UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Un estudio de procesos de diseño de bases de datos NoSQL

AUTORES: Verena Macarena Olsowy

DIRECTOR: Pablo Thomas - Luciano Marrero

CODIRECTOR: -

ASESOR PROFESIONAL: Fernando Tesone

CARRERA: Licenciatura en sistemas

Resumen

Las bases de datos relacionales han sido la elección predominante en la historia de la gestión de datos, pero la creciente diversidad de requisitos ha generado la necesidad de explorar soluciones más flexibles. En este contexto, las bases de datos no relacionales surgen como una alternativa válida, aunque requieran enfoques y metodologías de diseño distintos a los tradicionales. Las metodologías tradicionales de diseño y construcción de bases de datos relacionales han sido ampliamente desarrolladas, pero no resultan apropiadas para las Bases de Datos no Relacionales. Adaptarse a estos cambios implica explorar nuevas formas de modelar y administrar los datos en el panorama actual.

Palabras Clave

Procesos de diseño.
Bases de Datos NoSQL.
DBMS NoSQL.
Procesos de diseño para bases de datos NoSQL.

Conclusiones

No se encontró ningún proceso de diseño que sea aplicable a todos los tipos de bases de datos NoSQL. No se encontró ningún proceso de diseño aplicable a bases de datos orientadas a grafos. Se puede apreciar que existen más enfoques de diseño para el tipo de bases de datos documentales en comparación con otros tipos de bases de datos, ya que las bases de datos documentales son de propósito general.

En resumen, este trabajo ofrece una visión integral y detallada de la situación actual para el diseño de bases de datos NoSQL, a través de un análisis minucioso de los enfoques de diseño y evaluación de su aplicabilidad.

Trabajos Realizados

Se realizó una recopilación y revisión sobre procesos de diseño de bases de datos NoSQL. Se estudiaron cada uno de los 18 enfoques de diseño encontrados. Se presentó un análisis comparativo de dichos enfoques. Se presentaron las conclusiones de acuerdo a los enfoques analizados y las comparaciones realizadas.

Trabajos Futuros

Como trabajo futuro, se pretende utilizar los procesos de diseño analizados en proyectos reales, a fin de evaluar su aplicabilidad.

Además, a partir de los hallazgos y conclusiones obtenidos en este estudio, se pretende presentar un proceso de diseño específico para alguno de los tipos de bases de datos NoSQL.

Fecha de la presentación:

Un estudio de procesos de Diseño de Bases de Datos NoSQL

Verena Macarena Olsowy

2023

Índice

Capítulo 1. Introducción..... 4
1.1 Motivación..... 4
1.2 Objetivo..... 5
Capítulo 2 - Tipos de Bases de Datos..... 6
2.1 Introducción..... 6
2.2 Base de Datos Jerárquico y de Red..... 7
2.3 Bases de Datos Relacional..... 9
2.4 Bases de Datos Orientado a Objetos..... 11
2.5 Bases de Datos No Relacionales (NoSQL)..... 11
Capítulo 3 - Bases de Datos No Relacionales (NoSQL)..... 13
3.1 Introducción..... 13
3.2 Tipos de bases de datos No Relacionales (NoSQL)..... 15
3.2.1 Almacenamiento Clave/Valor..... 16
3.2.2 Almacenamiento Documental..... 17
3.2.3 Almacenamiento de Familia de Columnas..... 19
3.2.4 Almacenamiento de Grafos..... 21
3.3 Motores de Bases de Datos No Relacionales (NoSQL)..... 22
Capítulo 4 - Diseño de Bases de Datos..... 26
4.1 Introducción..... 26
4.2 Diseño Jerárquico y de Red..... 26
4.3 Diseño Relacional..... 26
4.4 Diseño Orientado a Objetos..... 28
4.5 Diseño No Relacionales (NoSQL)..... 29
Capítulo 5 - Trabajos Relacionados..... 31
Capítulo 6 - Estudio de Procesos de Diseño en Bases de Datos NoSQL..... 32
6.1 Introducción..... 32
6.2 Enfoques para bases de datos documentales..... 32
6.2.1 Enfoque de Brahim, A. ; Ferhat, R. ; Zurfluh, G..... 32
6.2.2 Enfoque de Abdelhedi, F. ; Brahim, A. ; Ferhat, R. ; Zurfluh, G..... 34
6.2.3 Enfoque de C'anovasi, J. ; Jordi, C..... 34
6.2.4 Enfoque de Klettke, M.; Storl, U.; Scherzinger, S..... 36
6.2.5 Enfoque de Atzenia, P. ; Bugiottib, F. ; Cabibboa, L. ; Torlonea, R. (Documental)... 38
6.2.6 Enfoque de Rossel y Manna (Documental)..... 39
6.2.7 Enfoque de Varga V.; Jánosi-Rancz, K.; Balázs, K.;..... 42
6.3 Enfoques para bases de datos clave-valor..... 45
6.3.1 Enfoque de Rossel y Manna..... 45
6.4 Enfoques para bases de datos orientadas grafos..... 47

6.4.1 Enfoque de Comyn-Wattiau, I. ; Akoka, J.....47
6.4.2 Enfoque de De Virgilio, R; Maccioni, A; Torlone, R..... 48
6.4.3 Enfoque de Pokorný, J..... 50
6.5 Enfoques para bases de datos familia de columnas..... 52
6.5.1 Enfoque de Chebotko, A. ; Kashlev, A. ; Lu, S..... 52
6.5.2 Enfoque de Mior, M.; Salem, K.; Aboulnaga, A.; Liu, R.....54
6.6 Enfoques aplicables a más de un tipo de base de datos..... 56
6.6.1 Enfoque de De La Vega, A.; García-Saiz, D. ; Blanco, C.; Zorrilla, M.; Sánchez, P.. 56
6.6.2 Enfoque de Atzenia, P. ; Bugiottib, F. ; Cabibboa, L. ; Torlonea, R.....57
6.6.3 Enfoque de Banerjee, S. ; Sarkar, A.....59
6.6.4 Enfoque de Morales S. ; Garcia, J. ; Ruiz, S..... 60
6.6.5 Enfoque de Mok w..... 61
6.6.6 Enfoque de Gueidi A. ; Gharsellaoui, H. ; Ben Ahmed, S..... 63
Capítulo 7 - Análisis comparativo de Procesos de Diseño de Bases de Datos NoSQL..... 65
7.1 Resumen comparativo..... 65
Capítulo 8 - Conclusiones y trabajo futuro..... 69
Bibliografía..... 71

Capítulo 1. Introducción

1.1 Motivación

En la actualidad, la mayoría de las aplicaciones de software son multiplataforma. El avance en la comunicación digital, el acceso constante a la información (internet) y el aumento en la utilización de tecnología móvil, genera que estas aplicaciones sean de uso común por millones de usuarios simultáneamente. [1]

El modelo relacional de bases de datos (Codd 1970) [2, 3, 4] es, sin duda, el modelo predominante de almacenamiento de información. Sin embargo, la idea de considerar que un único modelo de datos pueda adaptarse de forma eficiente a todos los requerimientos ha sido discutida, y ha dado lugar a un conjunto de alternativas que plantean almacenar la información de forma no estructurada. Surgen así, nuevos Sistemas de Gestión de Bases de Datos que poseen sus propias implementaciones, no relacionales, y se denominan Bases de Datos NoSQL (No solo SQL).

Las Bases de Datos No Relacionales surgen como una alternativa de solución a problemas no resueltos eficientemente por los Sistemas de Gestión de Bases de Datos tradicionales. El término NoSQL no responde a un sólo tipo de Bases de Datos, sino que representa un conjunto de tipos de Bases de Datos, con diferentes formas y características para representar la información [5, 6, 7].

Las metodologías tradicionales de diseño y construcción de Bases de Datos (principalmente las relacionales) han sido ampliamente desarrolladas, estudiadas y aplicadas por décadas. En la actualidad, a pesar de los avances en los procesos de la Ingeniería de Software, en donde la documentación ha perdido protagonismo, estas metodologías continúan siendo un eslabón importante en la historia de las Base de Datos, como así también, para recabar requerimientos del usuario y del contexto.

Generalmente, un proceso de diseño para una Base de Datos Relacional, inicia con el análisis de una especificación de requerimientos en lenguaje natural, la cual deriva de la mirada de expertos sobre una problemática real. La especificación generada, posee la ambigüedad propia del lenguaje, lo que introduce un cierto grado de subjetividad al momento de plantear o pensar en un esquema de datos. Es importante tener en cuenta que el proceso de diseño de una Base de Datos será el que determine cómo estarán organizados los datos que contendrá la Base de Datos.

El diseño de una Base de Datos se realiza generalmente en tres fases o pasos. La primera, parte de la especificación de requerimientos y se genera una descripción conceptual, abstracta y de alto nivel, denominada “*esquema conceptual*”, en donde,

usualmente, se utiliza un DER (Diagrama de Entidades y Relaciones). En la segunda fase, se convierte el esquema conceptual en una especificación concreta *(esquema lógico*) que es aplicable a un tipo de base de datos. Esta fase se denomina "*diseño lógico*". En la tercera y última fase, se determinan las estructuras de almacenamiento físico resultando en lo que se conoce como *"esquema físico"* [8].

Los principios o reglas que se aplican a un esquema de datos relacional no resultan apropiados para una Base de Datos No Relacional (NoSQL). Esto se debe a que NoSQL no representa un tipo de Base de Datos, sino que son un conjunto de tipos de Bases de Datos y cada una de ellos posee su propia forma de estructurar y almacenar sus datos. En general, NoSQL se basa en admitir la redundancia, la desnormalización, para obtener eficiencia en las consultas que impactarán sobre el sistema. El modelo relacional y NoSQL proponen visiones diferentes, por lo tanto, su diseño también tendrá que ser diferente [9].

Generalmente, cuando se trabaja con Bases de Datos no relacionales, se plantea directamente su estructura a nivel físico teniendo en cuenta el problema a resolver y la flexibilidad de los esquemas no estructurados de datos. Realizar un esquema a nivel conceptual para una Base de Datos NoSQL es algo que aún es tema de discusión.

Un esquema de datos conceptual describe sus componentes en términos de entidades y cómo éstas se relacionan independientemente del tipo de base de datos y del motor de base de datos específico a utilizar.

1.2 Objetivo

Este trabajo pretende realizar un relevamiento exhaustivo de los distintos procesos de diseño para Bases de Datos No Relacionales (NoSQL) y un posterior análisis detallado para elaborar un informe sobre el tema.

Además se pretende realizar un análisis comparativo que resulte en una guía rápida que muestre las distintas características de cada enfoque.

Capítulo 2 - Tipos de Bases de Datos

2.1 Introducción

El término “base de datos” se planteó por primera vez en 1963 en un simposio en California y se lo definió como un conjunto de información relacionada que se encuentra agrupada ó estructurada.

Siempre existió la necesidad de guardar información, por eso los orígenes de las bases de datos provienen de la antigüedad, en donde existían grandes bibliotecas y toda clase de registros, entre ellos, los que se llevaban de las cosechas.

El origen de las bases de datos ligadas a la informática se remonta a 1884 con Herman Hollerith, quién desarrolló el tabulador electromagnético de tarjetas perforadas, con el objetivo de automatizar los trabajos de cómputos y clasificar grandes volúmenes de información. Posteriormente fue de gran ayuda para la contabilidad.

En la década del 50 aparecen las cintas magnéticas como medio para realizar copias de respaldo, pero con la desventaja de que solo se podía realizar recuperación de información de forma secuencial.

A partir de la década del 60 y con el avance de los años las computadoras fueron mejorando su tecnología y reduciendo su costo, popularizando la utilización de los discos magnéticos. En este contexto aparece la primera generación de bases de datos que incluye a las bases de datos jerárquicas, y luego, las bases de datos de red.

Edgar Frank Codd [3], científico informático inglés, definió el modelo relacional y publicó una serie de reglas para las bases de datos relacionales. Codd publicó su trabajo en 1970, denominado “*A Relational Model of Data for Large Shared Data Banks”[10]* (Un modelo relacional de datos para grandes bancos de datos compartidos). Posteriormente, Codd también definió las tres primeras formas normales que se aplican para la normalización de bases de datos y la forma normal de Boyce-Codd que lleva el nombre.

Más tarde, Lawrence “Larry” Ellison, aprovechó los aportes de Codd para desarrollar “Relational Software System”, lo que actualmente se conoce como Oracle Corporation, que se destacaba por su estabilidad, escalabilidad, transacciones y multiplataforma.

En la década del 80 se desarrolló el lenguaje de consulta estructurado (SQL – Structured Query Language) que permite realizar consultas con el fin de recuperar datos de interés de una base de datos relacional y realizar modificaciones de forma sencilla. SQL pasó a ser el estándar del Instituto Nacional Estadounidense de

Estándares (ANSI) en 1986 y de la Organización Internacional de Normalización (ISO) en 1987.

En la década del 90 surgieron las bases de datos orientadas a objetos que tuvieron éxito para representar otro tipo de datos (por ejemplo: audios, videos, imágenes, etc).

Los orígenes del término orientado a objetos se atribuye al paradigma orientado a objetos(OO).

Las bases de datos orientadas a objetos se diseñaron para trabajar eficientemente en conjunción con lenguajes de programación orientados a objetos. Los sistemas de gestión de base de datos orientados a objetos (ODBMS, por su sigla en inglés) usan el mismo paradigma que estos lenguajes de programación [2, 3].

En los últimos años, el avance en la comunicación digital, el acceso constante a la información, internet de las cosas (IoT, por su sigla en inglés) y el aumento en la utilización de tecnología móvil generan grandes volúmenes de datos, lo que ha cambiado la forma clásica de almacenar y consultar los datos.

En lo que se refiere a la gestión de los datos, los arquitectos y los desarrolladores de software se han visto en la necesidad de buscar nuevas soluciones que se adapten a distintos criterios para el análisis y explotación de los datos. Así surgen las bases de datos NoSQL.

El acrónimo NoSQL corresponde a "Not only SQL", originalmente "No SQL". Este término hace referencia a los sistemas de gestión de datos no relacionales y que por lo general no utilizan el lenguaje SQL para realizar las consultas.

Este tipo de bases de datos comenzaron a originarse en el año 1998 y no fueron creadas por una organización específica, sino que fueron concebidas por distintas organizaciones y grupos independientes que buscaban soluciones puntuales a sus problemas. Tampoco es un tipo de base de datos, sino que son un conjunto de tipos de bases de datos, ya que existen distintas categorías de almacenamiento para bases de datos NoSQL dependiendo las necesidades.

A continuación se detallan los aspectos más relevantes de los principales tipos de bases de datos.

2.2 Base de Datos Jerárquico y de Red

Los primeros Sistemas de Bases de Datos (SGBD) se basaban en el modelo jerárquico. En consecuencia, existía un amplio número de aplicaciones que utilizan Bases de Datos jerárquicas.

El modelo jerárquico posee dos conceptos importantes para definir su estructura, estos son: el tipo de registro y las interacciones “padre-hijo”. En este modelo los datos se encuentran organizados en forma de árbol.

Un esquema de Bases de Datos jerárquica contiene varias jerarquías, en donde cada una (o el esquema completo), consiste en varios tipos de registros y las respectivas relaciones “padre-hijo”, colocados de manera que formen un árbol.

Este es un modelo muy rígido, donde los diferentes conceptos de un determinado problema se organizan en múltiples niveles de acuerdo con la relación padre-hijo, es decir, que un registro padre puede tener varios descendientes y un registro hijo tendrá un único padre en el nivel superior inmediato. Este esquema de base de datos se caracteriza por:

- Una implementación que se lleva a cabo mediante el manejo de punteros o enlaces.
- Una estructura jerárquica que se organiza en un conjunto de niveles, siendo el nodo raíz, es el más alto y corresponde al nivel 0.
- El árbol tiene un único nodo raíz .
- Un nodo puede brindar acceso a un número ilimitado de otros nodos de nivel inferior (nodos hijos).
- Cada nodo, excepto la raíz, se corresponde con un solo nodo de nivel superior (nodo padre). Se denominan nodos “hojas” a todos los nodos que no poseen descendientes (nodos hijos).

El modelo jerárquico, posee restricciones inherentes a la estructura:

- Solo se tiene un único punto de entrada, la raíz.
- Permiten relaciones uno a uno, o uno a muchos pero, no permiten relaciones de muchos a muchos.
- No existen relaciones reflexivas.
- La estructura se debe recorrer siempre de acuerdo con un orden prefijado.
- Una vez creada la estructura no es posible su modificación.

El mantenimiento en las bases de datos jerárquicas requiere de una operatoria costosa, debido a sus restricciones. Toda inserción, excepto el nodo raíz, requiere de un nodo padre, se deben considerar los accesos en la búsqueda de este nodo.

Realizar un borrado, puede implicar que desaparezca todo el subárbol que posee dicho elemento como nodo raíz, se pueden perder datos importantes, se deben facilitar mecanismos para que esto no ocurra.

Posteriormente, y para intentar resolver las restricciones de las bases de datos jerárquicas, y como extensión de estas aparecen las Bases de Datos en Red.

En el caso del modelo de red, también existen dos conceptos básicos de estructuras, estos son: tipos de registros y tipos de conjuntos. Cada tipo de registro describe la estructura de un conjunto de registros que almacenan el mismo tipo de información. Un tipo de conjunto es una relación de uno a muchos entre dos tipos de registros.

El modelo de red es una extensión del modelo jerárquico, permitiendo relaciones N a N entre los registros, además de las relaciones 1 a 1, o 1 a N. Las relaciones existentes entre los nodos quedan expresadas en términos de las aristas existentes entre estos.

El modelo de red, es más flexible que el modelo jerárquico y esto se debe a la ausencia de las restricciones que posee dicho modelo. Éste modelo necesita añadir restricciones con el fin de poder realizar una implementación a nivel físico y obtener el mejor rendimiento del sistema.

Un ejemplo del modelo de datos de red es el "*modelo CODASYL*" (1969) [3, 11]. Este es un modelo simplificado del modelo de red original que solo admite cierto tipo de relaciones y posee un conjunto de restricciones adicionales con el objetivo realizar una implementación eficiente sin limitar la flexibilidad del modelo original.

2.3 Bases de Datos Relacional

El modelo relacional fue propuesto en 1970 por Edgar Frank Codd y desde entonces su popularidad fue creciendo a través de los años, llegando a ser el modelo más popular hasta la actualidad.

El modelo relacional es un modelo simple, potente y formal para representar la realidad. La sencillez del modelo ha posibilitado la construcción de lenguajes de consultas e interfaces amigables para los usuarios finales, lo que ha resultado en una mejor productividad para los proveedores de bases de datos. Este modelo de datos está basado en la teoría de las relaciones, donde los datos se estructuran lógicamente en forma de relaciones (o tablas), siendo el objetivo fundamental del modelo mantener la independencia de esta estructura lógica respecto al modo de almacenamiento y a cualquier otra característica de tipo físico [1].

El modelo relacional continúa siendo uno de los más populares y utilizados hasta la actualidad. Esto se debe a la sencillez de la representación lógica de sus datos y al uso de un lenguaje de definición y manipulación de datos [11].

Este modelo presenta los siguientes objetivos:

> *Independencia Física:* la forma en que se almacenan los datos no debe influir en su manipulación lógica. No se deben modificar los programas por cambios en el almacenamiento físico.
>
> *Independencia lógica:* los cambios en cualquiera de las relaciones, no debe afectar a quienes la estén utilizando.
>
> *Flexibilidad:* se puede ofrecer a cada usuario la visión más adecuada para su aplicación.

Uniformidad: las estructuras lógicas de los datos siempre se presentan en tablas.

Las bases de datos relacionales representan los datos en tablas compuestas por filas y columnas. Cada fila de la tabla corresponde a un registro individual y contiene un conjunto de columnas o "*atributos"* (nombre, apellido, DNI, etc.) que poseen una coherencia lógica entre sí.

Las tablas tienen un nombre único. Cada atributo posee un nombre único dentro de la tabla y un dominio. El dominio de un atributo es un conjunto finito de valores homogéneos y atómicos.

Además, dentro de una misma tabla, no existen filas duplicadas. En una tabla, no puede existir dos filas que posean el mismo valor en todas sus columnas, debe existir al menos un valor que las diferencie.

Cada tabla debe tener definida una clave primaria que puede formarse de un atributo o de un conjunto de atributos. Esta clave debe ser única y mínima, es decir, no puede existir más de una fila que tenga el mismo valor y no debe existir un subconjunto de la clave que sea único.

El modelo relacional permite relacionar filas de distintas tablas o incluso de la misma tabla a través de lo que se denomina clave foránea. El valor de una clave foránea en una fila, es el mismo valor que contiene la clave primaria de otra fila de la misma tabla u otra tabla con la que se quiere relacionar.

Existen también restricciones de integridad. Los datos almacenados deben cumplir ciertas reglas para garantizar la integridad de los datos en el modelo. Estas reglas son:

Integridad de dominio. Se debe definir el dominio de cada atributo, es decir, el conjunto de valores posibles que dicho atributo podrá tomar. El atributo no puede contener un valor por fuera de su dominio.

Integridad de clave primaria. El o los atributos que forman parte de la clave primaria pueden obtener valores nulos.

Integridad referencial. Esta regla establece que, si una relación contiene una clave foránea, su valor debe coincidir con el valor de clave primaria con el cual se establece el vínculo, o bien, contener un valor nulo.

Restricción o regla de negocio. Estas reglas no están definidas para el modelo relacional y dependen del dominio, el contexto o el usuario. A través de estas reglas se pueden establecer ciertas restricciones que son estratégicas y específicas para el funcionamiento del modelo.

2.4 Bases de Datos Orientado a Objetos

Este modelo define la base de datos como una colección de objetos.

En el tipo de base de datos relacional, los datos y el comportamiento se almacenan de forma separada, en cambio en el tipo de base de datos orientado a objetos se combinan en una misma entidad. Los objetos se dotan de un conjunto de características propias, que a su vez les diferencian de objetos similares. Los objetos similares se agrupan dentro de una clase y cada objeto es una instancia de una clase.

El tipo de Base de Datos Orientada a Objetos (BDOO) es una extensión del Paradigma de Programación Orientado a Objetos. El concepto de Objeto es análogo en las aplicaciones y en las Bases de Datos, con la diferencia que en la base de datos existe la persistencia, es decir el objeto no desaparece cuando finaliza su uso.

En este tipo de base de datos, las relaciones entre objetos se representan mediante el identificador de un objeto. Un identificador es un atributo que posee cada objeto, el cual no es manipulable directamente, su valor es administrado por el sistema.

Para que las relaciones funcionen, los identificadores de los objetos deben corresponderse en ambos extremos de la relación, es un tipo de integridad de relaciones similar a la integridad referencial de las Bases de Datos Relacionales, en donde se gestiona la verificación especificando relaciones inversas.

De la misma forma que en las bases de datos relacionales se deben especificar las reglas de integridad, en las BDOO se deben especificar las relaciones inversas indicando el rol de cada relación, es decir, el significado o correspondencia de cada atributo para ambos extremos del vínculo.

