****

**Rendimiento de bases de datos**

**en modelos orientados**

**a dominios funcionales.**

|  |  |  |
| --- | --- | --- |
| Titulación:  Master en Big Data  y Data Science  Curso académico  2020 – 2022 | Alumno/a:  Herranz Segundo, Daniel  D.N.I: 53432485-N  Director/a de TFM:  Pablo Suárez-Otero González | Convocatoria: Primera.  Mayo 2022 |

Índice

[Tabla de Acrónimos y definiciones 8](#_Toc99788693)

[Resumen 10](#_Toc99788694)

[1. Enfoque e introducción 12](#_Toc99788695)

[2. Objetivos 15](#_Toc99788696)

[3. Referencias previas 16](#_Toc99788697)

[4. Situación tecnológica en la industria 17](#_Toc99788698)

[1.1. Bases de datos relacionales SQL 19](#_Toc99788699)

[1.2. Bases de datos NoSQL 21](#_Toc99788700)

[1.2.1. Tipo clave-valor 21](#_Toc99788701)

[1.2.2. Orientadas a documentos 22](#_Toc99788702)

[1.2.3. Basadas en grafos 22](#_Toc99788703)

[1.2.4. Columnares 22](#_Toc99788704)

[1.3. Bases de datos NewSQL 24](#_Toc99788705)

[1.4. Comparativa NoSQL respecto a CAP 24](#_Toc99788706)

[6. Definición del modelo de información 26](#_Toc99788711)

[6.1. Casos de uso de pruebas 27](#_Toc99788712)

[6.2. Definición de entidades relacionales 28](#_Toc99788713)

[6.3. Definición del modelo de datos documental 29](#_Toc99788714)

[6.4. Definición del modelo de información columnar 32](#_Toc99788715)

[7. Metodología y entornos de pruebas 38](#_Toc99788716)

[7.1. Plataforma de test 42](#_Toc99788720)

[8. Desarrollo del proyecto 43](#_Toc99788721)

[9. Resultados 44](#_Toc99788722)

[10. Conclusión y trabajos futuros 45](#_Toc99788723)

[11. Referencias 47](#_Toc99788724)

[Anexos I – Codigo fuente e instalaciones 49](#_Toc99788725)

[Anexos II – Diccionario de datos 53](#_Toc99788729)

A mis compañeros de master, por estar siempre ahí. Juntos somos más.

A mis profesores por su profesionalidad y paciencia.

A mi mujer Manuela por su apoyo incondicional y respeto a lo que hago y a mis hijos Ian y Leo, por su interminable curiosidad y por hacerme superarme para ser un ejemplo para ellos. Ojalá algún día recordéis de vuestros padres que el aprendizaje no acaba nunca y que el éxito es simplemente saber algo más que ayer.

Índice de ilustraciones

Ilustración 1. Ranking de las BBDD más demandadas en 2020. 19

Ilustración 2. Ejemplo de estructura BBDD columnar 23

Ilustración 3. Diagrama modelo datos columnar datos maestros 33

Ilustración 4. Diagrama modelo datos columnar tipos de dato 34

Ilustración 5. Tabla columnar por consulta de moneda 34

Ilustración 6. Tabla columnar por consulta de agregados de importe 35

Ilustración 7. Diagrama modelo datos columnar por dominios 36

Ilustración 8. Arquitectura cluster InnoDB MySQL sandbox 38

Ilustración 9. Arquitectura cluster de pruebas MongoDB 39

Ilustración 10. Modelo ETL del proyecto 41

Ilustración 11. Arquitectura ecosistema pruebas 42

Ilustración 12. Arquitectura DIH 46

Índice de tablas

Tabla 1. Comparativa CAP (Elaboración propia) 25

Tabla 3. Trazabilidad entidades modelo relacional (fuente. Elaboración propia) 37

Tabla 4. Trazabilidad entidades modelo documental (fuente. Elaboración propia) 37

