-2025].

[Jordan, 2023] Jordan, M. (2023). What is NoSQL Data Modeling? Definition & FAQs | ScyllaDB — scylladb.com. https://www.scylladb.com/glossary/nosql-data-modeling/. [Accessed 07-05-2025].

[Kelly, 2022] Kelly, D. (2022). A brief history of databases: From relational, to NoSQL, to distributed SQL — cockroachlabs.com. https://www.cockroachlabs.com/blog/history-of-databases-distributed-sql/. [Accessed 28-07-2025].

[Koupil y Holubová, 2022] Koupil, P. y Holubová, I. (2022). A unified representation and transformation of multi-model data using category theory. *Journal of Big Data*, 9(1):61.

[Liu *et al.*, 2018] Liu, Z. H., Lu, J., Gawlick, D., Helskyaho, H., Pogossiants, G., y Wu, Z. (2018). Multi-model database management systems-a look forward. En *VLDB Workshop on Data Management and Analytics for Medicine and Healthcare*, pp. 16–29. Springer.

[Lomakin y Garulli, 2023] Lomakin, A. y Garulli, L. (2023). OrientDB — github.com. https://github.com/orientechnologies. [Accessed 21-04-2025].

[Lu, 2017] Lu, J. (2017). Towards benchmarking multi-model databases. En *CIDR*.

[Lu y Holubová, 2019] Lu, J. y Holubová, I. (2019). Multi-model databases: A new journey to handle the variety of data. *ACM Comput. Surv.*, 52(3).

[Meier y Kaufmann, 2019] Meier, A. y Kaufmann, M. (2019). NoSQL Databases — link.springer.com. https://link.springer.com/chapter/10.1007/978-3-658-24549-8_7. [Accessed 28-07-2025].

[Mihai, 2020] Mihai, G. (2020). Multi-model database systems: The state of affairs. *Economics and Applied Informatics*, (2):211–215.

[MongoDB, 2021] MongoDB (2021). Data Modeling - Database Manual - MongoDB Docs — mongodb.com. https://www.mongodb.com/docs/manual/data-modeling/. [Accessed 29-07-2025].

[Neupane, 2024] Neupane, J. (2024). History of DBMS — jeevan.neupane003. https://medium.com/@jeevan.neupane003/history-of-dbms-a1791d52a253. [Accessed 28-07-2025].

[Oracle, 2019] Oracle (2019). Multimodel database. White Paper.

[Padhy *et al.*, 2011] Padhy, R. P., Patra, M. R., y Satapathy, S. C. (2011). Rdbms to nosql: reviewing some next-generation non-relational database's. *International Journal of Advanced Engineering Science and Technologies*, 11(1):15–30.

[Phiri, 2022] Phiri, M. (2022). EXPONENTIAL GROWTH OF DATA — mwaliph. https://medium.com/@mwaliph/exponential-growth-of-data-2f53df89124. [Accessed 28-07-2025].

[Pore, 2019] Pore, A. (2019). How to Design Schema for Your NoSQL Database? - DATAVERSITY — dataversity.net. https://www.dataversity.net/how-to-design-schema-for-your-nosql-database/. [Accessed 28-07-2025].

[Redis, 2024] Redis (2024). About - Redis — redis.io. https://redis.io/about/. [Accessed 07-05-2025].

[Sadalage y Fowler, 2013] Sadalage, P. J. y Fowler, M. (2013). *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education. Ver página 15 del prefacio.

[Scherzinger *et al.*, 2013] Scherzinger, S., Klettke, M., y Störl, U. (2013). Managing schema evolution in nosql data stores.

[Software, 2025] Software, R. (2025). DB-Engines Ranking — db-engines.com. https://db-engines.com/en/ranking. [Accessed 07-05-2025].

[SurrealDB, 2022] SurrealDB (2022). SurrealDB.