2.5 Bases de Datos No Relacionales (NoSQL)

Las bases de datos no relacionales representan un tipo que se divide en subtipos que poseen sus propias implementaciones para el almacenamiento de datos. Este tipo de base de datos no trabaja con estructuras definidas. Los datos no se almacenan en tablas, y la información tampoco se organiza en registros o campos. No utilizan el lenguaje SQL para las consultas, o al menos no lo usan como herramienta principal de consulta, sino como apoyo. Por eso se denominan Bases de Datos NoSQL o «no solo SQL».

Las bases de datos no relacionales son posteriores a las relacionales, y su desarrollo se ha basado en la necesidad de crear sistemas de gestión capaces de trabajar con datos no estructurados o semi-estructurados. Tienen una gran escalabilidad y están pensadas para la gestión de grandes volúmenes de datos.

En el siguiente capítulo se detallan las características principales de este tipo de base de datos.

Capítulo 3 - Bases de Datos No Relacionales (NoSQL)

3.1 Introducción

El término NoSQL fue utilizado por primera vez por Carlo Strozzi en 1998 para referirse a su base de datos que era open-source, que no ofrecía una interface SQL (Structured Query Language), pero sí seguía el modelo relacional.

Eric Evans, un empleado de Rackspace[12], volvió a utilizar el término NoSQL cuando Johan Oskarsson de Last.fm[13] quiso organizar un evento para discutir bases de datos distribuidas de código abierto. El término NoSQL intentaba englobar el número creciente de bases de datos no relacionales y distribuidas que no garantizaban ACID.

ACID es un grupo de 4 propiedades que garantizan que las transacciones en las bases de datos se realicen de forma confiable, ACID es el acrónimo de:

- Atomicidad: en una transacción que contiene una serie de pasos, se ejecutan todos o ninguno. Si una parte de la transacción falla, todas las operaciones de la transacción fallan, y por lo tanto la base de datos no sufre cambios. Se garantiza la atomicidad en cualquier operación y situación, incluyendo fallas de alimentación eléctrica, errores y caídas del sistema.
- Consistencia: asegura que cualquier transacción llevará a la base de datos desde un estado consistente a otro también consistente. Cualquier transacciones sólo debe cambiar datos afectados, y los datos escritos en la base de datos deben ser válidos según todas las restricciones y el esquema de datos definido.
- Aislamiento: cada transacción se debe ejecutar como única en el sistema. Si existen dos transacciones concurrentes, la ejecución de una no debe afectar a la otra. Cada transacción debe ejecutarse en aislamiento total.
- Durabilidad: una vez que se confirmó una transacción (commit), quedará persistida, incluso ante eventos como pérdida de alimentación eléctrica, errores y caídas del sistema.

NoSQL se distingue de las bases de datos relacionales en diversos aspectos, siendo el más notorio, el de no poseer un lenguaje de consulta estructurado (SQL, por sus siglas en inglés) como lenguaje principal. Mayormente no utilizan claves que permitan relacionar un conjunto de datos con otros, por lo que no soportan operaciones JOIN, no requieren de una estructura fija y tabular, no garantizan por completo las propiedades de ACID, y en general son adecuadas para la escalabilidad horizontal [14, 15, 16].

La escalabilidad horizontal es un modelo que implica tener diversos servidores que se conocen como nodos, trabajando como un todo. Se configuran en forma de una red de servidores denominada clúster. La finalidad es dividir eficientemente la demanda de trabajo entre todos los nodos que conforman la red de servidores.

Cuando el rendimiento del clúster se ve afectado por el incremento de usuarios u operaciones, se amplía la cantidad de nodos (“se escala”) para equilibrar la demanda de trabajo en cada nodo. Para lograr un óptimo funcionamiento de esta metodología debe existir un servidor primario desde el cual se ejecute la administración del clúster. Si un nodo presenta una falla, no se detendrá el funcionamiento del cluster porque los demás nodos seguirán funcionando.

Las bases de datos no relaciones sacrifican algunas características de las bases de datos relacionales en pos de nuevas prestaciones o mejoras en algunos aspectos, sin afectar el rendimiento del sistema. Para lograr esto, proponen una visión más liviana y optimista, aceptando que la consistencia en la base de datos sea flexible o que la disponibilidad se logre mediante el apoyo a fallas.

NoSQL propone un sistema denominado “BASE (**B**ase **A**vailability, **S**oft State y **E**ventual Consistency)” qué es diferente a las propiedades de ACID de las bases datos relacionales. A través de las propiedades BASE se logra disponibilidad básica (Base Availability), esto significa que el sistema se encontrará disponible la mayoría del tiempo. Con el estado débil (Soft State) el sistema se vuelve más flexible en cuanto a consistencia y con la consistencia eventual (Eventual Consistency) se garantiza que el sistema eventualmente se volverá consistente.

Las bases de datos NoSQL están pensadas para ser escalables y distribuidas, al ser distribuidas hay que tener en cuenta el teorema CAP.

El teorema CAP o teorema Brewer, plantea que en sistemas distribuidos no es posible garantizar consistencia, disponibilidad y tolerancia a particiones (Consistency-Availability-Partition Tolerance) al mismo tiempo.

- Consistencia: al realizar una consulta o inserción siempre se tiene que recibir la misma información, con independencia del nodo o servidor que procese la petición.
- Disponibilidad: que todos los usuarios puedan leer y escribir, aunque se haya caído alguno de los nodos.
- Tolerancia a particiones: los sistemas distribuidos pueden estar divididos en particiones, ésta condición implica que el sistema tiene que seguir funcionando aunque existan fallos o caídas parciales que dividan el sistema.

Según el teorema de CAP[17], las bases de datos NoSQL se puede clasificar en:

- AP: garantizan disponibilidad y tolerancia a particiones, pero no la consistencia, al menos de forma total. Pueden ofrecer una consistencia parcial a través de la replicación y la verificación.
- CP: garantizan consistencia y tolerancia a particiones. Para lograr la consistencia y replicar los datos a través de los nodos, sacrifican la disponibilidad.
- CA: garantizan consistencia y disponibilidad. Suelen gestionar la tolerancia a particiones replicando los datos.

Esto se puede ver gráficamente en la figura 1.

Las bases de datos NoSQL responden a la evolución en el almacenamiento de la información y no son exactamente un tipo de bases de datos, sino que son un conjunto de tipos de bases de datos. Están diseñadas específicamente para modelos de datos específicos y tienen esquemas flexibles para soportar aplicaciones con alta demanda de información.

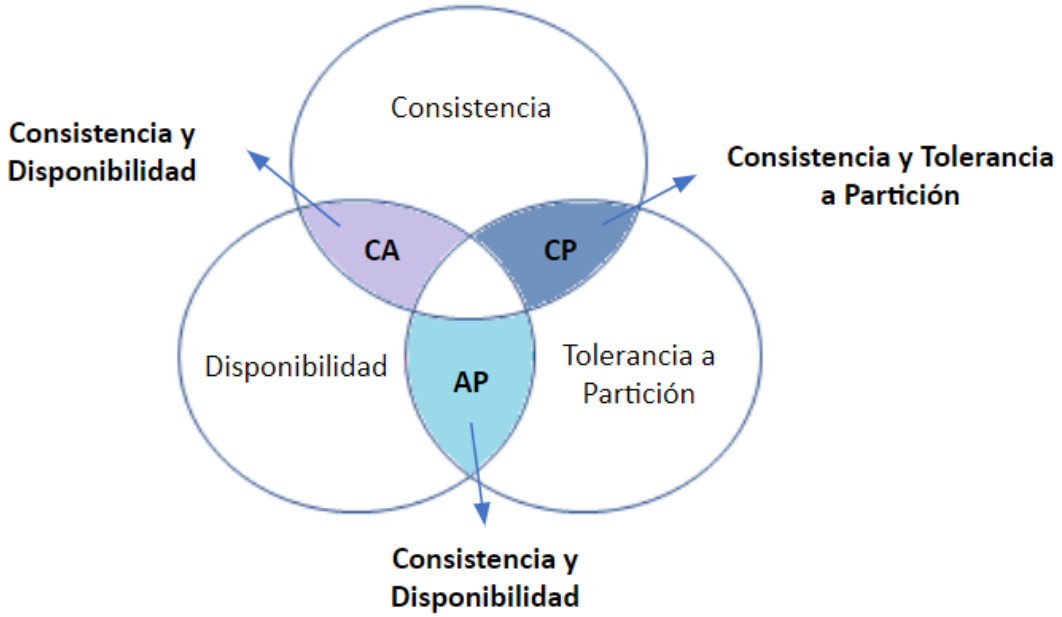

Figura 1. Teorema de CAP.

3.2 Tipos de bases de datos No Relacionales (NoSQL)

Existen cuatro tipos principales de almacenamiento para bases de datos NoSQL:

- Almacenamiento Clave/Valor
- Almacenamiento Documental
- Almacenamiento Familia de columnas

- Almacenamiento de Grafos

3.2.1 Almacenamiento Clave/Valor

En este tipo de almacenamiento se almacenan datos como un conjunto de pares “clave/valor” (key-value), donde la clave representa un identificador que puede retornar un objeto. Tanto las claves como los valores pueden ser desde objetos simples hasta objetos compuestos complejos.

Este tipo de base de datos administra los datos de forma diferente a las bases de datos relacionales, ya que las bases de datos relacionales predefinen una estructura de campos con tipos bien definidos para cada tabla. Por el contrario, los sistemas de clave-valor tratan los datos como una única colección, que puede tener campos diferentes para cada registro, lo cual ofrece una flexibilidad considerable.

Dado que los valores opcionales no se representan mediante marcadores de posición o parámetros de entrada, las bases de datos clave-valor suelen utilizar menos memoria que otro tipo de base de datos para almacenar la misma información.

Al igual que el resto de las Bases de Datos NoSQL, no cuenta con lenguaje estándar para el manejo de datos, mayormente cuentan con lenguajes de consultas básicos, lo cual impide realizar consultas complejas. Las consultas y el ordenamiento de la información pueden ser realizados sólo a partir de la clave primaria.

Este tipo de base de datos está diseñada para almacenar, recuperar y administrar arreglos asociativos (diccionario o tabla hash). Los diccionarios contienen una colección de objetos, o registros, que a su vez poseen muchos campos diferentes internamente. Estos diccionarios se almacenan bajo una clave que identifica de manera única el registro.

El uso principal de este tipo de almacenamiento se centra en tres operaciones básicas

- get para obtener datos vinculados a una clave
- put para vincular un valor determinado a una clave
- delete para eliminar una entrada con una clave

Este tipo de base de datos se puede considerar utilizar cuando los datos no son estructurados o cuando no es importante establecer relaciones entre los datos. Además, puede ser utilizada en grandes proyectos que necesiten, por ejemplo, registro de sesiones, carritos de compras on-line y/o administrar un gran número de usuarios en tiempo real, sin ralentizar las operaciones.

Además estas bases de datos pueden manejar pérdida de nodos de almacenamiento, ya que poseen propiedades de redundancia incorporada.

Al almacenar los datos en diccionarios, se garantiza tener la información en una clasificación funcional que se puede recuperar y usar en cualquier momento. La velocidad y escalabilidad que ofrecen son de los puntos más fuertes de estas bases de datos.

La figura 2 ilustra un ejemplo de la estructura de este tipo de base de datos, donde se puede observar que para las diferentes claves (k1-k5) se puede recuperar distintas cantidades y tipos de datos, estos datos se encuentran separados por coma.

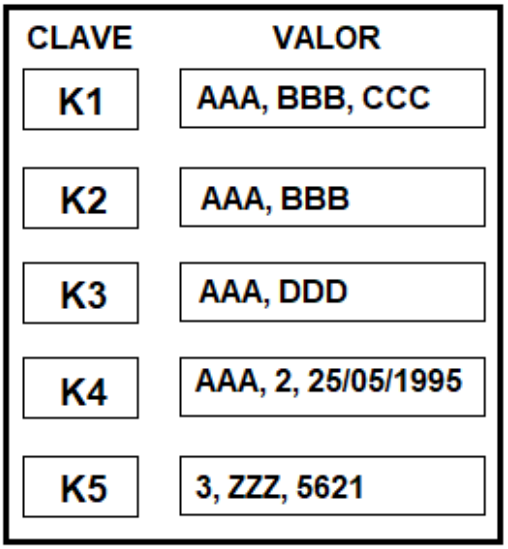

Figura 2: ejemplo de estructura clave-valor

3.2.2 Almacenamiento Documental

Este tipo de base de datos almacena, recupera y gestiona datos en documentos de forma semiestructurada sin un esquema predefinido. Los documentos se pueden estructurar de forma diferente. Al contrario que una base de datos relacional en la que todos los registros deben tener los mismos atributos (que pueden quedar vacíos), en un documento no existen atributos vacíos. De este modo es posible añadir nueva información sin necesidad de modificar una estructura predefinida.

Los datos no estructurados se pueden almacenar fácilmente, ya que cada documento contiene las claves y valores que la lógica de la aplicación requiere. La información no se reparte en varias tablas enlazadas, sino que se almacena en el mismo documento, por lo que se puede tener documentos anidados (documentos dentro de documentos), lo que puede mejorar el rendimiento.

Los documentos encapsulan y codifican datos o información bajo algún formato estándar (XML, YAML, JSON, BSON). Se agrupan en colecciones que tienen un

propósito análogo al de una tabla relacional, suele haber múltiples colecciones dentro de una base de datos documental. Existen distintas implementaciones de este tipo de base de datos y cada una se estructura de forma diferente, entre las que se encuentran colecciones, etiquetas, jerarquía de directorios, etc.

Las bases de datos de documentos adicionalmente proporcionan mecanismos de consulta para las colecciones de documentos permitiéndole estudiar de forma individual o conjunta los atributos particulares o especiales que pueda tener. A diferencia de las bases de datos relacionales, no existe un lenguaje estándar de consultas definido para las bases de datos documentales.

Los documentos se direccionan mediante una clave única que identifica el documento. Generalmente esta clave se compone de una simple cadena de caracteres, pero podría tratarse de un URL o un camino, que sirve para recuperar el documento de la base de datos. Generalmente la base de datos mantiene un índice de dichas claves, por lo que la recuperación por clave es más rápida.

Algunos casos de usos pueden ser:

- Administración de contenido, como blogs y plataformas de vídeo ya que, cada entidad que rastrea la aplicación se puede almacenar como un único documento. Si el modelo de datos necesita cambiar, solo se deben actualizar los documentos afectados. Al no existir un esquema predefinido no se requiere actualización del esquema y no se necesita tiempo de inactividad de la base de datos para realizar los cambios.
- Catálogos de productos, los productos generalmente tienen cantidades diferentes de atributos. Los atributos de cada producto se pueden describir en un solo documento para que la administración sea fácil y la velocidad de lectura sea más rápida. Cambiar los atributos de un producto no afectará a otros productos.

La figura 3 ilustra un ejemplo de la estructura de este tipo de base de datos. Donde existen varias colecciones, una es de personas, y se muestran dos documentos en detalle de dos personas.

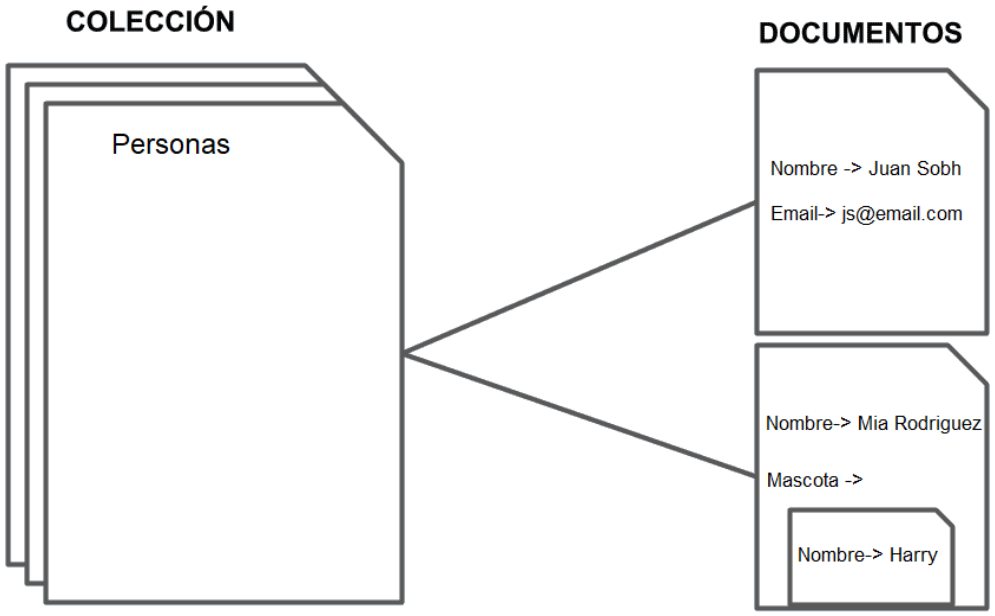

Figura 3: ejemplo de estructura documental

3.2.3 Almacenamiento de Familia de Columnas

En este tipo de almacenamiento los datos se encuentran organizados por columnas, en lugar de filas.

Las bases de datos de columnas utilizan un concepto llamado Keyspace. Un Keyspace contiene todas las familias de columnas, es decir, contiene a la base de datos.

En la figura 4 se muestra un KeySpace con dos familias de columnas(Clientes y Productos). Cada familia de columnas está compuesta por filas de datos.

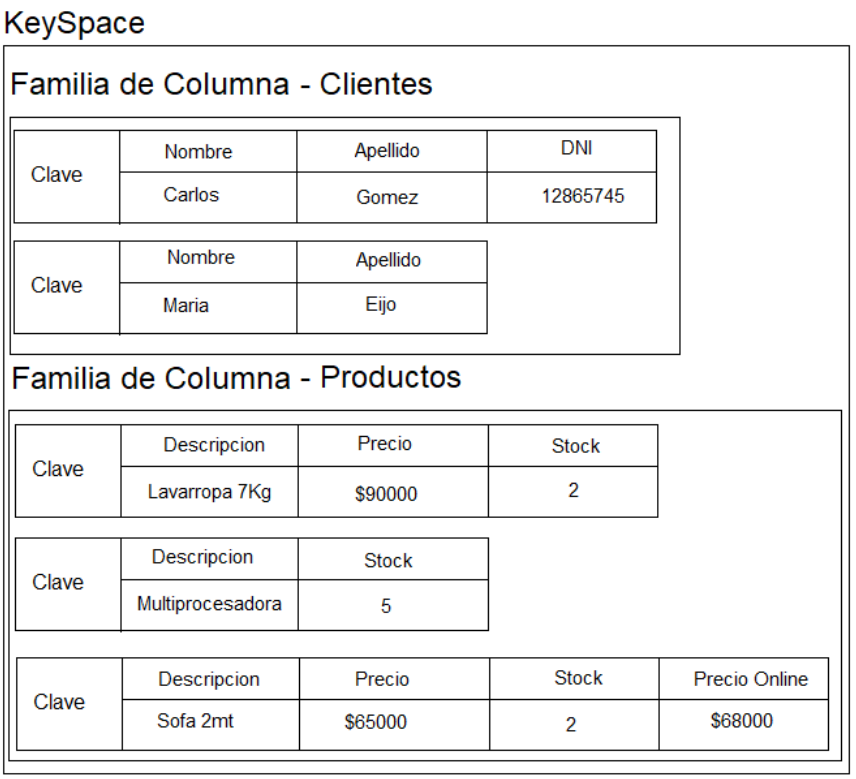

Figura 4: ejemplo de estructura Familia de Columnas

Cada familia de columnas consta de varias filas. Cada fila se forma de una clave y puede contener una cantidad diferente de columnas. Las columnas no tienen que coincidir con las columnas de las otras filas (es decir, pueden tener diferentes nombres de columna, tipos de datos, etc.). Cada columna contiene un par de nombre/valor, junto con una marca de tiempo.

En la figura 5, se muestra gráficamente la estructura de una fila.

Los conceptos utilizados en el almacenamiento Familia de Columnas son:

- Row Key o Llave de fila: Cada fila tiene una clave única, que es un identificador único para esa fila.
- Columna: contiene un nombre, un valor y una marca de tiempo (Name, Value, Timestamp)
- Nombre: Este es el nombre del par nombre/valor.
- Valor: Este es el valor del par nombre/valor.
- Marca de tiempo: Esto proporciona la fecha y hora en que se insertaron los datos. Esto se puede usar para determinar la versión más reciente de los datos.

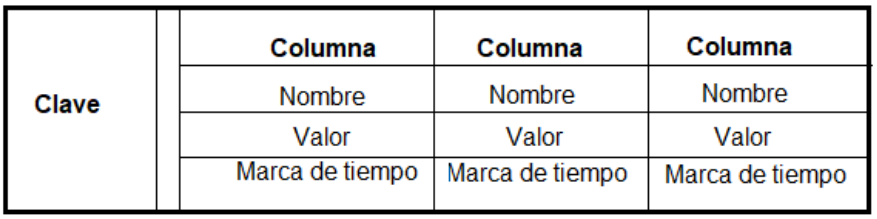

Figura 5: estructura de fila

En este tipo de bases de datos una de las principales ventajas es la compresión y/o partición de datos. La compresión permite que las operaciones en columna, como MIN, MAX, SUM, COUNT y AVG, se realicen muy rápidamente. Debido a su estructura, las bases de datos en columnas funcionan particularmente bien con las consultas de agregación (como SUM, COUNT, AVG, etc.).

Algunas bases de datos de columnares sólo se pueden acceder utilizando su propio proveedor de lenguaje de consultas y herramientas.

3.2.4 Almacenamiento de Grafos

Este tipo de almacenamiento representa la información en nodos y aristas cumpliendo con la disciplina matemática de la Teoría de Grafos. Un grafo se compone por dos elementos: los nodos (vértices) y las relaciones (aristas). Un nodo representa una entidad, en el que se almacenan piezas de datos o atributos de tipo clave-valor, mientras que las relaciones representan cómo se conectan y se asocian dos nodos. Permite almacenar la información como nodos de un grafo y sus respectivas relaciones con otros nodos, y se aplica la teoría de grafos para recorrer el grafo, es decir, la base de datos. Algunas implementaciones permiten guardar datos y establecer etiquetas en las propias relaciones que servirán a posterior en las consultas. Estas Bases de Datos son muy útiles para almacenar información en modelos con muchas relaciones entre distintas entidades o nodos.

Sus características principales son:

- Son multidimensionales, pueden almacenar atributos de diverso tamaño en los nodos
- Las relaciones pueden almacenar atributos
- Las relaciones pueden ser sin dirección, unidireccionales y bidireccionales lo que puede convertir la representación a grafos dirigidos, muy útiles en el cálculo de caminos.
- Tienen alto rendimiento en la búsqueda de resultados y sobre todo en la búsqueda de caminos.

Existen diferentes maneras para realizar consultas en este tipo de almacenamiento, principalmente porque no poseen un lenguaje de consulta uniforme. Al contrario de lo que ocurre en modelos tradicionales, las bases de datos orientadas a grafos utilizan

algoritmos especiales para poder realizar su principal tarea: simplificar y acelerar consultas complejas.

Algunos algoritmos son la búsqueda en profundidad y la búsqueda en anchura. La búsqueda en profundidad busca el siguiente nodo más profundo, mientras que la búsqueda en anchura se mueve de un nivel a otro. Los algoritmos permiten encontrar patrones (los llamados graph patterns) y nodos relacionados directa o indirectamente.