Tabla 5. Trazabilidad entidades modelo columnar (fuente. Elaboración propia) 38

Tabla 6. Definición datos entidad OBReadAccount5 (fuente. Elaboración propia) 53

Tabla 7. Definición datos entidad OBReadBalance1 (fuente. Elaboración propia) 54

Tabla 8. Definición datos entidad OBParty2 (fuente. Elaboración propia) 55

# Tabla de Acrónimos y definiciones

|  |  |
| --- | --- |
| **SIGLAS** | **SIGNIFICADO** |
| **DDD** | Domain-driven design (diseño orientado a dominios). |
| **BBDD** | Base de datos. |
| **BIAN** | Banking Industry Architecture Network |
| **SQL** | Structured Query Language. |
| **NoSQL** | Not only SQL. |
| **JSON** | Java Script Object Notation. |
| **BSON** | Binary JSON. |
| **TI** | Tecnología Informática. |
| **TTM** | Time to market. |
| **PSD2** | Payment services (PSD 2) - Directive (EU) 2015/2366. |
| **DBMS** | Database Management System. |
| **DL/I** | Data Language Interface. |
| **DB2** | Gestión de base de datos de IBM. |
| **RDBMS** | Relational DataBase Management System. |
| **CAP** | Coherencia*,*disponibilidad y tolerancia a la partición. |
| **IA** | Inteligencia artificial. |
| **CDO** | Chief Data Officer. |
| **ACID** | Atomicidad, Consistencia, Aislamiento y Durabilidad. |
| **OLTP** | Online Transaction processing. |
| **OpenBanking** | Estándar de intercomunicación entre servicios financieros. |
| **OpenData** | Proyecto de la Open Knowledge Foundation para promover la creación de conocimiento y su distribución. |
| **OpenAPI** | Estándar unificado la descripción de APIs. |
| **OLAP** | Online Analytical processing. |
| **E/R** | Entidad relación |
| **CQL** | Cassandra Query Language |
| **MQL** | MongoDB Query Language |
| **DSE** | DataStax Enterprise |
| **CSV** | Comma Separated Values |
|  |  |
|  |  |
|  |  |

# Resumen

El trabajo tiene como objetivo analizar el rendimiento de distintas soluciones de base de datos actuando con modelos orientados a agregados de dominios de información. No se pretende extraer conclusiones únicamente técnicas, sino una visión holística con casos de uso de negocio reales.

Los ejemplos del análisis se centran en soluciones orientadas a entidades bancarias, sector con presencia en cualquier región geográfica y con necesidades tanto de alta transaccionalidad como de elevada volumetría de datos, pero puede ser extrapolado a cualquier otra actividad adaptando las características a las casuísticas expuestas en el estudio.

Se pretende extraer criterios con los que perfiles como los de arquitectos de sistemas TI, puedan apoyarse en la elección tecnológica más adecuada, reduciendo tanto el tiempo empleado en el análisis inicial como eficiencia en la explotación posterior del sistema.

**Keywords**: dominios, entidades bancarias, criterios, eficiencia

The objective of the work is to provide elements to help in the selection process of the base technology applicable to data models oriented to information domains, so as to improve the efficiency of the systems and the final cost. It is not intended to draw only technical conclusions by applying synthetic test data, but rather a holistic view with business cases.

The examples of the analysis focus on solutions aimed at banking entities, sectors with a presence in any geographical region and with needs for both high transactionality and data volume, but it can be extrapolated to any other sector and project by adapting the characteristics to the cases exposed. at work.

The aim is to extract criteria with which profiles such as those of IT solutions architects can rely on the most appropriate technological choice, reducing both the time spent on the initial analysis and the efficiency in the subsequent exploitation of the system.