Otros algoritmos también se ocupan de calcular la ruta más corta entre dos nodos y de identificar cliques (subconjuntos de nodos) y hotspots (datos altamente interconectados). Una de las ventajas de las bases de datos orientadas a grafos es que las relaciones están guardadas en la propia base de datos, de manera que no son calculadas a partir de cada solicitud de búsqueda. Gracias a ello, la base de datos opera a gran velocidad incluso en búsquedas complicadas.

La figura 6 ilustra un ejemplo de la estructura de este tipo de base de datos.

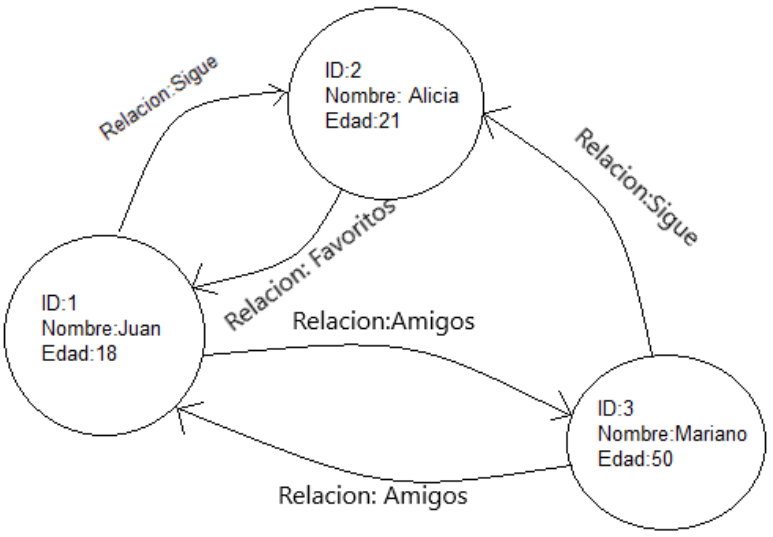

Figura 6: ejemplo de estructura de grafo

3.3 Motores de Bases de Datos No Relacionales (NoSQL)

Existen distintas implementaciones para los distintos tipos de bases de datos, a continuación se detallan algunos de cada tipo.

Almacenamiento clave/valor:

- Redis [18]

Redis es un almacén de estructura de datos de clave-valor que trabaja en memoria principal con opción de persistencia. Es de código abierto.

Gracias a su velocidad y facilidad de uso, Redis es una opción popular para aplicaciones web, móviles, de juegos, de tecnología publicitaria, que requieren el mejor desempeño de su clase. Entre los usos principales de Redis se encuentran el almacenamiento en caché, la administración de sesiones, publicación/suscripción y las clasificaciones.

Tiene licencia BSD, está escrito en código C optimizado y admite numerosos lenguajes de desarrollo. Redis es el acrónimo de REmote DIctionary Server (servidor de diccionario remoto).

- DynamoDB [19]

  Es una base de datos híbrida entre clave-valor y documento.

  Esta base de datos NoSQL de Amazon Web Service admite modelos de datos basados en clave-valor y documentos que se adecuan para los desarrolladores de aplicaciones modernas que no poseen servidor propio.

  Entre los principales atributos novedosos que ofrece DynamoDB se encuentra su modo operativo de capacidad bajo demanda, su compatibilidad integrada con transacciones ACID, copias de seguridad bajo demanda, modo de recuperación inmediata y el cifrado de datos en reposo.

Almacenamiento documental:

- MongoDB [20]

  MongoDB cuenta con un formato de almacenamiento de documentos en un formato similar al JSON (JavaScript Object Notation). Está codificado en lenguaje C++, es multiplataforma, de código abierto y completamente gratuito.

  MongoDB cuenta con un almacenamiento flexible basado en JSON. Además tiene soporte para la creación de índices desde cualquier atributo y no requiere la definición de procesos.

- CouchDB [21]

  CouchDB es una base de datos NoSQL de código abierto pensada para facilitar la accesibilidad y compatibilidad web con diversos dispositivos. Se diseñó desde el principio para prestar sus servicios en Internet.

  En esta Base de Datos los datos se almacenan en formato JSON y están organizados en pares de valor clave. La clave es un identificador único de datos y el valor es el propio dato o un apuntador de la ubicación de los

datos. Las funciones estándar de la base son realizadas por JavaScript. Cuenta con indexación, búsqueda y recuperación rápida de datos.

CouchDB posee la capacidad de hacer replicación fácil entre servidores.

Almacenamientos familia de columna:

- Cassandra [22]

  Es un sistema de gestión de bases de datos (DBMS) de código abierto, escrito en Java, que permite grandes volúmenes de datos en forma distribuida. Los datos son replicados automáticamente en varios nodos para la tolerancia a errores. Admite la replicación en varios centros de datos y los nodos con errores pueden ser reemplazados.

  Su objetivo principal es la escalabilidad horizontal y la disponibilidad. La arquitectura distribuida de Cassandra está basada en una serie de nodos iguales que se comunican con un protocolo P2P con lo que la redundancia es máxima.

  Cassandra es desarrollada por la Apache Software Foundation. En las versiones iniciales utilizaba un API propia para poder acceder a la base de datos. En los últimos tiempos están optando por usar un lenguaje denominado CQL (Cassandra Query Language) que posee una sintaxis similar a SQL aunque con muchas menos funcionalidades. Permite acceder en Java desde JDBC.

- Apache HBase [23]

  Es un gestor de base de datos de código abierto modelado a partir de Google BigTable y codificado en Java. Su desarrollo forma parte del proyecto Hadoop de la Fundación de Software Apache y se ejecuta sobre el sistema de archivos distribuidos de Hadoop.

  Las tablas en HBase pueden servir como entrada o salida para tareas MapReduce en Hadoop, y se puede acceder a través del API en Java, como servicio REST, o con los API de conexión Avro y Thrift. Es adecuado para acelerar operaciones de lectura y escritura en los grandes conjuntos de datos con un alto rendimiento y una baja latencia de entrada/salida.

  Actualmente, Hbase se utiliza en sitios web orientados a datos, incluyendo la plataforma de mensajería de Facebook.

  En términos del teorema CAP de Eric Brewer, HBase es un tipo de sistema CP (consistencia y tolerancia a particiones).

Almacenamiento de grafos:

- Neo4j [24]

  Neo4j es el pionero de este tipo de bases de datos. Es de código abierto, por lo que su comunidad es muy numerosa.

  Neo4j ha desarrollado su propio lenguaje declarativo para grafos, denominado Cypher. Además, Neo4j cuenta con una amplia galería de algoritmos para realizar consultas en los datos y también admite la vinculación a otros softwares.

- Memgraph [25]

  Memgraph es una base de datos de código abierto diseñada para el streaming en tiempo real. Está dirigida tanto a desarrolladores como científicos de datos que trabajan con datos interconectados, Memgraph otorga rápidamente información procesable inmediata. Ofrece una interfaz estándar para consultar los datos utilizando Cypher, un lenguaje de consulta ampliamente utilizado y declarativo.

  Una de las características distintivas de Memgraph es su capacidad para conectarse directamente a la infraestructura de streaming. Esto permite la ingestión de datos desde diversas fuentes, como Kafka, SQL o archivos CSV, lo que facilita la integración de Memgraph en sistemas existentes.

  Memgraph está implementada en C/C++ y utiliza una arquitectura "in-memory first", lo que significa que prioriza el almacenamiento en memoria para garantizar un rendimiento óptimo y constante, sin sorpresas. Además, es compatible con ACID, lo que asegura la integridad de los datos, y ofrece alta disponibilidad para asegurar la continuidad del servicio.

Capítulo 4 - Diseño de Bases de Datos

4.1 Introducción

El diseño de base de datos es un proceso cuyo objetivo es definir la estructura adecuada para el almacenamiento de datos de un sistema de información. Una base de datos bien diseñada influye de forma directa en la eficiencia para almacenar, recuperar y analizar los datos.

4.2 Diseño Jerárquico y de Red

Para las bases de datos jerárquicas y en red no se ha encontrado un proceso de diseño definido.

En los años que se utilizaban este tipo de bases de datos, los diseños de bases de datos eran generados con técnicas más artesanales que sistémicas que se valían de herramientas muy rudimentarias [2].

4.3 Diseño Relacional

El diseño de base de datos relacional consta de un proceso bien definido que está considerado un estándar de facto. Está compuesto por distintas etapas que se ejecutan en modo secuencial. A partir de la lista de requerimientos existen tres etapas o fases: el diseño conceptual, diseño lógico y diseño físico. La figura 7 muestra las etapas del proceso de diseño de bases de datos relacionales.

En el diseño conceptual se presenta una descripción a alto nivel del contenido de la base de datos, independientemente del tipo de sistema de gestión de base de datos que se utilizará.

Se define un diagrama que contiene las entidades, sus atributos y las relaciones entre ellas. El objetivo es representar las necesidades indicadas en la especificación de requerimientos y no las estructuras de almacenamiento que se necesitarán para manejar dicha información.

El modelo más ampliamente utilizado para las bases de datos relacionales es el modelo Entidad Relación (ER), el cual presenta las entidades de datos, sus atributos asociados y las relaciones entre las entidades.

En el modelo ER las entidades representan un elemento de utilidad para el problema que es distinguible del resto. Cada entidad se caracteriza por un conjunto de atributos. Y una relación establece un nexo o vínculo entre las entidades.

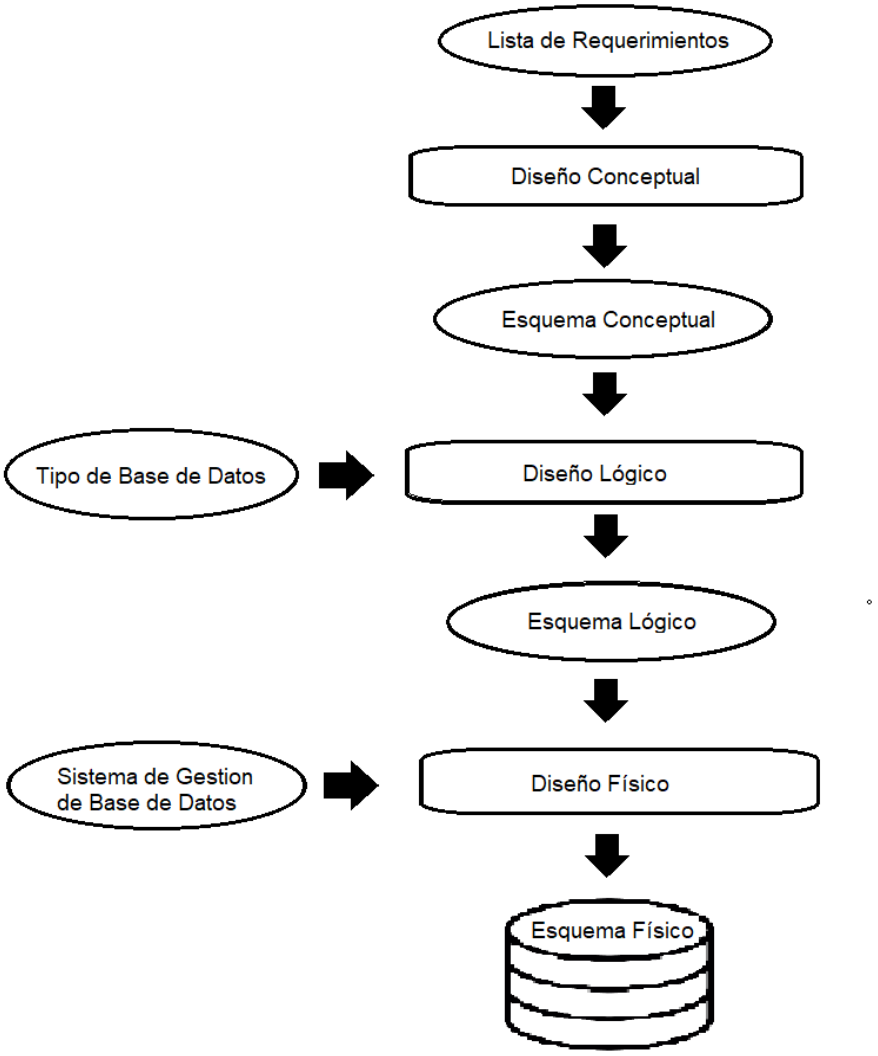

Figura 7 - Proceso de diseño de las bases de datos relacionales

A partir del esquema conceptual obtenido en la etapa anterior, es necesario decidir el tipo de base de datos que se utilizará para realizar el diseño lógico. En este punto se asume un sistema de gestión de bases de datos relacional.

En el diseño lógico se realiza una conversión del esquema conceptual a un esquema lógico aplicando reglas de decisiones asociadas al modelo relacional. Estas reglas están vinculadas al rendimientos y características del esquema conceptual que no es posible representar en los sistemas de gestión de bases de datos relacionales. Por ejemplo: jerarquías, atributos polivalentes y atributos compuestos.

A partir del esquema lógico, es necesario definir el motor de base datos específico a utilizar para realizar el diseño físico de la base de datos.

En el diseño físico se definen las estructuras de almacenamiento de la base de datos de forma física. A partir del esquema lógico se realiza la conversión de entidades relaciones y atributos a una descripción de la implementación de una base de datos

en memoria secundaria, es decir, un esquema físico. En este punto diseñan las tablas propiamente dichas (estructuras bidimensionales con filas y columnas) y cómo se relacionan entre sí. Aquí se tienen en cuenta aspectos concretos del sistema de gestión de bases de datos sobre el que se vaya a implementar.

4.4 Diseño Orientado a Objetos

El diseño orientado a objetos es el mismo que el diseño en los lenguajes de programación orientados a objetos. Las bases de datos orientadas a objetos permiten la persistencia de los objetos que manejan los lenguajes de programación orientados a objetos.

El diseño orientado a objetos se encarga de definir los objetos y las interacciones entre ellos.

En primer lugar se requiere saber cuáles son los objetos del programa, así como las relaciones entre éstos. Los objetos necesitan interactuar y comunicarse, para realizar dicha comunicación, los objetos utilizan su propia interfaz pública, dicha interfaz se compone principalmente de métodos y propiedades.

El desarrollo de cualquier sistema consiste en realizar tres etapas: análisis, diseño y programación.

Durante el proceso de análisis, en el diseño orientado a objetos, se realiza el modelado y la declaración de objetos. El manejo de los objetos está determinado por las implementaciones de los métodos detectados en el análisis.

Componentes del Diseño Orientado a Objetos:

- La identificación de objetos, sus atributos y servicios.
- Identificación de las relaciones entre clases (generalizaciones, agregaciones y asociaciones).
- La construcción de descripciones dinámicas de objetos que muestran cómo se usan los servicios.
- La especificación de interfaces de objetos.

Para la construcción de modelos de objetos, y modelos software en general, se debe contar con un lenguaje de modelado, es decir, un lenguaje para especificar, visualizar, construir y documentar los componentes de los sistemas software. El Lenguaje Unificado de Modelado (UML) se utiliza como el lenguaje de modelado que se ha convertido en estándar de facto de las notaciones en orientación a objetos.

UML desempeña un rol importante, ya que es una forma de mostrar visualmente el comportamiento y la estructura de todos los objetos de un sistema o proceso. Se trata

de un estándar que se ha adoptado a nivel internacional por numerosos organismos y empresas para crear esquemas, diagramas y documentación relativa a los desarrollos de software. En la figura 8 se puede ver un ejemplo de un modelado en UML [26].

El término “lenguaje” ha generado bastante confusión respecto a lo que es UML, ya que no es un lenguaje propiamente dicho, sino una serie de normas y estándares gráficos respecto a cómo se deben representar los esquemas relativos al software. UML son una serie de normas y estándares que dicen cómo se debe representar algo.

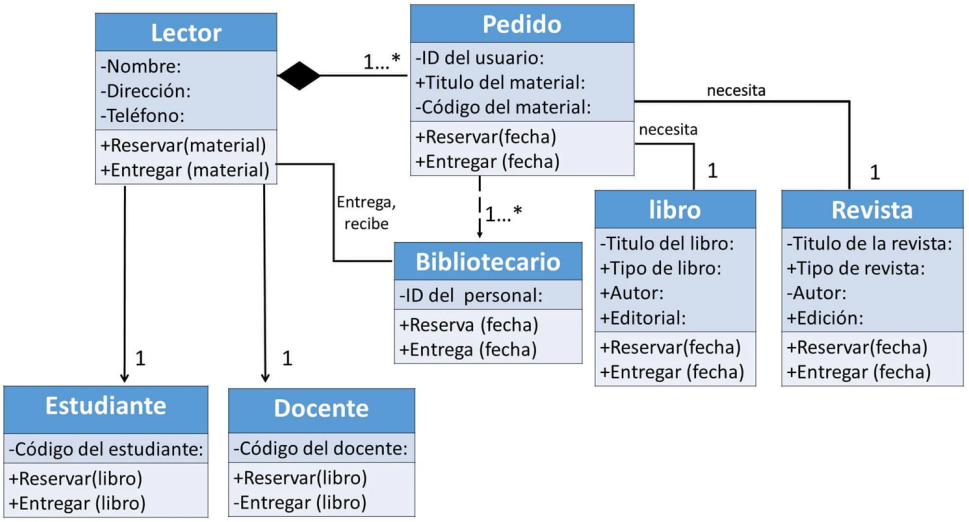

Figura 8. Representación de una Base de Datos Orientada Objetos mediante la utilización de un Diagrama de Clases que representa un sistema bibliotecario [27].

4.5 Diseño No Relacionales (NoSQL)

Las metodologías tradicionales de diseño y construcción de bases de datos relacionales han sido ampliamente estudiadas, aplicadas y refinadas por décadas. Los principios o reglas que se aplican a un modelo de datos relacional no resultan apropiados para una base de datos no relacional (NoSQL). Esto se debe a que NoSQL no representa un tipo de base de datos, sino que son un conjunto de tipos de bases de datos y cada una de ellas posee su propia forma de estructurar y almacenar sus datos. En general, la estructura de la información en una base de datos no relacional, está basada en la redundancia, la desnormalización y las consultas que impactarán sobre el sistema. Las bases de datos relacionales y NoSQL proponen modelos diferentes, por lo tanto, su diseño también tendrá que ser diferente.

Este trabajo presenta una recopilación sobre procesos de diseños existentes para bases de datos NoSQL, los cuales son presentados en el siguiente capítulo.

En [9] se presentó un resultado preliminar de este trabajo. Además en [28] se realizó un análisis de procesos de diseño encontrados dentro de los enfoques estudiados en [9].

Capítulo 5 - Trabajos Relacionados

En las publicaciones analizadas se han encontrado sólo tres trabajos relacionados que realizan revisiones bibliográficas sobre diseño de Bases de Datos NoSQL.

En [29] se realiza una revisión sistemática de la literatura para el diseño de Bases de Datos no Relacional. El análisis realizado en [29] se basa en aplicar 3 preguntas a cada bibliografía analizada. Las preguntas que se aplican son las siguientes:

1. ¿Qué métodos de diseño se han desarrollado en respuesta a la nueva era de las bases de datos?
2. ¿Existen enfoques integradores para el manejo de una mezcla de BDs que emplean diferentes modelos.
3. ¿Cuáles son las futuras direcciones de investigación para el diseño de las bases de datos en la nueva era?

Además se muestran estadísticas de acuerdo al material analizado.

En [30] se realiza una revisión de los enfoques existentes para el diseño de Bases de Datos en general, se presenta un breve resumen de cada uno con el objetivo de detectar consideraciones y necesidades en comparación al diseño tradicional de Base de Datos (Relacional). En el análisis presentado se detalla en líneas generales el estado del arte a partir de la bibliografía analizada. También se enuncia las necesidades que existen sobre el diseño de las bases de datos NoSQL.

En [31] se realiza un estudio exhaustivo que ayuda a la elección del gestor de base de datos a utilizar ante distintas características y decisiones de diseño para el almacenamiento no estructurado de información; además se incluyen otras temáticas como modelo de consistencia, partición de datos y teorema de CAP.

En este trabajo, a diferencia de los enunciados anteriormente, se consideran los siguientes aspectos::

- Se presenta una breve explicación de cada enfoque, que permite entender rápidamente lo que realiza cada uno.
- Se realiza un análisis comparativo expresado en tablas que permiten una rápida comparación. Estas tablas se encuentran divididas por el tipo de base de datos al que son aplicables, lo que ayuda a elegir un enfoque si ya se tiene decidido el tipo de almacenamiento a utilizar. Además se plantea una pequeña recomendación sobre qué enfoques se pueden aplicar dependiendo el tipo de base de datos a utilizar.
- Finalmente se presenta un resumen del estado del arte.

Capítulo 6 - Estudio de Procesos de Diseño en Bases de Datos NoSQL

6.1 Introducción

Con el objetivo de analizar la situación actual para el diseño de Base de Datos que utilizan almacenamiento no estructurado, en esta sección se presenta un estudio sobre 23 propuestas de procesos de diseño de Bases de Datos NoSQL.

Cuando se piensa en el diseño de una Base de Datos NoSQL, generalmente, se presenta directamente su estructura a nivel físico teniendo en cuenta el problema a resolver. Realizar un modelo de datos en un nivel conceptual para NoSQL es algo que aún es tema en discusión. Generalmente, un modelo de datos conceptual describe sus componentes en términos de entidades y cómo éstas se relacionan, independientemente del tipo de base de datos a utilizar. Existen aspectos como la redundancia, la desnormalización y las consultas del sistema que en una etapa conceptual aún no son consideradas.

A continuación, se presenta un relevamiento de trabajos que han abordado la temática de procesos de diseño para Bases de Datos NoSQL desde diferentes perspectivas.

Para la búsqueda de los trabajos se han consultado diversas fuentes como Google Scholar [34], IEEE Xplore [32], Springer [33], ResearchGate [35], entre otros.

Los trabajos relevados se separaron de acuerdo al tipo de base de datos NoSQL al que son aplicables. Algunos de ellos son aplicables a más de un tipo de base de datos NoSQL.

En el punto 5.7, se presentan trabajos relacionados.

6.2 Enfoques para bases de datos documentales

En este punto se analizan los procesos de diseño que se aplican sólo a base de datos documentales.

6.2.1 Enfoque de Brahim, A. ; Ferhat, R. ; Zurfluh, G.

En [36] se propone un proceso automático para obtener un modelo físico a partir del motor de base de datos no relacional documental. Este proceso se basa en una

Arquitectura Dirigida por Modelos (Model-Driven Architecture o MDA) que proporciona un marco formal para automatizar las transformaciones de modelo.

El proceso genera un modelo físico NoSQL a partir de una base de datos NoSQL orientada a documentos aplicando una secuencia de transformaciones formalizadas con el estándar Query/View/Transformation (QVT). El modelo resultante describe las estructuras de las colecciones de la base de datos, como así también las relaciones entre ellas.

Se plantea un caso de estudio que cumple con las características de Big Data, es decir, las 3V (Volumen, Variedad y Velocidad). La información recogida puede sumar varios terabytes, es diferente en cuanto a su estructura y además se produce información de forma continúa, casi en tiempo real. El caso de estudio es una aplicación médica que se ocupa de programas científicos para el seguimiento de patologías. La base de datos utilizada es MongoDB.

El proceso se denomina “ToNoSQLModel Process”. La figura 9 presenta una visión general del proceso realizado.

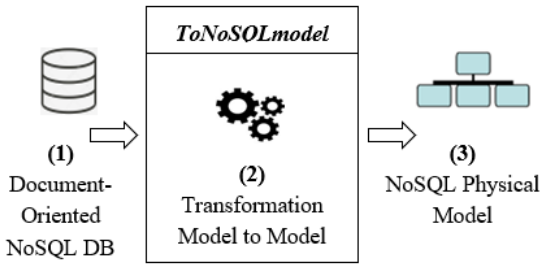

Figura 9: extracción dirigida por modelos del esquema de base de datos. Extraída de [36]

El origen de la información es una base de datos orientada a documentos, y el destino un modelo físico NoSQL. Se definen 5 reglas con notación QVT que se aplican en el proceso de transformación.

La evaluación se realizó utilizando un entorno de transformación de modelos denominado Eclipse Modeling Framework (EMF), que cuenta con un conjunto de plugins para generar modelos y otros resultados basados en dicho modelo. De Eclipse se utilizó Ecore (lenguaje para crear metamodelos) y QVT; y se utilizó el lenguaje OMG (Object Management Group) para especificar las transformaciones del modelo. Es necesario definir los metamodelos de entrada y salida en Eclipse, además de las reglas QVT.

6.2.2 Enfoque de Abdelhedi, F. ; Brahim, A. ; Ferhat, R. ; Zurfluh, G.

Se presenta como una continuación o ampliación de [36]. En [37] propone un enfoque automático de ingeniería inversa con el objetivo de aportar un elemento semántico cercano al entendimiento humano y transformar el modelo físico ya obtenido en [36] en un modelo conceptual. El proceso se denomina “ToConceptualModel”. Este proceso aplica un conjunto de transformaciones para realizar el paso del modelo físico NoSQL hacia un diagrama de clases UML.

En el recuadro rojo de la figura 10 se presenta este proceso.

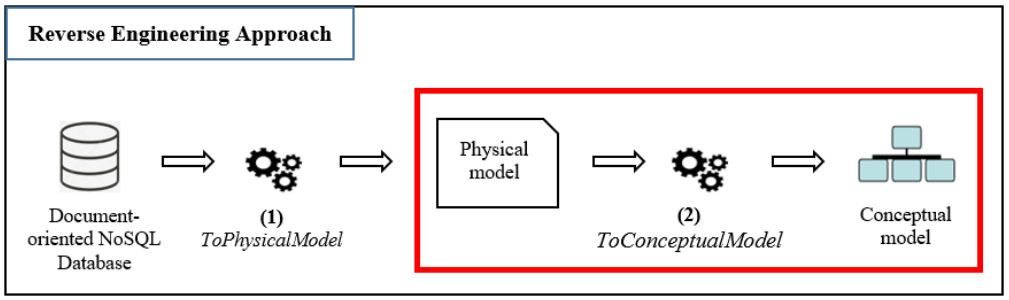

Figura 10: Visión general del proceso ToConceptualModel. Extraído de [37]

La correspondencia entre el modelo físico y el modelo conceptual se garantiza aplicando seis algoritmos de transformación.

Para implementar el proceso ToConceptualModel, se utilizó Eclipse Modeling Framework (EMF), que es un entorno adecuado para el modelado, metamodelado y transformación de modelos. El proceso se expresa como una secuencia de pasos que construyen el modelo resultante (diagrama de clases UML) :

1. Creación de un metamodelo de origen y otro de destino para representar los conceptos manejados por el proceso.
2. Creación de una instancia del metamodelo de origen.
3. Implementación de los algoritmos de transformación mediante el lenguaje QVT incluido en EMF.
4. Transformación ejecutando el script QVT creado en el paso 3. Este script toma como entrada el modelo de origen y devuelve como salida un diagrama de clases UML. El resultado se proporciona en forma de archivo XML.

6.2.3 Enfoque de C’anovasi, J. ; Jordi, C.

En [38] se presenta una herramienta denominada “JSON Discoverer” que permite mapear un documento JSON y obtener su esquema implícito a través de un diagrama UML [26].

JSON Discoverer tiene como objetivo asistir a los desarrolladores en las tareas que involucren inferir y visualizar el esquema implícito de los datos que poseen un documento JSON. Ofrecen tres funcionalidades principales, que pueden utilizarse por separado o encadenados:

1) Descubrimiento simple. Descubre el esquema de un conjunto dado de documentos JSON. Cuantos más sean, mejor, ya que algunas propiedades del modelo sólo pueden deducirse cuando hay varios ejemplos.
2) Descubridor avanzado. Toma la salida del descubridor simple para inferir su esquema global.
3) Compositor. Toma un conjunto de esquemas inferidos y busca enlaces de composición (es decir, conceptos o atributos comunes). Como resultado, se genera un gráfico de composición.

JSONDiscoverer dibuja la información del esquema como diagramas de clase UML, incluyendo conceptos (es decir, clases) y sus propiedades (es decir, atributos y asociaciones que vinculan los diferentes conceptos).

El trabajo se enfoca principalmente en brindar asistencia en el análisis de documentos JSON que son utilizados en los servicios brindados por las APIs Web [39] y no en la estructura de una Base de Datos NoSQL documental. No se detallan las reglas utilizadas por la herramienta para llegar al diagrama UML. En la figura 11 se muestra gráficamente los pasos que realiza JSONDiscover para obtener el diagrama, las principales funcionalidades de la herramienta se representan con cajas rellenas de negro mientras que los datos de entrada/salida se representan con cuadros rellenos de blanco.

En la figura 11 se muestra una captura de pantalla de la herramienta con la funcionalidad descubrimiento simple.

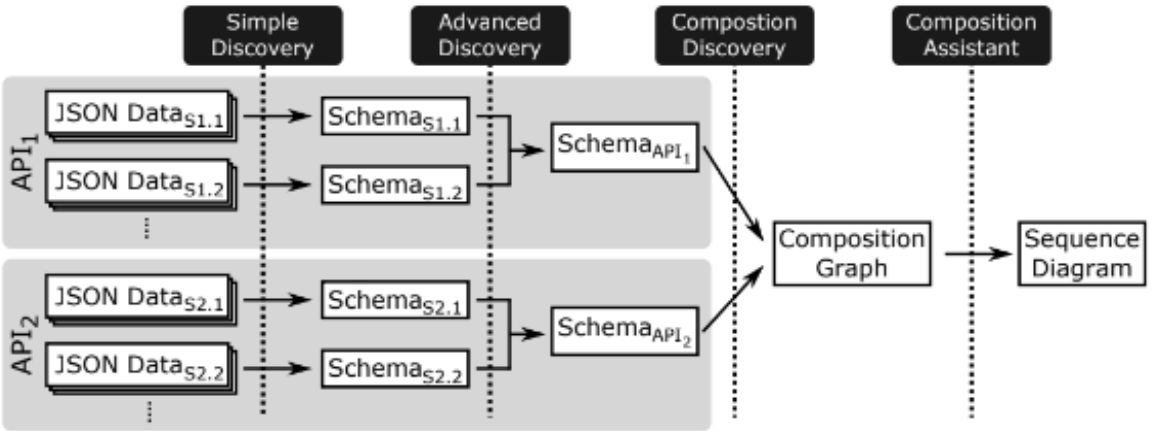

Figura 11: funcionalidades de JSONDiscover. Extraída de [38]

6.2.4 Enfoque de Klettke, M.; Storl, U.; Scherzinger, S.

En [40] se presenta un enfoque de extracción de esquemas adaptado para trabajar con documentos JSON mediante un algoritmo para la extracción de esquemas.
La figura 12 muestra los pasos del algoritmo.

Para declarar un esquema y validar los datos se necesita una descripción explícita del esquema. Para no definir un lenguaje propio, se utilizó un lenguaje de descripción de esquemas ya disponible: JSON Schema, que define restricciones estructurales sobre conjuntos de objetos JSON.

El método de extracción de esquema está relacionado con la extracción de DTD de documentos XML [40]. Una estructura de datos en forma de grafo resume la información estructural de todos los documentos persistentes. Los nodos representan propiedades JSON, objetos anidados o matrices, y las aristas capturan la estructura jerárquica de los documentos JSON. En particular, los nodos y las aristas están etiquetados con información que especifica en qué documentos JSON está presente la propiedad estructural.

El algoritmo de extracción de esquemas deriva un esquema explícito de una colección de documentos JSON. Si hay diferentes tipos de documentos JSON organizados en la misma colección, se preselecciona un subconjunto

La información del esquema de todos los documentos JSON de una colección o subconjunto de una colección se almacena en el llamado gráfico de identificación de la estructura.

En primer lugar, se construye el SG (grafo dirigido) a partir de los datos de entrada de la base de datos. Los nodos de los documentos JSON se recorren y se enumeran en preorden. Para cada nodo de los datos de entrada, se realiza una adición o extensión de un nodo en el SG y una adición o extensión de una arista desde el nodo a su nodo padre. Los números de los nodos especifican en qué orden se construye el SG.

Cuando se añade un nodo, se distinguen dos casos: si el nodo aún no existe en el SG, se añade el nodo. El nodo se almacena con un nombre de nodo y con una lista que contiene sólo una referencia: el docID del nodo actual y el número de nodo único i. Si el nodo ya existe en el SG, entonces la información del nodo actual se añade a la lista que se almacena con el nodo. La adición de una arista es similar. Si la arista aún no existe en el *SG*, se añade, de lo contrario la lista de aristas se actualiza añadiendo el docID y el número de nodo único del nodo padre.

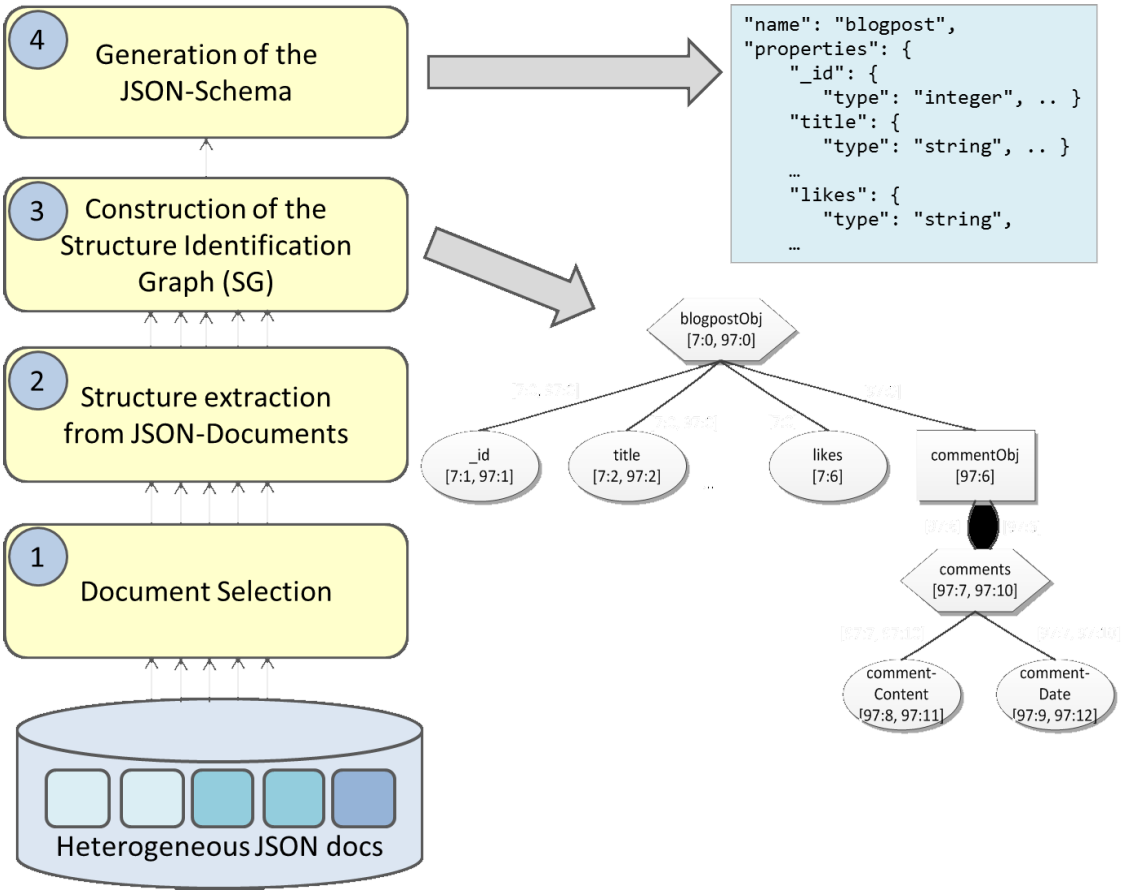

Figura 12: Pasos del algoritmo. Extraída de [40]

El SG se puede utilizar también para realizar estadísticas, encontrar valores atípicos en los datos y calcular medidas de regularidad. En la búsqueda de valores atípicos el algoritmo distingue dos casos:

a. estructuras que solo se dan en raras ocasiones
b. errores en los datos NoSQL

Decidir si un hallazgo es un error o no, requiere la interacción del usuario. En la figura 13 se muestra el esquema del algoritmo.

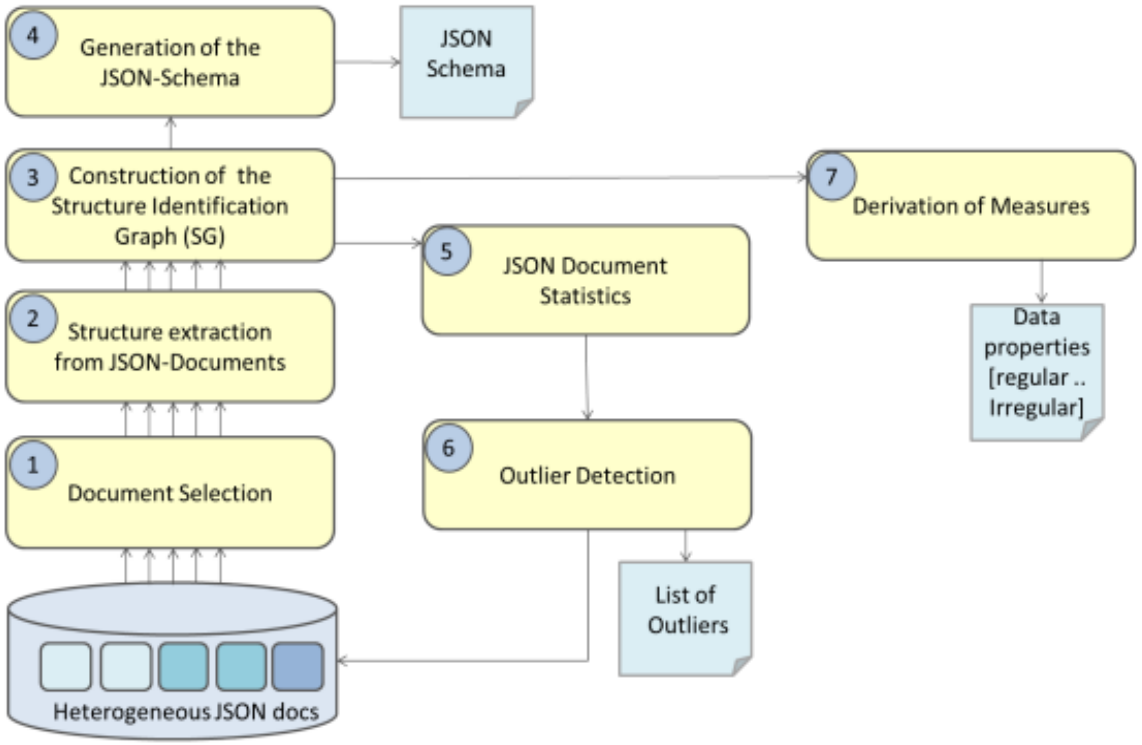

Figura 13: Extracción de esquemas, detección de valores atípicos y cálculo de medidas[40]

También se presenta un Gráfico de Identificación de Estructura Reducido (RG), esto es posible porque los documentos JSON tienen una estructura más simple que los documentos XML y por lo tanto no es necesario referenciar al JSON original en cada nodo.

6.2.5 Enfoque de Atzenia, P. ; Bugiottib, F. ; Cabibboa, L. ; Torlonea, R. (Documental)

En el enfoque [41] se plantean directrices de modelado para bases de datos NoSQL documental. Estas directrices abarcan tanto la etapa de diseño lógico como la de diseño físico. Cada una de ellas se desarrolla sobre conocimientos empíricos y están preparadas para ser intuitivas.

Inicialmente se presenta un diagrama conceptual que sigue las notaciones ERD (Diagrama de Relación de Entidades), aunque también se introduce UML, para estandarizar enfoques y notaciones. Los rectángulos se utilizan para mostrar entidades, las flechas corresponden a flujo de datos o conexiones entre entidades, y los corchetes y llaves para indicar atributos y matrices de clave.

Para mapear al esquema lógico y físico se plantean un conjunto de directrices que ofrecen recomendaciones sobre cómo desarrollar una base de datos NoSQL.

Existen 4 categorías de directrices, en la figura 14 se muestra gráficamente cómo se categoriza:

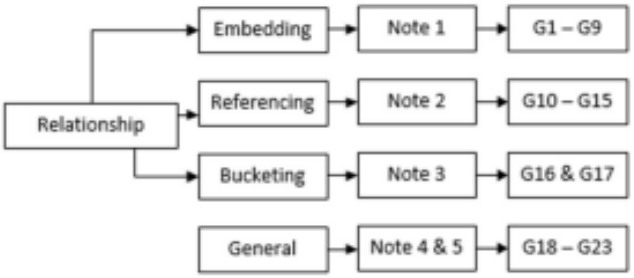

Figura 14: Categorización de directrices. Extraída de [41]

- Incrustación: se enuncian 9 directrices cuyo objetivo es responder preguntas relacionadas con la inserción de un documento en otro
- Referenciación: se enuncian 9 directrices cuyo objetivo es responder preguntas relacionadas con el proceso de conexión entre dos o más documentos mediante un identificador único
- Bucketing: son 2 directrices que se refieren a la división de documentos en tamaños más pequeños y manejables
- General: se enuncian directrices que no entran en ninguna de las categorías anteriores

También se propone un análisis para la priorización de directrices.

6.2.6 Enfoque de Rossel y Manna (Documental)

En el enfoque [42] se propone un nuevo modelo lógico para bases de datos NoSQL documentales. Dicho modelo se denomina DID (Diagrama de Interrelación de Documentos).

Al igual que en las bases de datos relacionales, se inicia realizando el modelo conceptual(ERD), con notación de entidades y relaciones de Chen [1]. Cómo los principios del diseño lógico relacional no son apropiados para las bases de datos documentales se propone un nuevo modelo lógico basado en consultas que puede implicar redundancia y desnormalización. La figura 15 muestra los pasos para modelar una base de datos NoSQL documental.

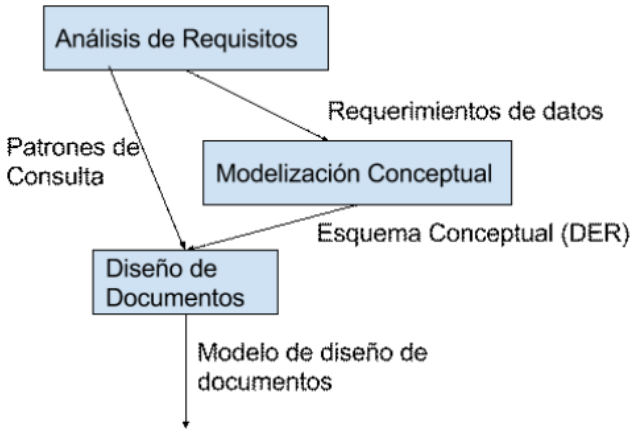

Figura 15: Pasos para modelar una base de datos documental. Extraída de [42]

Existen dos formas de relacionar a los documentos (unidad mínima de accesos en base de datos documentales), referenciar o embeber. Ésta decisión se toma en base a los patrones de accesos a los datos. Se deben analizar los distintos casos para mapear el DER al modelo lógico documental. La notación DID mantiene la notación DER agregando algunos elementos que se detallan a continuación:

- Relaciones 1 a 1: lo más directo es embeber un documento en otro. Para indicar que un documento se embebe dentro de otro se utiliza gráficamente una llave. La figura 16 muestra que el documento Gerente será embebido dentro del documento Departamento.

Figura 16: ejemplo de modelo lógico documental. Extraído de [42]

- Relaciones 1 a N: en este caso embeber el documento A en B, A se relaciona con solo un documento de B. Pero también es posible que B tenga una lista de documentos de A. No necesariamente se debe embeber, se puede referenciar, en dicho caso se marca gráficamente con una flecha entre la relación y la entidad como indica la figura 17.

Figura 17: ejemplo de modelo lógico documental. Extraído de [42]

- Relación N a M: Se puede embeber o referenciar desde un documento a otros o en ambos sentidos.
- Entidades débiles y atributos polivalentes: puede ser modelados embebiéndolos dentro del documento principal como se muestra en la figura 18.

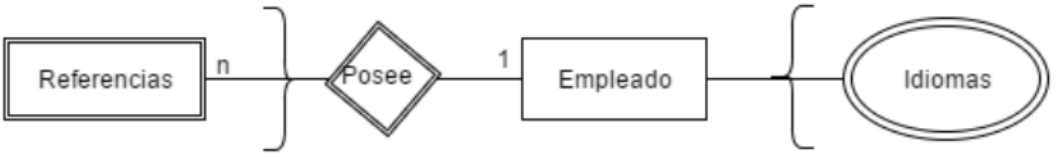

Figura 18: ejemplo de modelo lógico documental. Extraído de [42]

- Desnormalización: al embeber un documento se debe decidir si además tiene una existencia independiente, si la entidad se marca con una cruz, se entiende que solo se embebe y no tendrá existencia de forma independiente, caso contrario además de embeber el documento, existe de forma independiente. En la figura 19 se presenta un ejemplo de esta situación.

  Además de embeber o referenciar se puede utilizar una técnica intermedia, que es sólo embeber los tributos que se utilizan en las consultas; para representar esto se lista los atributos a embeber como muestra la figura 19.