**Keywords**: Domains, banking entities, criteria, efficiency

# Enfoque e introducción

Actualmente muchas corporaciones, sin importar su desempeño, se encuentran en un proceso de transformación digital con el objetivo de mejorar su TTM (time to market) y productividad, reduciendo los costes migrando cores habitualmente mainframe hacia entornos ligeros distribuidos.

Como parte de esta transformación, las entidades planean una racionalización de su modelo informacional basada en diseños informacionales DDD (Domain-driven design) y arquitecturas hexagonales. En especial el sector bancario requiere de un enfoque hacia el open banking, permitiéndoles interactuar con terceros y cumplir con los requerimientos normativos europeos PSD2 (Payment services (PSD 2) - Directive (EU) 2015/2366, 2015). Para logar este propósito es habitual apoyarse en el estándar BIAN como particionado de dominios, lo que supone una necesidad de rediseño del modelo de datos existente en sus sistemas legacy.

Desde la perspectiva del profesional IT, este rediseño supone un reto en cuanto a determinar la mejor solución software a aplicar en la transformación tecnológica. En muchas ocasiones se parte del temido concepto de “folio en blanco”, encontrando problemas que son recurrentes, donde la capacidad de respuesta de las bases de datos, la disponibilidad, el volumen de información y la posibilidad de escalar y extender el modelo en un futuro resultan clave. Para lograr esto, en muchos proyectos se implementan soluciones basando las elecciones en el estado del arte, modas o simplemente reutilizando los motores de BBDD existentes en la compañía, sin una valoración más compleja.

Las últimas décadas han estado dominadas por motores SQL representando el mundo real mediante entidades que se implementan en tablas de columnas o atributos de la entidad y filas que describen a cada elemento. Estos modelos son normalizados y se distribuyen formando diversas tablas que se relacionan mediante claves primarias y externas. “La combinación de información basada en un valor coincidente para una clave principal y una clave externa en varias tablas en el modelo relacional requiere el uso de una operación de combinación. Cuanto más grande sea el esquema y más tablas deban unirse, más tardará la base de datos relacional en recuperar los datos” (Parker, Poe, & Vrbsky, 2013)

Por otro lado, ante la ingente cantidad de información a procesar y la variedad de la misma en datos no estructurados, han provocado la aparición de las BBDD NoSQL. Estas tecnologías dejan a un lado las capacidades ACID que aseguran transacciones atómicas características de las BBDD SQL, en favor de los principios CAP donde se parte del hecho en el que existe “una imposibilidad de garantizar al mismo tiempo consistencia, disponibilidad y tolerancia a la partición en un sistema” (Seth Gilbert, Nancy A. Lynch, 2012), pero sin la capacidad de hacer consultas complejas sobre varias entidades (joins) que forman un agregado de información, entendido el concepto de agregado como “datos en unidades que tienen una estructura más compleja que un conjunto de tuplas” (Pramod J. Sadalage, Fowler M, NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence , Addison-Wesley, 2013, p. 26)., como sí ocurre en las SQL, aunque mejorando por otro lado en la flexibilidad del esquema.

Los modelos de datos NoSQL representan las entidades por su parte a modo de documentos y colecciones (bases documentales), columnares o de clave-valor. Al ser un enfoque sin esquema, es posible asociar la información sin restricciones uniendo varias entidades en un único elemento (más allá del rendimiento por tamaño de documento o incluso por estrategias de réplica), pero genera a la larga problemas de acoplamiento en diseños DDD.

Respetando un correcto particionado de entidades en dominios funcionales, es importante separar la problemática de casos de negocio que aplican a un solo dominio funcional, donde a priori las bases de datos NoSQL parten de ventaja teórica dado su enfoque no estructurado en tablas dispersas, de los casos de uso donde intervienen transacciones y consultas que involucran a un agregado de varios dominios. Actualmente esto es un problema al que debe enfrentarse los profesionales IT de la industria bancaria.