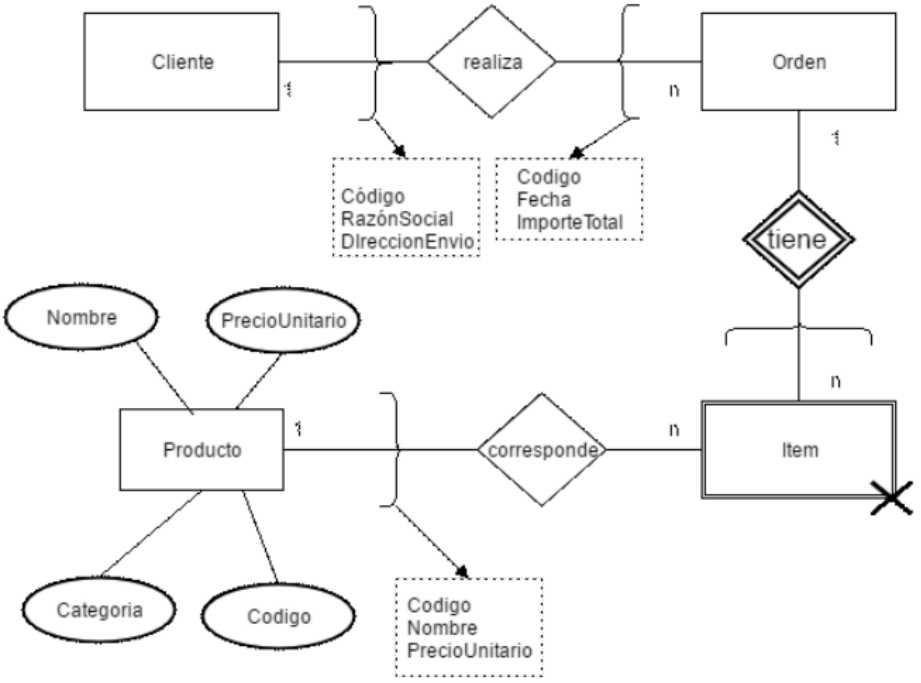

Figura 19: ejemplo de modelo lógico documental. Extraído de[42]

6.2.7 Enfoque de Varga V.; Jánosi-Rancz, K.; Balázs, K.;

En el enfoque [43] se basa en un fundamento matemático denominado FCA (Análisis Conceptual Formal), el cual es un método de análisis de datos que permite descubrir el conocimiento oculto de los datos. Los datos se representan mediante un contexto formal que contiene objetos y atributos, a partir de esto se generan conceptos formales agrupando objetos que tienen el mismo conjunto de atributos, produciendo la base para el posterior análisis de los datos.

Este enfoque propone un mapeo del modelo Entidad-Relación ampliamente conocido a un esquema semiestructurado; para esto se trabajan las relaciones uno a uno, uno a muchos y muchos a muchos.

Se consideran dos formas de convertir el esquema conceptual Entidad-Relación en un esquema para datos semiestructurados:

- Plana: cada entidad está a un mismo nivel y hace referencia a otra entidad mediante claves externas.
- Anidada: se incrustan las entidades dentro de otras tanto como sea posible.

Durante el diseño hay que analizar cuál enfoque utilizar. Ambos tienen ventajas y desventajas. En el enfoque de anidados la principal ventaja es que se puede recuperar toda la información en una sola consulta, pero como desventaja no se puede acceder a las entidades anidadas como independientes.

Primero se crea el contexto Objeto-Objeto de los datos, y se construye un entramado con dicho contexto para descubrir su conocimiento a fondo, y poder decidir si es factible el anidamiento o no aplicando el algoritmo de la figura 20.

Algorithm 1.
Input: a concept lattice
Output: nesting is possible or not
begin
if intermediate nodes are situated in one level **then**
  if intermediate nodes are labeled with utmost one key from E_1
   and utmost one key from E_2 **then**
    the relationship is One-to-One;
    if top node has labels **and** bottom node has labels **then**
      nesting is not possible;
    else
      nesting is possible;
    end;
  end;
  if intermediate nodes are labeled with more keys from E_1
   and utmost one key from E_2 **then**
    the relationship is Many-to-One;
    if top node has labels from E_1 **or** bottom node has labels from E_1 **then**
      nesting is not possible;
    else
      nesting is possible; // E_1 nested in E_2;
    end;
  end;
else
  the relationship is Many-to-Many;
  nesting is not possible;
end;
end;

Figura 20: algoritmo para determinar si se puede incrustar una entidad. Extraído de [43]

A continuación se analizan los tres tipos de relaciones para realizar el mapeo:

- Mapeo de relaciones uno a uno: lo más ventajoso sería embeber, pero la pregunta es cómo embeber, suponiendo dos entidad A y B, ¿se debe embeber A en B o B en A? Para ayudar a esta decisión se utiliza el entramado como el que se muestra en la figura 21, donde se muestran los 4 casos que pueden ocurrir en una relación uno a uno:

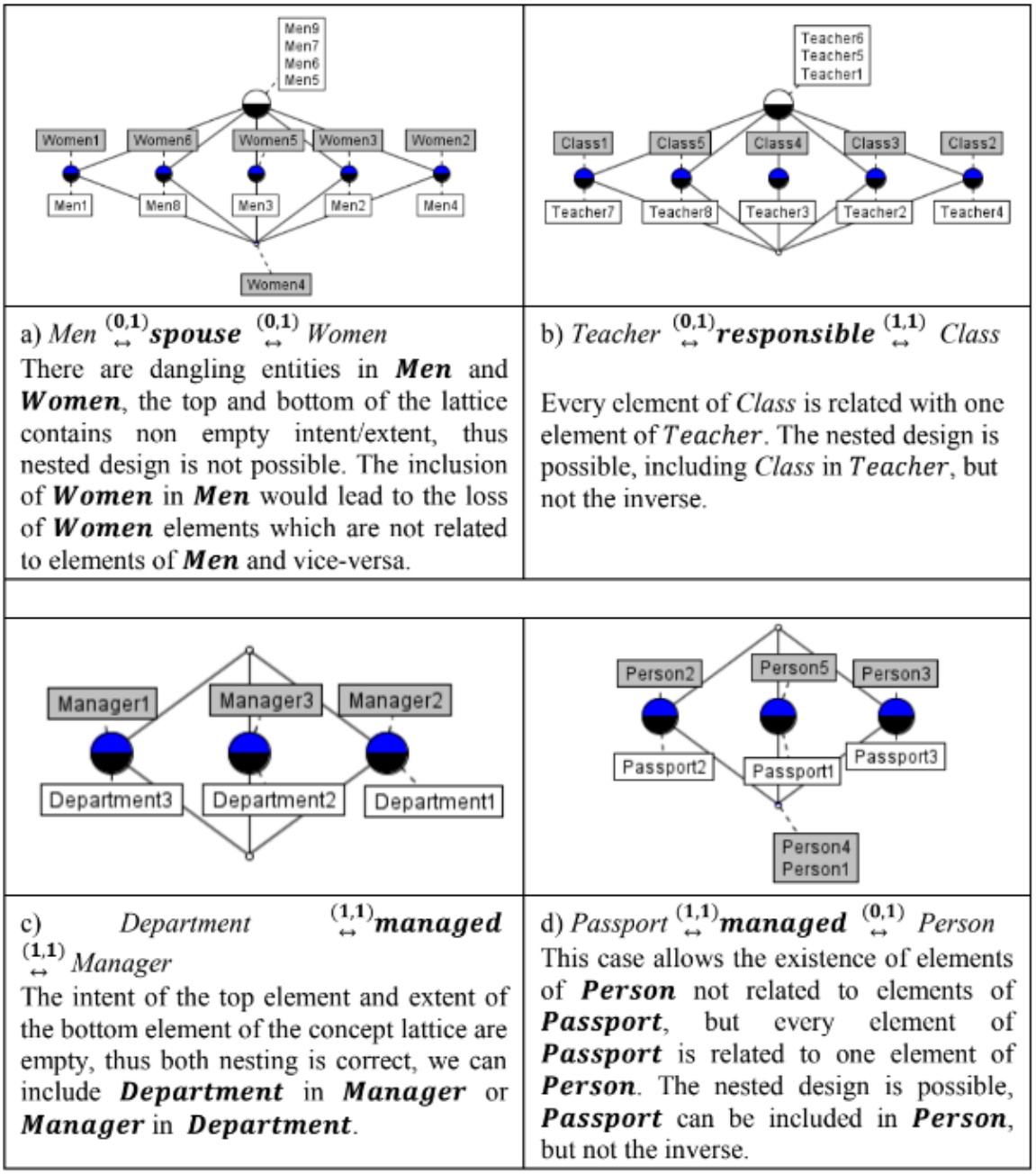

Figura 21: entramado del contexto Objeto-Objeto para relación uno a uno. Extraído de [43]

- En a, existen objetos Women y Men que no están relacionados entre sí, por lo tanto no es posible embeber, ya que sino se perderán objetos de la entidad embebida
- En b es posible embeber Class en Teacher, pero no a la viceversa, ya que existen Teacher que no están relacionadas con Class
- En c es posible embeber Departament dentro de Manager y viceversa ya que todos los elementos de Departament están relacionados con Manager y viceversa.

    - En d es posible embeber Pasaport dentro de Person, ya que existen elementos de Person que no están relacionados con Passport
- Relaciones Uno a Muchos
    - Lo más conveniente es incrustar la entidad del lado en que la cardinalidad es N, siempre y cuando esta entidad no sea accedida fuera del contexto de relación, en tal caso no es posible el anidamiento de las entidades.
- Relaciones Muchos a Muchos
    - En este tipos de relaciones se puede utilizar referenciación, simple o bidireccional, para el caso de una referenciación bidireccional hay que tener en cuenta que las actualizaciones puede ya no ser atómicas o puede requerir joins a nivel aplicación

6.3 Enfoques para bases de datos clave-valor

6.3.1 Enfoque de Rossel y Manna

En el enfoque [44] se propone un nuevo modelo lógico para bases de datos NoSQL clave-valor.

Se realiza un modelo conceptual en base a los requerimientos, puede ser un DER o una representación UML. A continuación se propone realizar una derivación a un modelo lógico, utilizando como base no solo el modelo conceptual sino también las consultas que se realizarán.

A partir del modelo conceptual y de los patrones de acceso, se especifican las entidades que se representan en la base de datos mediante el concepto de conjuntos de datos. El acceso a los valores se realiza mediante las claves. Para cada conjunto de datos se debe realizar un correcto diseño y selección de claves.

Una vez establecidos los conjuntos de datos iniciales, se deben definir las relaciones, indicando cómo la información obtenida de un conjunto de datos puede permitir el acceso a los elementos de otro. Pueden surgir nuevos conjuntos de datos cuya función sea simplemente organizar las claves y favorecer la navegabilidad de las aplicaciones que acceden a la base de datos. La Figura 22 muestra las etapas de la metodología propuesta.

Este enfoque propone un diseño lógico representado por un conjunto de datos interrelacionados, definidos como una 4-tupla:

dataset := (dn;ns; ks; v);

Donde:

- dn = nombre del dataset

- ns = espacio de nombres
- ks = lista ordenada de elementos que forman la clave
- v = estructura o tipo de valor

dn permite identificar el dataset y debe ser único; si corresponde a una entidad del modelo conceptual se utiliza el mismo nombre. El espacio de nombres puede ser una cadena o un valor nulo. En los motores de bases de datos que no soportan espacios de nombre, su valor pasa a formar parte de la clave. El conjunto de claves ks contiene la clave completa para acceder a un valor.

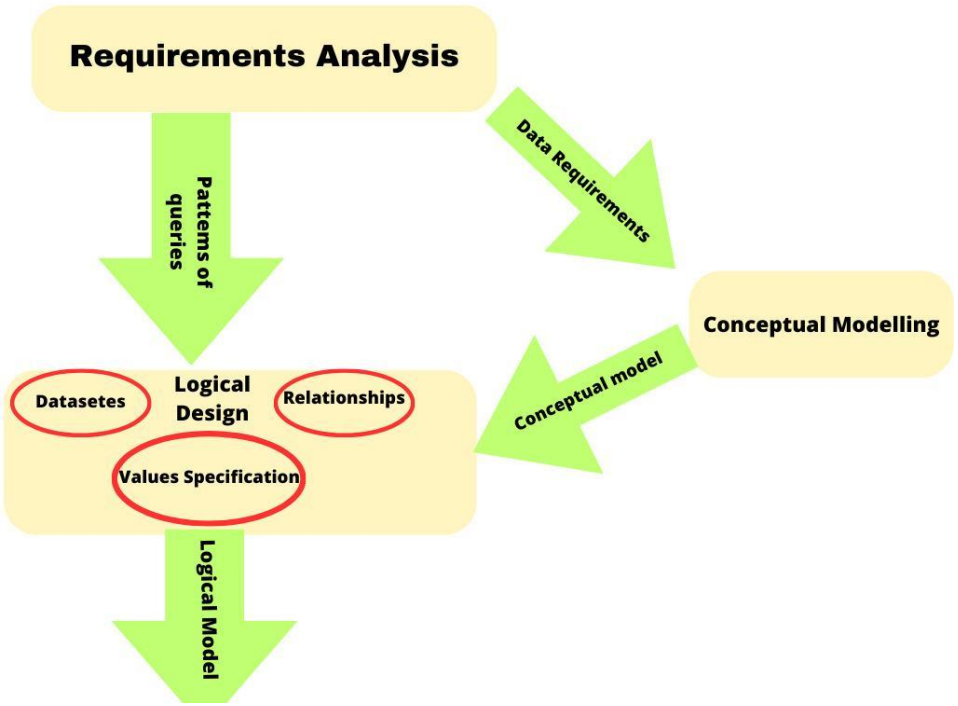

Figura 22: Etapas del diseño de base de datos clave-valor. Extraída de [44]

Una interrelación entre dos dataset Ds_1 y Ds_2, indica que un elemento de e_1, que pertenece a Ds_1, (un par clave-valor o elemento) está relacionado con un elemento e_2 que pertenece a Ds_2, y se define del siguiente modo:

- $Ds_1 := (dn_1;ns_1; ks_1; v_1)$
- $Ds_2 := (dn_2;ns_2; ks_2; v_2)$
- rel := $((Ds_1,Ds_2)$, lstVinc)
- lstVinc := $(k_2|(k_1, k_2))^*$

Donde:

$k_1 € ks_1$ and $k_2 € ks_2$

El par ordenado (Ds_1, Ds_2) indica la dirección de la navegabilidad desde Ds_1 hacia Ds_2 . La lista de enlaces lstVinc define la interrelación, a partir de un elemento concreto e_1 que pertenece a Ds_1, la información para ensamblar la clave de acceso a un elemento

e_2 que pertenece a Ds_2. Si sólo hay una clave en la lista entonces su valor se obtiene a partir del valor de e_1.

6.4 Enfoques para bases de datos orientadas grafos

6.4.1 Enfoque de Comyn-Wattiau, I. ; Akoka, J.

En [45] se presenta un enfoque basado en una arquitectura dirigida por modelos (Model-Driven Architecture o MDA) [46] que permite lograr una ingeniería inversa de las bases de datos orientadas a grafos obteniendo como resultado un esquema Entidad-Relación Extendida (EER), para el motor de base de datos orientado a grafos Neo4j.

Partiendo de un código escrito en Cypher (lenguaje utilizado por Neo4j) y utilizando reglas de transformación, se deriva a un modelo de grafos lógico y un esquema EER.

Los pasos que implementan son los siguientes:

1. Tomar los códigos Cypher que permitieron la generación de la base de datos Neo4j.
2. Analizar el código y aplicar reglas de transformación para obtener un grafo lógico.
3. Mapear el grafo lógico a un esquema conceptual EER utilizando reglas de transformación.
4. Completar el esquema con las cardinalidades.

En este enfoque se presenta un conjunto de reglas que se utilizan para crear el grafo lógico a partir de las sentencias create Cyper. Básicamente analizan las sentencias Cyper y de acuerdo a reglas propuestas se crea un nodo o una arista con sus respectivas propiedades. En la figura 23 se muestra un ejemplo del gráfico lógico que se propone.

A partir del esquema lógico representado por un grafo, aplicando un conjunto de reglas se realiza la transformación a un esquema conceptual.

Una vez que se obtiene el modelo conceptual, se complementan por defecto las cardinalidades en 1, se analizan las instancias de cada entidad dentro de la base de datos en relación a la otra entidad, y se cambia de 1 a N cuando la ocurrencia sea mayor o igual a 2.

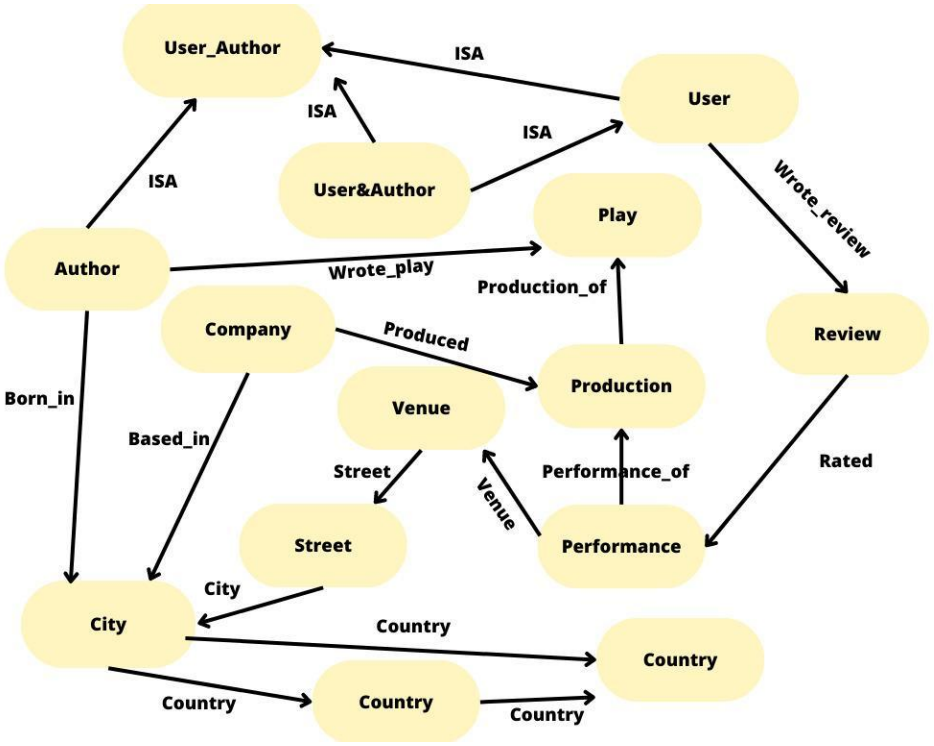

Figura 23:ejemplo de grafo lógico. Extraída de [45]

6.4.2 Enfoque de De Virgilio, R; Maccioni, A; Torlone, R.

En [47] se propone una metodología basada en un nuevo modelo lógico para bases de datos orientadas a grafos.

Este enfoque parte de un modelo conceptual, específicamente de un DER, se traduce a un grafo en el que las entidades y las relaciones se agrupan de acuerdo a las restricciones definidas en el DER. Tiene como objetivo minimizar la cantidad de operaciones necesarias para recuperar datos relacionados. La traducción es completamente automática.

La estrategia de diseño tiene 3 fases:

- Generación de un diagrama ER orientado: se convierte el diagrama ER que es un grafo no dirigido en un grafo dirigido, denominado ER orientado (O-ER), y con una función se asigna el peso (w) de cada arista. El O-ER se genera aplicando a un diagrama ER las siguientes reglas de transformación:
    - Una relación uno a uno se convierte en una arista dirigida doble “e” tal que w(e) = 0, es decir su peso es cero.
    - Una relación uno a muchos se convierte en una arista unidireccional “e” tal que w(e) = 1, que va desde la entidad con menor multiplicidad hacia la entidad con mayor multiplicidad.
    - Una relación mucho a muchos se convierte en una arista “e” de doble dirección donde w(e)=2.

  En la figura 25 se puede ver un ejemplo de O-ER después de aplicar las reglas mencionadas al diagrama ER que se muestra en la figura 24.

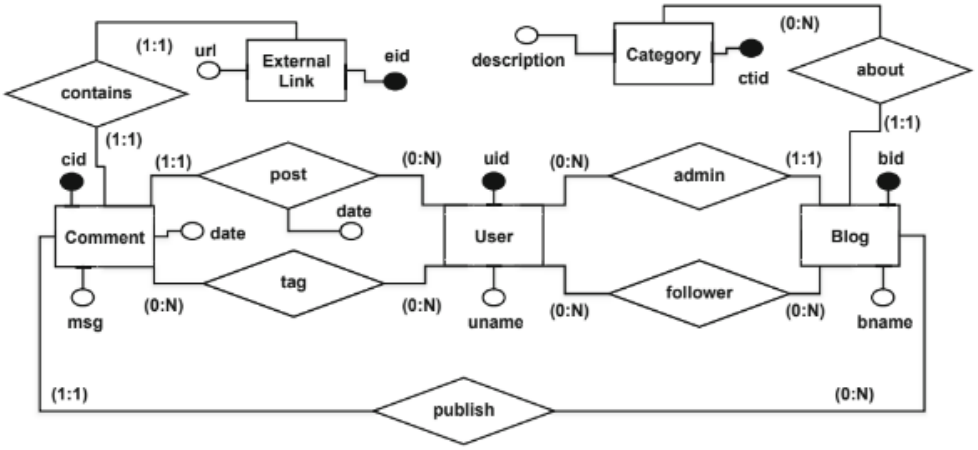

Figura 24: diagrama ER. Extraído de [47]

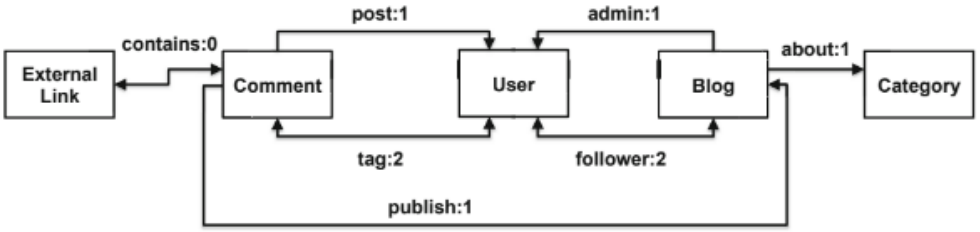

Figura 25: diagrama O-ER. Extraído de [47]

- Partición de los elementos del diagrama del punto anterior: se agrupan los elementos. Para llevar a cabo la agrupación, se calculan los pesos para los nodos y se aplican un conjunto de reglas. El cálculo del peso de los nodos se divide en $w^+(n)$ y $w^-(n)$, que corresponde a la suma de los pesos de las aristas de salientes y entrantes. Teniendo en cuenta esto, se aplican las siguientes reglas:
    - Si un nodo está desconectado, el grupo es formado por sólo ese nodo.
    - Si un nodo tiene $w^-(n) > 1$ y $w^+(n) >= 1$, entonces n forma un grupo por sí sólo.
    - Si un nodo tiene $w^-(n) <= 1$ y $w^+(n) <= 1$, entonces n se añade a un grupo de un nodo m tal que exista una arista (m,n).

  En la figura 26 se muestra un ejemplo del diagrama de partición.

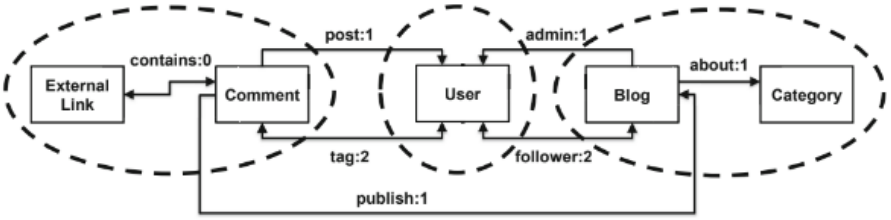

Figura 26: diagrama de partición. Extraído de [47]

- Definición de una plantilla sobre la partición resultante: la plantilla funciona como esquema para una base de datos gráfica en la que cada nodo y cada arista contiene las propiedades implicadas. Los nombres de las propiedades provienen de las entidades que se encuentran dentro del mismo grupo, en general al nombre del atributo se le antepone el nombre de la entidad. En la figura 27 se puede ver un ejemplo de una plantilla.

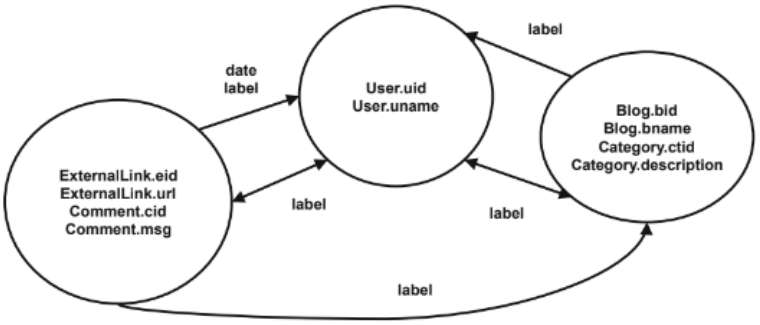

Figura 27: Ejemplo de plantilla. Extraído de [47]

6.4.3 Enfoque de Pokorný, J.

En [48] propone un metodología para modelar bases de datos orientada a grafos, este enfoque se basa en grafos de propiedades [49].

Se propone un variante binaria del modelo conceptual E-R, con tipos de entidades fuertes, débiles, tipos de relaciones, atributos, claves de identificación, claves de identificación parciales, jerarquías y restricciones de cardinalidad mínima y máxima.

Tanto los tipos de entidades como las relaciones pueden tener atributos. Pueden expresarse restricciones de cardinalidad mínimas y máximas, mediante la notación pata de gallo para el nodo inicial y final, una línea recta y otra discontinua representan una relación obligatoria y opcional respectivamente.

Los tipos de entidades débiles dependen de la existencia e identificación de otro tipo de entidad. Por ejemplo en la figura 28, Street es un tipo de entidad débil, su identificación parcial es name_street, dentro de la base de datos la clave sería (name_town, name_street), la línea perpendicular indica la dependencia de identificación y existencia.

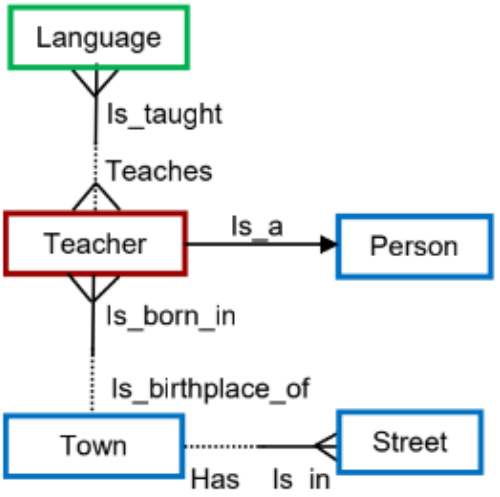

Figura 28: ejemplo 1 de modelo conceptual de grafo. Extraído de [48]

También es posible representar las jerarquías, como se muestra en la figura 11, donde Teacher hereda de Person. Teacher tiene su propia clave de identificación, la clave de supertipo puede ser útil para simplificar consultas de la base de datos.

Podría existir el caso en que el tipo de entidad depende de la identificación de dos o más tipos de entidades, por ejemplo solicitudes de préstamos que se identifica con la fecha (date) mas el identificador de dos personas, el marido y su mujer (man, woman), esto se puede representar como se muestra en la figura 29, que es más expresivo o de forma simplificada como se ve en la figura 30.

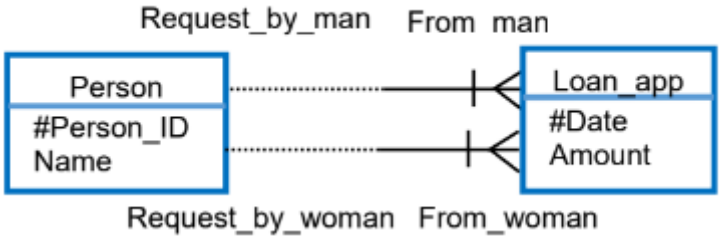

Figura 29:ejemplo 2 de modelo conceptual de grafo. Extraído de [48]

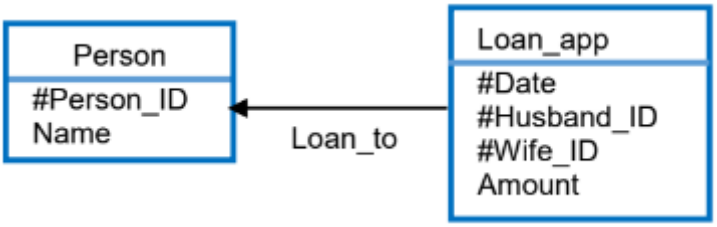

Figura 30: ejemplo 3 de modelo conceptual de grafo. Extraído de [48]

Para mapear el modelo conceptual presentado a un esquema de base de datos de grafos de definen 5 reglas:

1. Para cada tipo de entidad fuerte se crea un tipo de nodo que incluya todos sus atributos.

2. Para cada tipo de entidad débil se crea un tipo de nodo donde la clave resultante es la clave parcial de la entidad débil más la clave de la entidad fuerte.
3. Para cada tipo de relación se crea un tipo de borde que incluye todos los atributos.
4. Las restricciones de cardinalidad máxima y mínima se convierten como restricciones explícitas de la base.
5. Las jerarquías se transforman directamente como las relaciones. Se recomienda propagar la clave de la entidad padre a las entidades hijas.

6.5 Enfoques para bases de datos familia de columnas

6.5.1 Enfoque de Chebotko, A. ; Kashlev, A. ; Lu, S.

En [50] se propone una metodología de modelado de datos basado en consultas para el motor de base de datos Cassandra.

No sólo se tiene en cuenta la información a almacenar sino también las consultas a realizar para el modelado de la base de datos.

En primer lugar se realiza un esquema conceptual con un Diagrama Entidad Relación en la notación de Chen [1], ya que es independiente de la tecnología que se vaya a utilizar.

Una vez realizado el modelado conceptual se lo mapea a un modelo lógico basado en las consultas definidas en el flujo de trabajo de la aplicación. El esquema lógico corresponde a un esquema de base de datos Cassandra, con tablas, columnas, claves primarias, claves de partición y claves de agrupación. En la figura 31 puede verse gráficamente los pasos de este proceso de diseño.

Figura 31: pasos del proceso de diseño propuesto. Extraída de [50]

El mapeo al esquema lógico está basado en principios de modelados de datos, reglas de mapeo y patrones de mapeo.

Se enuncian cuatro principios de modelado de datos:

1. El primero es la comprensión de los datos que se capta con el modelado conceptual de los datos. Los tipos de entidad, atributos y relaciones, no solo define los datos a almacenarse sino también las propiedades, como los identificadores de las entidades y los tipos de relaciones.

2. El segundo principio se basa en las consultas que se obtienen a través de un modelo de flujo de trabajo de la aplicación. Al igual que los datos, las consultas afectan directamente a las tablas. Además de considerar las consultas, también se debe tener en cuenta la ruta de acceso de cada consulta para organizar los datos de forma eficiente. Existen tres vías de acceso:
    a. partición por consulta
    b. partición + consulta
    c. tabla o tabla + por consulta

    La opción a (partición por consulta) es la más eficiente, ya que sólo recupera una fila, un conjunto o todas las filas de una partición en una consulta.

    Las rutas b y c se refieren a la recuperación de los datos de unas pocas particiones de una tabla o muchas particiones de una tabla o más. Para conseguir un rendimiento óptimo deben evitarse.

3. La tercera clave es el anidamiento de datos, es decir, organizar múltiples entidades juntas basándose en un criterio conocido. El criterio puede ser un atributo que contenga el mismo valor para todas las entidades anidadas. Para las entidades anidadas existen dos mecanismos, las particiones de varias filas o colecciones; esta metodología se basa principalmente en la primera opción.

4. Duplicación de los datos. La duplicación de los datos en Cassandra, en distintas tablas, particiones y filas es común para soportar eficientemente distintas consultas sobre los mismos datos. Es mejor duplicar datos para tener un acceso partición por consulta que unir datos de múltiples particiones y/o tablas.

Se definen 5 normas de asignación para mapear un esquema conceptual en un esquema lógico tomando en cuenta las consultas. Estas normas se resumen de la siguiente forma:

1. Los tipos de entidades y relaciones se asignan a tablas, mientras que las entidades y relaciones se asignan a filas de las tablas. Los tipos de atributos deben conservarse como columnas de la tabla.
2. Los atributos de búsqueda de igualdad que se utilizan en el predicado de una consulta, se asignan a las columnas del prefijo de una clave primaria de la tabla. Estas columnas deben contener todas las columnas de partición y, opcionalmente, una o más columnas de la clave de agrupación.
3. Un atributo de búsqueda de desigualdades, que se utiliza en el predicado de una consulta, se asigna a una columna de la clave de agrupación de la tabla.
4. Los atributos de ordenación, que se especifican en una consulta, se asignan a las columnas de la clave de agrupación con un orden de agrupación ascendente o descendente.
5. Los tipos de atributos claves se corresponden con las columnas de la clave primaria.

Para diseñar un esquema de tabla, es importante aplicar las reglas enunciadas en el contexto de una consulta concreta y de un subgrafo del modelo conceptual del que se ocupa la consulta. Las reglas deben aplicarse en el orden que se listan.

Resulta útil representar gráficamente los diseños de modelos de datos lógico y físico, para ello se propone un nueva técnica llamada “Diagrama de Chebotko”. Este diagrama presenta el diseño de un esquema de una base de datos como una combinación de esquemas de tablas individuales y transición de flujos de trabajo de aplicaciones basadas en consultas.

Además, para automatizar la metodología presentada se desarrolló una herramienta llamada KMD [50]. Esta herramienta se basa en las reglas definidas para automatizar las tareas de modelado más complejas: el mapeo del modelo conceptual al lógico, mapeo del modelo lógico al físico y generación de CQL (lenguaje de consulta Cassandra).

6.5.2 Enfoque de Mior, M.; Salem, K.; Aboulnaga, A.; Liu, R.

En [51] presenta un asesor de esquemas para bases de datos NoSQL del tipo familia de columnas.

El asesor toma dos datos de entrada. Un modelo conceptual de datos (la figura 32 se muestra un ejemplo de este modelo) y la carga de trabajo, es decir las consultas y actualizaciones que se esperan resolver. En la carga de trabajo se describe el conjunto de consultas y actualizaciones junto a un peso que indica la frecuencia relativa de cada trabajo. En la figura 33 se muestra un ejemplo de una consulta de la carga de trabajo. Dichas consultas se expresan sobre el modelo conceptual en una notación similar a la de SQL.

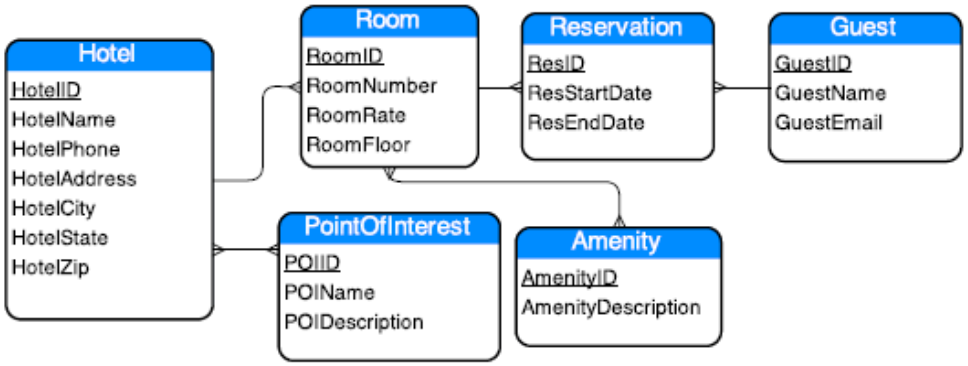

Figura 32: ejemplo de modelo conceptual de datos. Entrada del asesor. Extraída de [51]

```
SELECT Guest.GuestName, Guest.GuestEmail FROM
Guest WHERE Guest.Reservation.Room.Hotel
  .HotelCity = ?city AND
Guest.Reservation.Room.RoomRate > ?rate
```

Figura 33: ejemplo de carga de trabajo. Entrada del asesor. Extraída de [51]

Con los datos mencionados, el asesor recomienda un esquema de base de datos que describe las familias de columnas que debería de tener el esquema físico y un conjunto de planes de trabajo. Una familia de columna se identifica por su nombre, las claves de partición, claves de agrupación y valores de columna. El plan de trabajo describe cómo implementar cada consulta o actualización de la carga de entrada.

El asesor realiza cuatros pasos para la recomendación de esquemas como se muestra en la figura 34:

1. Enumeración de candidatos: el asesor genera un conjunto de familia de columnas candidatas que sean capaces de responder a las distintas consultas de la carga de trabajo. Esta se realiza en dos pasos:
    a. Se enumeran las familias de columnas candidatas para cada consulta de la carga de trabajo de la aplicación.
    b. Se completa el conjunto inicial con familias de columnas adicionales construidas combinando candidatos del conjunto inicial. El objetivo es añadir familias de columnas que puedan ser útiles para responder a más de una consulta y que consuman menos espacio que dos familias de columnas separadas.

    Puede ocurrir que para las actualizaciones se necesiten familias de columnas adicionales.

2. Planificación de consultas: el asesor genera todos los posibles planes de trabajo para cada consulta, utilizando todas las familias de columnas del paso anterior. El resultado de este proceso se denomina espacio de planes para la consulta en cuestión. En el paso 3, durante la optimización del esquema, el asesor de esquemas utilizará los espacios de planes de cada consulta para determinar cuál de las familias de columnas candidatas debe recomendar.
3. Optimización del esquema: el optimizador de esquemas elige un conjunto de familia de columnas que minimiza el costo de responder las consultas.
4. Recomendación del plan: de acuerdo a las familias de columnas recomendadas en el paso anterior se selecciona un plan de trabajo de los generados en el punto 2.

Se implementa un prototipo para el motor de bases de datos Cassandra, NoSE (NoSQL Schema Evaluator)

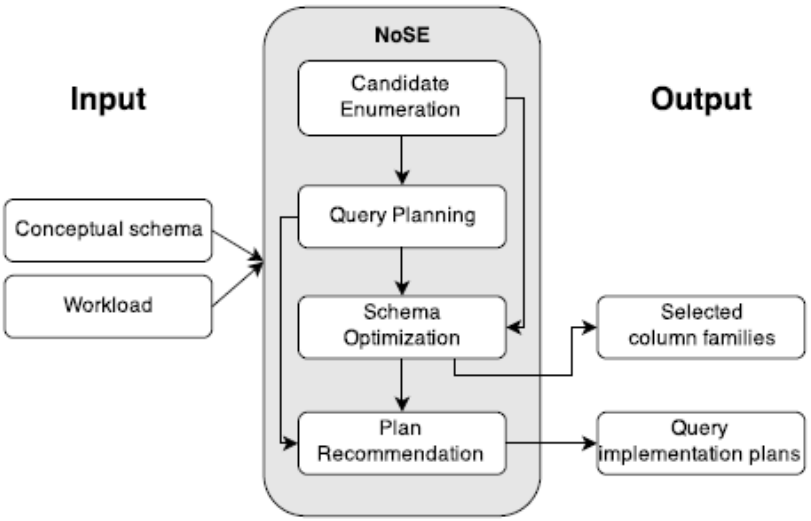

Figura 34: arquitectura del sistema asesor. Extraída de [51]

6.6 Enfoques aplicables a más de un tipo de base de datos

6.6.1 Enfoque de De La Vega, A.; García-Saiz, D. ; Blanco, C.; Zorrilla, M.; Sánchez, P.

El enfoque [52] describe un proceso de diseño de bases de datos NoSQL basado en modelos, en donde, a partir de un modelo conceptual de datos, independiente de cualquier tipo de bases de datos, se genera de forma autónoma una posible implementación para un sistema de bases de datos NoSQL concreto. Además, este proceso permite configurar el diseño final según las necesidades de cada contexto.

Este proceso necesita operar con modelos bien definidos, en particular, necesita como entrada un metamodelo en el cual se especifican el modelo conceptual de datos y las consultas que recuperarán y actualizarán la información representada. Además, permite realizar ciertas anotaciones generales, por ejemplo, la tasa de actualización que posiblemente tendrá una entidad. Este metamodelo es denominado Metamodelo de Datos Genéricos (GDM) y describe sus componentes (entidades, relaciones o referencias y consultas) mediante una notación textual que posee su propia sintaxis.

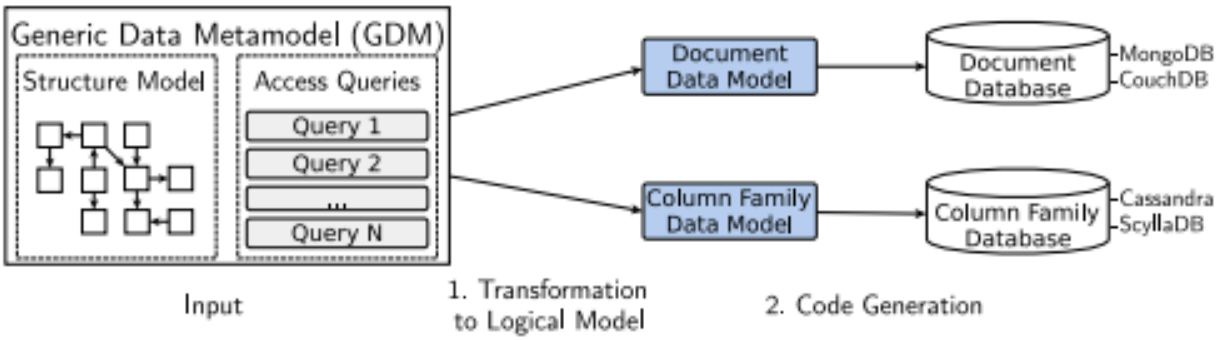

Figura 35: Proceso de transformación Mortadelo. Extraída de [52]

En este proceso, a partir de un caso de estudio representado a través de un GDM, se plantean modelos lógicos para dos categorías de almacenamiento no estructurado de datos, familia de columnas, en donde se utiliza el motor de base de datos Cassandra [22] y documental, en donde se utiliza el motor de base de datos MongoDB [20]. Para ello, se aplica un conjunto de reglas y algoritmos predefinidos para transformar una instancia de un modelo conceptual en un modelo lógico concreto para un tipo de base de datos NoSQL. En la figura 35 se muestra gráficamente el proceso de transformación.

Finalmente, mediante otro conjunto de reglas, generan transformaciones de código concretas para dos motores de bases de datos, Cassandra (almacenamiento en familia de columnas) y MongoDB (almacenamiento documental), es decir, se genera el esquema físico para un motor de base de datos NoSQL específico.

6.6.2 Enfoque de Atzenia, P. ; Bugiottib, F. ; Cabibboa, L. ; Torlonea, R.

El enfoque [53] propone un proceso de diseño que tiene una etapa conceptual, una etapa lógica, la cual es independiente del tipo de base de datos, y una etapa final que considera las características específicas de un motor de base de datos NoSQL. Este enfoque se basa en las siguientes actividades principales:

1. Modelado conceptual de datos a partir del Diseño Dirigido al Dominio (DDD) que da como resultado un diagrama UML.
2. Sobre el diagrama UML del punto anterior se identifican los agregados. Un agregado es un grupo de objetos relacionados, que representa una unidad de acceso y de manipulación atómica.
3. Implementación del modelo NoAM (NoSQL Abstract Model) a partir de la identificación de agregados.

El proceso comienza con el diseño de bases de datos, construyendo una representación conceptual de los datos de interés, en términos de entidades, relaciones y atributos. Luego, se identifican las agregaciones. Esta actividad puede ser impulsada por los patrones de acceso a los datos, así como por las necesidades de escalabilidad y consistencia. Específicamente, los agregados deben diseñarse como las unidades en las que se debe garantizar la atomicidad. Cada agregado debe ser lo suficientemente grande como para incluir todos los datos requeridos por una operación de acceso a datos relevantes. Por otro lado, los agregados deben ser lo más pequeños posible. Los agregados pequeños reducen las colisiones de concurrencia y cumplen con los requisitos de rendimiento y escalabilidad. En la figura 36 se muestra un ejemplo gráfico de un diseño de agregados.

En este enfoque, se utiliza NoAM como un modelo intermedio entre los agregados y las bases de datos NoSQL. En NoAM, la unidad de acceso y distribución de datos se modela mediante un bloque, el cual representa una unidad de datos máxima para la que se otorgan operaciones de acceso atómicas, eficientes y escalables. En la figura 37, se puede ver un ejemplo de modelo NoAM. Los sistemas NoSQL brindan operaciones eficientes, escalables y consistentes en bloques y, a su vez, esta elección propaga tales cualidades a las operaciones en agregados.

Finalmente, se analiza cómo se puede implementar una representación de datos NoAM en un motor de base de datos NoSQL específico. Por ejemplo, para MongoDB (base de datos NoSQL documental), una implementación natural para una base de datos NoAM se puede basar en una colección MongoDB distinta para cada colección de bloques y un único documento principal para cada bloque.

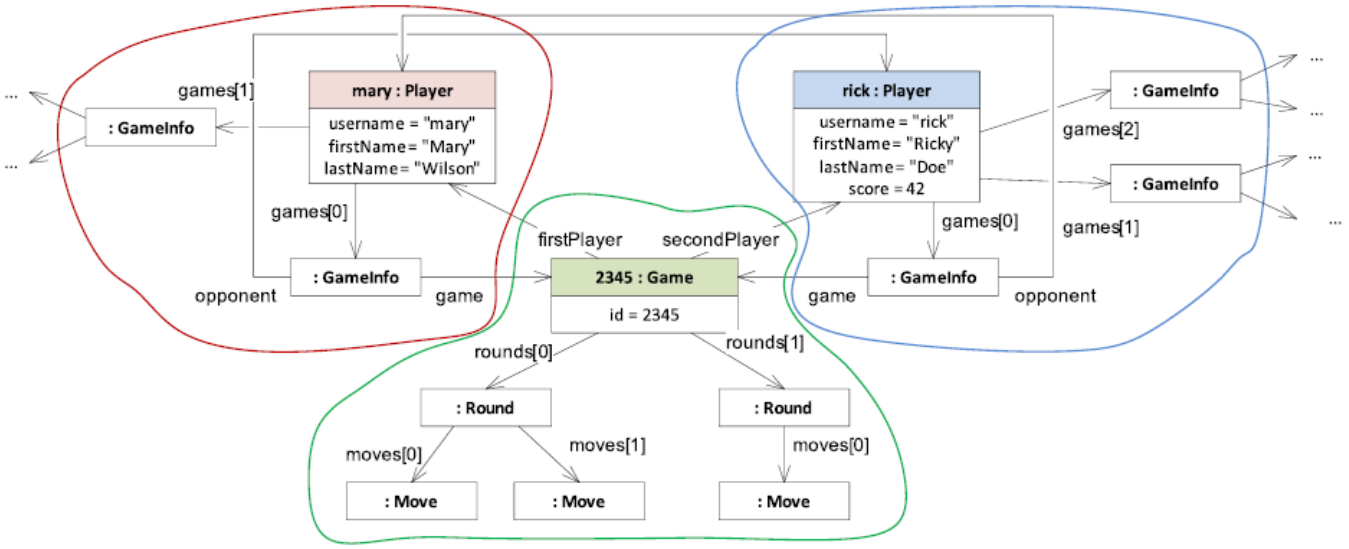

Figura 36: ejemplo de diseño de agregados. Extraído de [53]

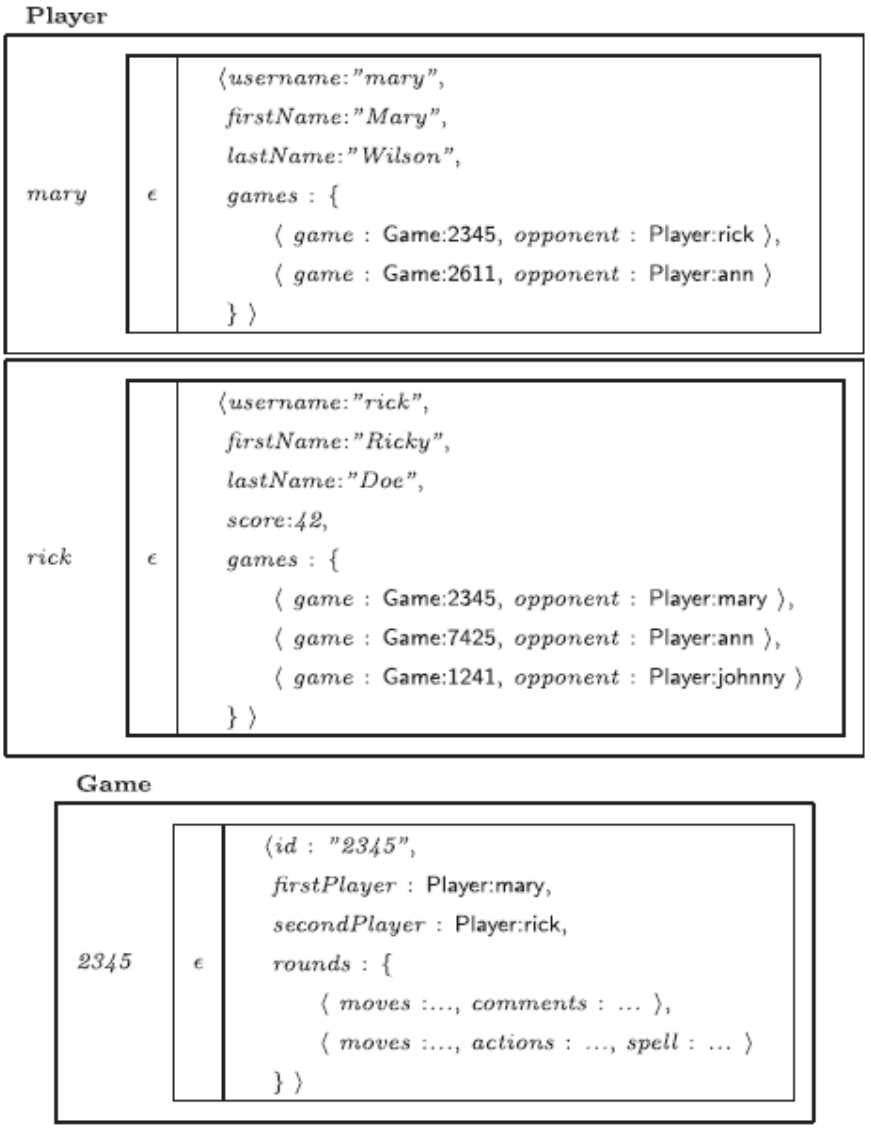

Figura 37: ejemplo de modelo NoAM. Extraído de [53]

6.6.3 Enfoque de Banerjee, S. ; Sarkar, A.

El enfoque [54, 55] propone un modelo de nivel conceptual común para varios tipos de bases de datos NoSQL y un lenguaje de especificación de datos NoSQL para representar un modelo de datos de nivel lógico, independiente de cualquier representación a nivel físico. Además, se han propuesto, formalizado e ilustrado distintas reglas de validación respecto al modelo conceptual mediante la evolución de un caso de estudio.

Este modelo conceptual posee un conjunto común de construcciones, relaciones y una serie de propiedades significativas de las relaciones para unificar las representaciones de nivel conceptual de diferentes bases de datos NoSQL. Este modelo consta de tres capas interrelacionadas: Colección, Familia y Atributo. La capa de atributos es la capa base del modelo conceptual y los tipos de construcción AT son los grupos de todos los mismos tipos posibles de instancias y de naturaleza elemental. En la figura 38 se ve la descripción de la organización por capas del modelo conceptual.

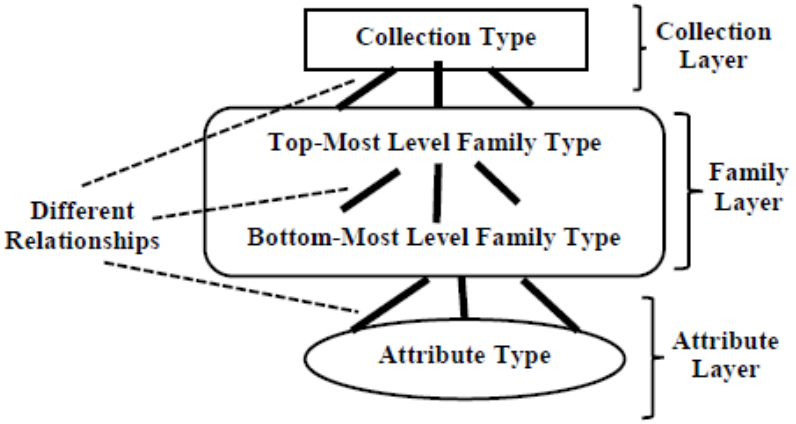

Figura 38: Modelo conceptual por capas. Extraída de [54]

La capa de familia es la capa intermedia del modelo conceptual y puede contener numerosos tipos de construcciones FA. Varios atributos relacionados semánticamente se agrupan para formar el tipo de construcción FA de capa de familia de nivel más bajo. Esta capa se puede descomponer en múltiples niveles según las preferencias de los diseñadores.

La capa colección es la capa superior del modelo conceptual. Las familias de la capa superior relacionadas semánticamente se ensamblan para formar una columna.

Desde un nivel superior, la base de datos se puede ver como un conjunto de columnas.

Las construcciones de este modelo se conectan entre sí mediante relaciones distintas. Estas relaciones pueden ser de dos tipos: relación de tipo entre capas y la relación de tipo dentro de la capa. Estas relaciones poseen varias propiedades, como

multiplicidad, orden, modalidad, disponibilidad, participación condicional y consistencia.

Se propone un lenguaje de especificación que permite transformar un modelo conceptual de datos en un modelo lógico y posteriormente a su correspondiente modelo físico para un motor de base de datos NoSQL.

Finalmente, se plantea un conjunto de reglas de validación para el modelo NoSQL obtenido, estas reglas se dividen en tres grupos: para validación estructural, para validación de restricciones y para validación de consistencia.

6.6.4 Enfoque de Morales S. ; Garcia, J. ; Ruiz, S.

En [56, 57] se presenta un enfoque de ingeniería inversa para la inferencia de un modelo UML a partir de datos almacenados en una base de datos. Es aplicable a los tipos de bases orientadas a documentos, clave-valor y orientadas a columnas, fue validado para los motores de bases de datos documentales MongoDB y CouchDB. Se realiza una transformación modelo a modelo, partiendo del modelo físico de la base de datos representado en formato JSON, y dando como resultado de la transformación un modelo UML orientado a agregación. El término “agregado”, se utiliza para referirse a una entidad raíz que agrega datos recursivamente a otra entidad.

Agregación y referencia son los dos tipos de asociaciones utilizadas en los modelos de datos. En la figura 39 se muestra un ejemplo donde se tiene agregación y referencia, los libros agregan a sus autores y referencia a la editorial.

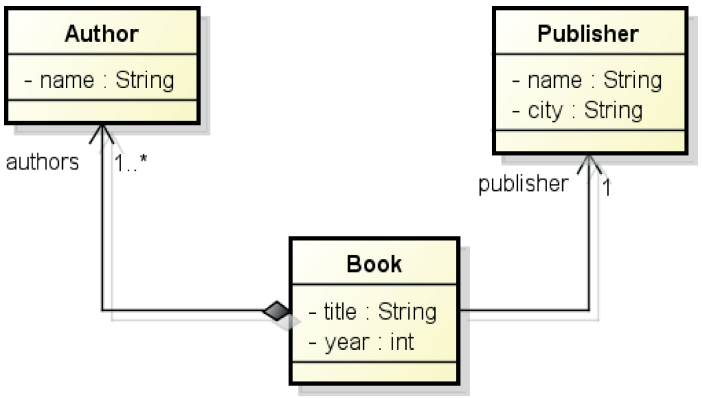

Figura 39: ejemplo de esquema con agregaciones y referencias. Extraído de [56]

Un objeto JSON se forma por el par clave valor, el valor puede ser de un tipo de dato primitivo, un objeto o un array de valores. Cuando el valor es otro objeto, se refiere a un agregado. También se puede representar referencias, que se pueden representar como un par, donde el valor es igual al valor de otra propiedad en otro objeto. Un documento del mismo tipo puede tener distinta estructura, este enfoque tiene en cuenta las distintas versiones del objeto, ya que las bases de datos NoSQL no tienen un esquema definido.

Se realiza una transformación de modelo a modelo de la siguiente forma: dado un modelo de objetos JSON, se crea una entidad por cada tipo de objeto JSON, ya sea un objeto raíz o embebido. Cuando se encuentra otra versión de un objeto también se crea una entidad que se asocia a la primera versión.

Para cada versión de entidad, primero se generan los atributos, que son aquellos que los tipos de datos de los valores son primitivos. Luego se generan las relaciones de agregación, es decir, las que el valor es un objeto embebido o un array de objetos. Y por último se generan las referencias, que pueden estar expresadas de distintas formas en los objetos JSON, pero lo que almacenan es una referencia a un campo de otro objeto. Las referencias opuestas se analizan luego de completar todas las referencias, para lo que se comprueba si a la entidad a la cual se hace referencia también contiene una referencia a la entidad original.

En la figura 40, a la derecha, se puede ver una representación gráfica del esquema inferido para el esquema JSON de la izquierda obtenido de la base de datos.

También se implementa una herramienta para la automatización que recibe como entrada el modelo JSON de la base de datos.

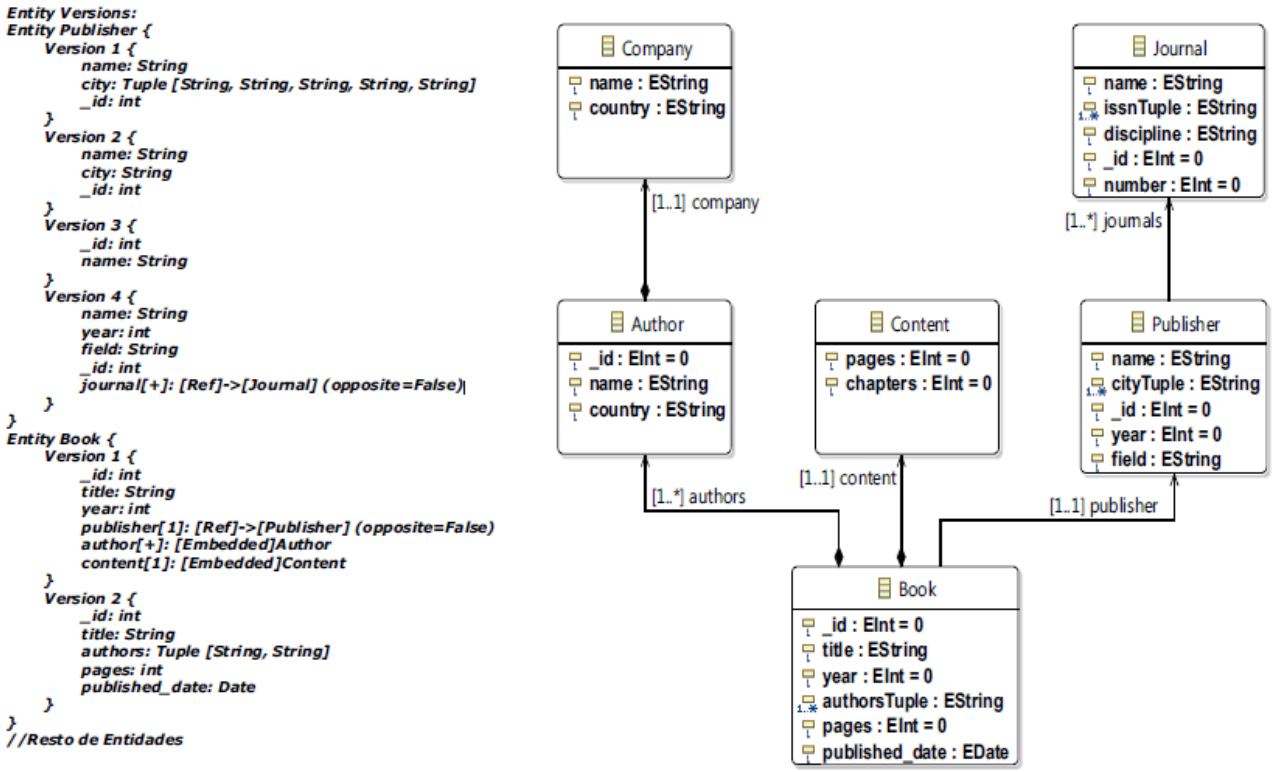

Figura 40: ejemplo de esquema inferido para esquema JSON. Extraída de [56]

6.6.5 Enfoque de Mok w.

En [58] se propone una metodología de diseño de esquema para el motor de base de datos DynamoDB. DynamoDB adopta el tipo clave-valor y también el tipo documental.

La metodología mencionada se desarrolla en 5 pasos:

- Generar un hipergrafo conceptual de acuerdo a los requerimientos. El hipergrafo es similar a un diagrama de clases UML. En la figura 41 se muestra un ejemplo del hipergrafo. En la figura 41 se observan seis conjuntos de objetos y tres conjuntos de relaciones. Los objetos son Estudiantes, SNN(Número de Seguridad Social), Semestre, Par Curso-Sección (contiene todos los pares de cada curso y las secciones del mismo), CrnSecId (conjunto de todas las cadenas de string) y Grado. Las relaciones son entre Estudiante y SSN, un estudiante está relacionado con un SSN, esto se representa con una línea y una flecha en ambos extremos de la línea, lo mismo pasa entre Par Curso-Sección y CrnSecID. La tercera relación es la que se da entre Semestre, Estudiante, Grado y Par Curso-Sección, que significa que para cada tupla del objeto Estudiante, el objeto Par Curso-Sección, y un objeto semestre solo puede haber un objeto Grado. El círculo cercano al Par Curso-Sección significa que este objeto puede no relacionarse con los otros tres.
- Identificar los patrones de acceso sobre el hipergrafo del punto anterior. Los patrones de acceso son las futuras consultas a la base de datos, se deben marcar los objetos involucrados en cada consulta.
- Simplificar el hipergrafo conceptual. Los objetos que tienen relación 1 a 1 se pueden simplificar. Ejemplo: Estudiante con SSN del hipergrafo de la figura 41.
- Generar un conjunto de esquemas jerárquicos. En la figura 41 se muestra un ejemplo de dos esquemas jerárquicos generados a partir de dos patrones de acceso. Los esquemas jerárquicos se definen según los patrones de acceso definidos en el punto 2, por ejemplo el modelo jerárquico de la izquierda de la figura 42 resuelve el patrón de acceso: por cada par CrnSecId y Semestre, hay un conjunto de SSN.
- Cargar el esquema físico al motor de base de datos de acuerdo a los esquemas jerárquicos definidos.

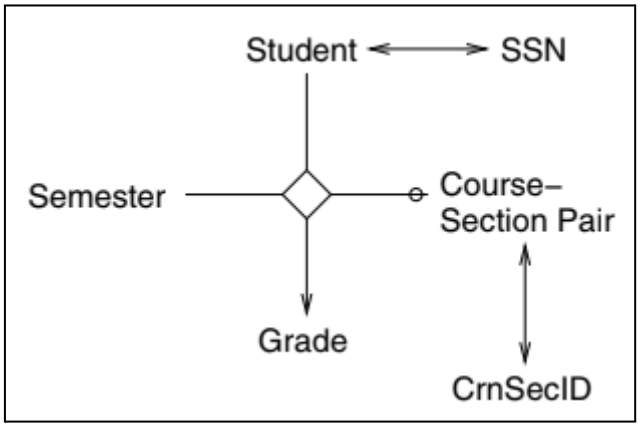

Figura 41: ejemplo de Hipergrafo conceptual. Extraída de [58]

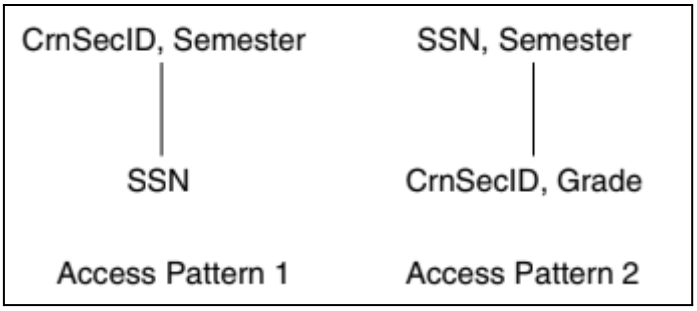

Figura 42: ejemplo de dos esquemas jerárquicos. Extraída de [58]

6.6.6 Enfoque de Gueidi A. ; Gharsellaoui, H. ; Ben Ahmed, S.

En [59] se presenta un modelado de esquemas unificado para soluciones NoSQL basado en un enfoque de mapeo. El enfoque hace hincapié en los tipos de bases de datos familia de columnas, pero además se puede aplicar a los tipos documental y orientadas a grafos.

Existe una variedad de esquemas de datos NoSQL para almacenar datos no estructurados. Se encontró una analogía entre los esquemas de datos relacionales y no relacionales para unificar ese esquema para todas las soluciones NoSQL por modelo. Los patrones de consulta también son fundamentales para el modelado de datos NoSQL y deben describirse en el modelo de datos, en el que las entidades suelen consultarse juntas. Estas entidades deben ser agregadas en el almacenamiento físico de datos NoSQL.

El enfoque se resume en las siguientes reglas:

- Regla 1: Relación uno a uno. Cada fila de una tabla de la base de datos está vinculada a una y sólo una fila de otra tabla. Podría estar embebido en una de las partes o separado.
- Regla 2: Relación Uno a Muchos, cada fila de la tabla puede estar relacionada con muchas filas de la tabla con la que se relaciona. Se puede embeber en cada grupo de NoSQL.
- Regla 3: Relación muchos a muchos, una o más filas de una tabla pueden estar relacionadas con 0, 1 o muchas filas de otra tabla. En un RDBMD es una tabla adicional. Podría generarse como referencia en cada documento del otro o en documentos adicionales, por ejemplo, en una base de datos de documentos como MongoDB
- Regla 4: Consulta frecuente utilizada para describir la relación. Agrupar entidades asociadas en una sola entidad compuesta si es posible. Es mejor crear un índice en la base de datos física.
- Regla 5: Las columnas se agrupan por línea y cada línea se identifica mediante un identificador único. Por lo general, se equiparan con tablas en el modelo relacional y se identifican con un único nombre.

Se definen 5 reglas para una analogía entre el modelo relacional y las bases de datos NoSQL familia de columnas. También se enuncian 6 reglas para mapear a las bases de datos orientadas a grafos.

Para obtener una imagen completa de las compensaciones de rendimiento, se realiza un experimento con un modelo de referencia TPC-H[60]. Se aplicó a una base de

datos orientada a columnas: HBase bajo Hadoop, y luego en Neo4j como base de datos orientada a grafos.

El modelo de datos utilizado para probar las capacidades de diseño de datos de cada base de datos se ha tomado del benchmark TPC-H como el mostrado en la figura 43.

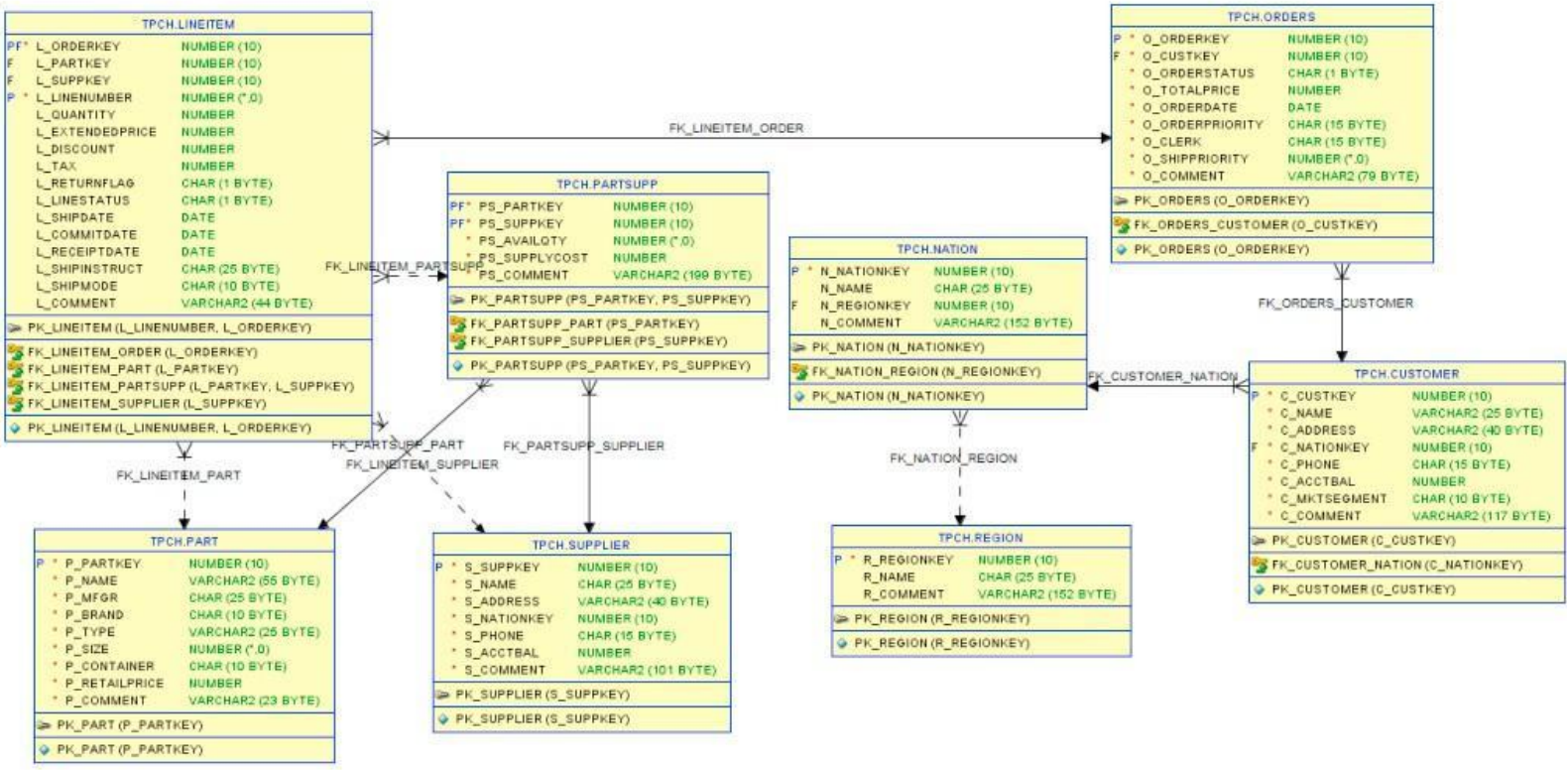

Figura 43: Modelo de datos Oracle TPC-H. Extraída de [59]

Capítulo 7 - Análisis comparativo de Procesos de Diseño de Bases de Datos NoSQL

7.1 Resumen comparativo

Luego de realizar el análisis de cada uno de los trabajos recopilados, se presenta una grilla comparativa para visualizar las distintas características que poseen los diferentes enfoques de procesos y/o métodos de diseño para motores de Bases de Datos NoSQL (clave-valor, documental, familia de columnas y grafos).

En este análisis se han definido los siguientes criterios de comparación:

- **Aplicable a:** tipo de base de datos (clave-valor, documental, familia de columnas o grafos), o motor de Bases de Datos NoSQL específico.
- **Enfoque:** pueden ser los siguientes:
    - *Ingeniería inversa:* a partir de un modelo físico realiza el trabajo inverso.
    - *Modelado de datos:* presenta un enfoque para una o más partes del diseño de las bases de datos, pero no llega a ser un proceso de diseño completo como el que se conoce de las bases de datos relacionales.
    - *Proceso de diseño:* realiza todos los pasos necesarios para el diseño de una base de datos, como los que se conocen para las bases de datos relacionales.
    - *Mapeo:* el enfoque presenta una herramienta o técnica que mapea de un modelo a otro modelo.
    - *Asesor de esquema:* el enfoque presenta un aplicación o algoritmo que dado ciertos datos de entradas propone esquemas posibles para un tipo de datos específico.
    - *Grupo de directrices:* conjunto de reglas para diseño de base de datos.
    - *Diseño Lógico:* enfoque que se aplica modelo y diseño lógico.
- **Herramienta de automatización:** si implementan una herramienta para automatizar alguna parte del enfoque.
- **Pautas definidas:** Si define todos los pasos necesarios para aplicar el proceso o técnica que describe.
- **Diseño completo:** si parte con un modelo inicial o utiliza algún modelo inicial conocido y llega hasta el modelo físico.
- **Diagramas:** diagramas que se utilizan en el enfoque descrito.
- **Ref:** referencia al artículos y/o trabajos analizados.
- **Nombre**: título con el que se referencia el enfoque en el capítulo 6.

A continuación, en las tablas 1, 2, 3, 4 y 5 se presenta el resultado del análisis realizado.

Tabla 1 - Enfoques aplicables a almacenamiento ***Documental***

Aplicable a	Enfoque	Herramienta de automatización	Pautas definidas	Diseño completo	Diagramas	Ref	Nombre
MongoDB	Ingeniería inversa	No	Si	No	Metamodelos UML Modelo físico documental Modelo físico NoSQL	[36, 37, 61]	Enfoque de Brahim, A. ; Ferhat, R. ; Zurfluh, G. Enfoque de Abdelhedi, F. ; Brahim, A. ; Ferhat, R. ; Zurfluh, G.
General	Herramienta de mapeo	Si	No	No	UML Físico JSON	[38]	Enfoque de C'anovasi, J. ; Jordi, C.
General	ingeniería inversa	No	Si	No	SG RG JSON	[40]	Enfoque de Klettke, M. ; Storl, U. ; Scherzinger, S.
General	Grupo de directrices	No	Si	Si	UML ERD	[41]	Enfoque de Atzenia, P. ; Bugiottib, F. ; Cabibboa, L. ; Torlonea, R.
General	Diseño Lógico	No	Si	No	DID (Diagrama de interrelación de documentos)	[42]	Enfoque de Rossel y Manna
General	Mapeo	No	Si	No	Fisico	[43]	Enfoque de Varga V.; Jánosi-Rancz, K.; Balázs, K.;

Tabla 2 - Enfoques aplicables a almacenamiento ***Clave-Valor***

Aplicable a	Enfoque	Herramienta de automatización	Pautas definidas	Diseño completo	Diagramas	Ref	Nombre
General	Diseño Lógico	No	No	Si	DER UML Lógico clave-valor	[44]	Enfoque de Rossel y Manna

Tabla 3 - Enfoques aplicables a almacenamiento de ***Grafos***

Aplicable a	Enfoque	Herramienta de automatización	Pautas definidas	Diseño completo	Diagramas	Ref	Nombre
Neo4j	Ingenierí a inversa	No	Si	No	Esquema físico Grafo lógico Modelo conceptual	[45]	Enfoque de Comyn-Wattia u, I. ; Akoka, J.

General	Diseño Lógico	No	Si	No	DER O-ER Diagrama de partición Plantilla	[47]	Enfoque de De Virgilio, R; Maccioni, A; Torlone, R.
General	Modelado de datos	No	Si	Si	Modelo conceptual Reglas de mapeo a modelo físico	[48]	Enfoque de Pokorný, J.

Tabla 4 - Enfoques aplicables a almacenamiento de ***Familia de Columnas***

Aplicable a	Enfoque	Herramienta de automatización	Pautas definidas	Diseño completo	Diagramas	Ref	Nombre
Cassandra	Proceso de diseño	Si	Si	Si	Modelo conceptual Modelo lógico Modelo físico Diagrama Chebotko	[50]	Enfoque de Chebotko, A. ; Kashlev, A. ; Lu, S.
General	Asesor de esquemas	Prototipo para Cassandra	Si	Si	Modelo conceptual Modelo físico	[51]	Enfoque de Mior, M.; Salem, K.; Aboulnaga, A.; Liu, R.

5 - Enfoques aplicables a más de un tipo de almacenamiento

Aplicable a	Enfoque	Herramienta de automatización	Pautas definidas	Diseño completo	Diagramas	Ref	Nombre
Familia de columnas Documental	Proceso de diseño	No	Si	Si	Generic Data Metamodel (GDM), Metamodelo para Familia de Columnas Metamodelo para Documental	[52]	Enfoque de De La Vega, A. ; García-Saiz, D. ; Blanco, C. ; Zorrilla, M. ; Sánchez, P.
Documental Clave-Valor Familia de Columnas	Proceso de diseño	No	Si	Si	UML Lógico NoSQL (NoAM) Físico	[53]	Enfoque de Atzenia, P. ; Bugiottib, F. ; Cabibboa, L. ; Torlonea, R
General	Modelado de datos	No	Si	No	Conceptual NoSQL Lógico JSON	[54, 55]	Enfoque de Banerjee, S. ; Sarkar, A.
Documental Clave-Valor Familia de Columnas	Ingeniería inversa	Si	Si	No	JSON UML	[56, 57]	Enfoque de Morales S. ; Garcia, J. ; Ruiz, S.

DynamoDB	Modelado de datos	No	Si	Si	Hipergrafo de modelo conceptual Esquema jerárquico	[58]	Enfoque de Mok w.
Documental Familia de Columnas Grafos	Mapeo de datos	No	Si	No	No utiliza	[59]	Enfoque de Gueidi A. ; Gharsellaoui , H. ; Ben Ahmed, S.

Capítulo 8 - Conclusiones y trabajo futuro

En los primeros capítulos se han presentado las bases teóricas más relevantes, explorando la evolución de las bases de datos, los diferentes tipos existentes y las propiedades ACID que caracterizan a las bases de datos relacionales tradicionales. Del mismo modo, se ha profundizado el estudio de las bases de datos NoSQL, destacando su propiedad BASE y el teorema de CAP, así como los diferentes tipos de bases de datos NoSQL existentes y algunos motores de Bases de Datos específicos que se utilizan en la actualidad.

En ese contexto, el aspecto clave abordado ha sido el diseño de bases de datos. Se han analizado en detalle los procesos de diseño conocidos y se ha proporcionado una visión integral de éstos. Se han estudiado los trabajos relacionados a este tema. Luego se ha llevado a cabo un exhaustivo análisis de los enfoques de diseño de bases de datos NoSQL.

Finalmente, ha presentado un minucioso análisis comparativo de los enfoques relevados.

Se han considerado un total de 18 enfoques, incluyendo aquellos que se basan en la ingeniería inversa. Es importante destacar que, si bien siempre se recomienda contar con el esquema de la base de datos, el hecho de aplicar un proceso de ingeniería inversa implica que no se ha llevado a cabo un proceso de diseño completo para obtener el esquema físico de la base de datos. Por lo tanto, puede haber aspectos que no se hayan tenido en cuenta al pensar en un esquema físico, a diferencia de como se haría en un proceso de diseño convencional.

Dentro de los enfoques de ingeniería directa, solo se encontraron tres procesos de diseño completos, es decir, que abarcan las etapas de diseño conocidas para las bases de datos relacionales.

No se encontró ningún proceso de diseño que sea aplicable a todos los tipos de bases de datos NoSQL; dos son aplicables a más de un tipo, pero no a todos, y uno sólo es aplicable a la familia de columnas. Esto se debe a que NoSQL no es un tipo de base de datos, sino que engloba un conjunto de tipos de bases de datos que cumplen las características mencionadas en los primeros capítulos sobre bases de datos NoSQL, pero la forma de estructurar los datos es totalmente diferente.

No se encontró ningún proceso de diseño completo aplicable a bases de datos orientadas a grafos. Para este tipo de base de datos se recomienda el enfoque presentado en 6.4.3, dado que es el más completo, presenta un esquema conceptual para este tipo de bases de datos y reglas de mapeo para llegar al modelo/esquema físico.

Se puede apreciar que existen más enfoques para el tipo de bases de datos documentales en comparación con otros tipos de bases de datos. Esto se debe a que los tipos de bases de datos documentales son aplicables a una amplia gama de contextos de problemas generales, mientras que los otros tipos de bases de datos no son aplicables en todos los contextos de problemas.

En resumen, este trabajo ha ofrecido una visión integral y detallada de la situación actual para el diseño de bases de datos NoSQL, analizando minuciosamente los enfoques de diseño y evaluando su aplicabilidad.

Como trabajo futuro, se plantea la selección de aquellos enfoques que realizan un proceso de diseño y su posterior evaluación en proyectos reales. El objetivo es mejorar y enriquecer el diseño de bases de datos NoSQL, teniendo en cuenta aspectos clave como la escalabilidad horizontal y otras características que no se han abordado exhaustivamente hasta ahora.

Además, a partir de los hallazgos y conclusiones obtenidos en este estudio, se pretende presentar un proceso de diseño específico para algunos tipos de bases de datos NoSQL. Esta contribución sería valiosa para abordar una necesidad actual en el campo de las bases de datos NoSQL, donde la falta de procesos de diseño completos ha sido una limitación importante.

A través de los trabajos futuros propuestos, se espera generar mejoras significativas en los procesos de diseño y promover el uso efectivo de las bases de datos NoSQL en proyectos reales, aportando soluciones innovadoras y adaptadas a las necesidades actuales del campo de la gestión de datos.

Bibliografía

1. Bertone, R., Thomas, P. (2011). Introducción a las bases de datos: fundamentos y diseño. Prentice Hall / Pearson Educación, 2011. ISBN: 9876151363, 9789876151368.
2. Batini, C., Ceri, S., Navathe, S. B (1994). Diseño Conceptual de Bases de Datos, un enfoque de entidades-interrelaciones. Addison-Wesley Iberoamericana. ISBN 0-201-60120-6 (1994).
3. Edgar Frank Codd. https://es.wikipedia.org/wiki/Edgar_Frank_Codd. Julio 2023.
4. T. T. Le and X. Lam Pham, "Towards NoSQL databases: Experiences from actual projects," 2022 3rd International Conference on Big Data Analytics and Practices (IBDAP), 2022, pp. 15-20, doi: 10.1109/IBDAP55587.2022.9907664. https://ieeexplore.ieee.org/document/9907664
5. Marrero, L., Olsowy, V., Tesone, F., Thomas, P., Delia, L., Pesado, P. (2021). Performance Analysis in NoSQL Databases, Relational Databases and NoSQL Databases as a Service in the Cloud. In: Pesado, P., Eterovic, J. (eds) Computer Science – CACIC 2020. CACIC 2020. Communications in Computer and Information Science, vol 1409. Springer, Cham. https://doi.org/10.1007/978-3-030-75836-3_11
6. N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," in Computer, vol. 43, no. 2, pp. 12-14, Feb. 2010, doi: 10.1109/MC.2010.58. https://ieeexplore.ieee.org/document/5410700
7. Johnson, J. L. (2000). Bases de Datos: modelos, lenguajes, diseños. OXFORD University Press, 2000. ISBN: 9706134611, 9789706134615.
8. Carlo Batini, Stefano Ceri, Shamkant B. Navathe. Diseño Conceptual de Bases de Datos, un enfoque de entidades-interrelaciones. ISBN 0-201-60120-6 (1994).
9. Un estudio de procesos de diseño de bases de datos NoSQL. Marrero, L.; Olsowy, V.; Tesone, F.; Thomas, P.; Corbalán, L.; Fernández Sosa, J.; Pesado, P. XXVIII Congreso Argentino de Ciencias de la Computación (CACIC 2022). ISBN: 978-987-1364-31-2. http://sedici.unlp.edu.ar/handle/10915/149452.
10. A Relational Model of Data for Large Shared Data Banks. https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf. Julio de 2023.
11. DB-Engines. https://db-engines.com/en/ranking. Julio de 2023.
12. Rackspace. https://en.wikipedia.org/wiki/Rackspace_Technology. Julio de 2023.
13. Time Graphics. https://time.graphics/es/event/1212428. Julio 2023
14. Análisis de performance en Bases de Datos NoSQL y Bases de Datos Relacionales. Pesado P., Thomas P., Delía L., Marrero L., Olsowy V., Tesone F.. XXVI Congreso Argentino de Ciencias de la Computación (CACIC 2020). ISBN 978-987-4417-90-9. http://sedici.unlp.edu.ar/handle/10915/114202.
15. Un estudio comparativo de bases de datos relacionales y bases de datos NoSQL. Pesado P., Thomas P., Delía L., Marrero L., Olsowy V., Tesone F., Fernandez S. J. XXV Congreso Argentino de Ciencias de la Computación (CACIC 2019). ISBN 978-987-688-377-1. http://sedici.unlp.edu.ar/handle/10915/91403.
16. NoSQL: modelos de datos y sistemas de gestión de bases de datos. Migani, S., Vera, C., Lund, M. XX Workshop de Investigadores en Ciencias de la Computación (WICC

2018, Universidad Nacional del Nordeste). http://sedici.unlp.edu.ar/handle/10915/67258.
17. Teorema de CAP. https://www.ibm.com/mx-es/topics/cap-theorem#:~:text=El%20teorema%20CAP%20aplica%20un,'P%20'%20en%20CAP). . Julio de 2023
18. Redis. https://redis.io/. Julio 2023.
19. DinamoDB. https://aws.amazon.com/es/dynamodb. Julio 2023.
20. MongoDB. https://www.mongodb.com/es. Julio 2023.
21. CouchDB. https://couchdb.apache.org/. Julio 2023.
22. Apache Cassandra. https://cassandra.apache.org/_/index.html. Julio 2023.
23. Apache HBase. https://hbase.apache.org/. Julio 2023.
24. Neo4j. https://neo4j.com/. Julio 2023.
25. Memgraph. https://memgraph.com/. Julio 2023.
26. https://www.academia.edu/23202106/DEFINICI%C3%93N_UML. Agosto de 2023.
27. Representación de una Base de Datos Orientada Objetos mediante un Diagrama de Clases UML. https://www.editorialpencil.es/%E2%96%B7-dos-ejemplos-de-diagramas-de-clases-uml-dos-mil-veintiuno/. Agosto de 2023
28. NoSQL database design processes: a comparative study. Marrero, L.; Olsowy, V.;| Thomas, P. XI Jornadas de Cloud Computing, Big Data & Emerging Topics. ISBN: 978-950-34-2271-7. http://sedici.unlp.edu.ar/handle/10915/155446. Junio 2023.
29. Noa Roy-Hubara, Arnon Sturm. Design methods for the new database era: a systematic literature review. Software and Systems Modeling (2020) https://www.springerprofessional.de/en/software-systems-modeling/5337962.
30. Noa Roy-Hubara, Arnon Sturm. Exploring the Design Needs for the New Database Era. Springer (2018) https://link.springer.com/chapter/10.1007/978-3-319-91704-7_18.
31. Davoudian, Ali and Chen, Liu and Liu, Mengchi. A survey on NoSQL stores. ACM Computing Surveys (CSUR), volumen=51, paginas=1--43, año=2018, publicado=ACM New York, NY, USA.
32. IEEE Xplore.https://ieeexplore.ieee.org/Xplore/home.jsp. Julio 2023.
33. Springer. https://link.springer.com/. Julio 2023.
34. Google Scholar. https://scholar.google.com/. Julio 2023.
35. ResearchGate. https://www.researchgate.net/. Julio de 2023.
36. Brahim, A.; Ferhat, R. and Zurfluh, G. (2019). Model Driven Extraction of NoSQL Databases Schema: Case of MongoDB.In Proceedings of the 11th International Joint Conference on Knowledge Discovery. V.1: KDIR, ISBN 978-989-758-382-7, pages 145-154.
37. Fatma ABDELHEDI, Amal AIT BRAHIM, Rabah TIGHILT FERHAT, Gilles ZURFLUH. Reverse engineering approach for NoSQL databases. 22nd International Conference, DaWaK 2020, Bratislava, Slovakia, September 14–17, 2020.
38. Izquierdo, J. L. C., & Cabot, J. (2016). JSONDiscoverer: Visualizing the schema lurking behind JSON documents. Knowledge-Based Systems, 103, 52-55.
39. https://aws.amazon.com/es/what-is/api/. Agosto de 2023.
40. M. Klettke, U. St ¨orl, S. Scherzinger, Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores, in: BTW conf., 2015, pp. 425–444.
41. Paolo Atzenia,Francesca Bugiottib,Luca Cabibboa,Riccardo Torlonea. Data Modeling Guidelines for NoSQL Document-Store Databases. International Journal of Advanced Computer Science and Applications (2018).

42. Diseño de Bases de Datos basadas en Documentos: Modelo de Interrelación de Documentos. Rossel Gerardo, Manna Andrea. XXII Congreso Argentino de Ciencias de la Computación (CACIC 2016). Páginas 662-671. http://sedici.unlp.edu.ar/handle/10915/56746.
43. Viorica Varga, Katalin Tünde Jánosi-Rancz, Balázs Kálmán. Conceptual Design of Document NoSQL Database with Formal Concept Analysis. Acta Polytechnica Hungarica. Vol. 13, No. 2, (2016).
44. Gerardo ROSSEL, Andrea MANNA. A Modeling methodology for NoSQL Key-Value databases. Database Systems Journal, Bucharest, Romania. 12-18, August (2017).
45. Isabelle Comyn-Wattiau;Jacky Akoka. Model driven reverse engineering of NoSQL property graph databases: The case of Neo4j. IEEE International Conference on Big Data (2017).
46. https://www.researchgate.net/publication/220595979_Model_driven_architecture_MDA . Agosto de 2023.
47. Roberto De Virgilio, Antonio Maccioni, and Riccardo Torlone. Model-Driven Design of Graph Databases. Conceptual Modeling 33rd International Conference, ER 2014, Atlanta, GA, USA, October 27-29, 2014.
48. Jaroslav Pokorný. Conceptual and Database Modelling of Graph Databases. IDEAS '16: Proceedings of the 20th International Database Engineering & Applications Symposium. Julio 2016. ISBN: 978-1-4503-4118-9 Pages 370–377
49. https://www.oracle.com/ar/autonomous-database/what-is-graph-database/#:~:text=Un%20grafo%20de%20propiedades%20tiene,a%20los%20que%20est%C3%A1n%20asociados. Agosto 2023.
50. Artem Chebotko, Andrey Kashlev, Shiyong Lu.A Big Data Modeling Methodology for Apache Cassandra. IEEE International Congress on Big Data (2015).
51. Michael Joseph Mior, Kenneth Salem, Ashraf Aboulnaga, Rui Liu. NoSE: Schema Design for NoSQL Applications. IEEE Transactions on Knowledge and Data Engineering (2016).
52. Alfonso de la Vega, Diego García, Saiz Carlos Blanco, Marta Zorrilla, Pablo Sánchez. Mortadelo: Automatic generation of NoSQL stores from platform-independent data models. Future Generation Computer Systems. Volumen 105, Abril 2020, Paginas 455-474.
53. Paolo Atzeni, Francesca Bugiotti, Luca Cabibbo, Riccardo Torlone. Data Modeling in the NoSQL World. HAL open science. https://hal.archives-ouvertes.fr/hal-01611628 (2020).
54. Shreya Banerjee, Anirban Sarkar. Modeling NoSQL Databases: From Conceptual to Logical Level Design. 3rd International Conference on Applications and Innovations in Mobile Computing (AIMOC – 2016) At: Kolkata, India.
55. Shreya Banerjee, Anirban Sarkar. Logical level design of NoSQL databases. IEEE Region 10 Conference (2016) (TENCON).
56. Severino Feliciano Morales, Jesús García Molina,Diego Sevilla Ruiz. Inferencia del esquema en bases de datos NoSQL a través de un enfoque MDE. Jornadas de Ingeniería del Software y Bases de Datos (2017).
57. Diego Sevilla Ruiz, Severino Feliciano Morales, Jesús García Molina. Inferring Versioned Schemas from NoSQL Databases and Its Applications. (2015) https://link.springer.com/chapter/10.1007/978-3-319-25264-3_35 (Springer).

58. W. Y. Mok.A Feasible Schema Design Strategy for Amazon DynamoDB: A Nested Normal Form Approach. Industrial Engineering and Engineering Management (IEEM 2020).
59. Afef Gueidi, Hamza Gharsellaoui, Samir Ben Ahmed. Towards Unified Modeling for NoSQL Solution Based on Mapping Approach. 25th KES-2021. http://kes2021.kesinternational.org/
60. Multidimensional database design via schema transformation: turning tpc h into the tpc h* d multidimensional benchmark. Cuzzocrea, A., Moussa, R. Proceedings of the 19th International Conference on Management of Data (2013), Computer Society of India. pp. 56–67.
61. Amal AIT BRAHIM; Rabah TIGHILT FERHAT; Gilles ZURFLUH. Extraction process of conceptual model from a document-oriented NoSQL database. 11th International Conference on Knowledge and Systems Engineering (KSE) (2019).