Se toma como referencia el estándar abierto Open Banking como implementación del enfoque de particiones de dominio de alto nivel BIAN, extrapolando de ellos un modelo de información que represente un agregado de datos base para el estudio.

Este modelado de agregado conceptual inicial genérico, nos permitirá implementar modelados físicos sobre las que desarrollar las pruebas, realizando mediciones sobre los diferentes modelos de datos con mayor uso y madurez en la industria: Relacional o SQL y NoSQL.

Variando parámetros de prueba respecto a volumetría de datos, se definirán escenarios sobre los que extraer los resultados. Estos escenarios técnicos se aplicarán a casos de uso de tipo transaccional y de tipo consulta, inserción y eliminación de registros, tanto sobre uno como sobre varios dominios al mismo tiempo.

Esta toma de decisión de que tecnologías utilizar, sin una inversión adecuada en analizar el problema en profundidad, supone habitualmente un coste económico en los proyectos, una dilación en los tiempos de puesta en producción y, si es fallido, frustraciones de los interesados.

# Objetivos

El principal objetivo de este análisis es analizar la mejor solución a aplicar para solucionar el problema recurrente en la industria como es implementar aplicaciones que deban interactuar, de forma productiva y con gran volumen de datos, con una información que se encuentra separada tanto lógica como físicamente, por diferentes dominios funcionales, los dispuestos por BIAN en el caso del sector bancario que nos ocupa específicamente en el trabajo.

Los resultados deben servir de base de conocimiento para que un diseñador de soluciones IT con suficiente juicio experto, pueda tener un análisis de referencia al iniciar proyectos, y discernir cuál es la mejor elección software a aplicar entre varios productos analizados, tanto SQL como NoSQL, según el caso de uso de negocio concreto con que se encuentre.

A día de la ejecución de este estudio y, debido a factores como necesidad de disponibilidad del servicio, seguridad y privacidad, factores de cumplimiento normativo y temor a dependencia con el proveedor cloud, no existe una extendida migración a la nube del core de entidades bancarias (modelo de negocio sobre el que pivota el estudio). Es por ello que las pruebas se realizarán en una instalacion OnPremise.

Este trabajo pretende discernir la implementación opensource que mejor se adapta por rendimiento a proyectos DDD. Por una razón de tiempo y recursos, tampoco se considera del alcance del estudio el posible tunning y configuraciones avanzadas. Se delimita el trabajo a productos Opensource de modo que el coste de licencias no suponga un sesgo para la elección de la tecnología.

Como caso de negocio habitual del core bancario y utilizando casos de uso realistas y un ejemplo de separación de los dominios involucrados que recomienda BIAN, se definirá una abstracción que cubra las entidades relacionales más habituales (1:1, 1:N y N:M..). Este modelo será de igual modo trasladado a modelos no estructurados para aplicarlos a las BBDD NoSQL: documental, clave-valor y columnar, siempre teniendo en cuenta el requisito de separación de información en dominios y, por consiguiente, en diferentes esquemas, colecciones o espacios de nombres según sea la tecnología.

Se analizarán parámetros de medición del propio motor de base de datos, como tiempos combinados de lectura/escritura, memoria usada y consumo de CPU. Estos elementos han sido objeto habitual de mediciones sintéticas en otros estudios similares de bases de datos.

Además de esos datos cuantificables se ponderarán también en el resultado los criterios funcionales que forman el teorema CAP y ACID. Dependiendo del problema de negocio que se evalúe debe tratarse también como criterio, las virtudes y defectos de los productos analizados respecto a este teorema.

# Referencias previas

Existe numerosa documentación académica respecto a análisis de rendimiento en bases de datos, incluso comparativas relacionando las diferentes tecnologías NoSQL, SQL, NewSQL.

Es menos habitual encontrar estudios que traten de relacionar las tecnologías y motores de bases de datos con problemas del mundo real y los desafíos que surgen en la implementación de dominios funcionales lógicos en modelos físicos, donde el concepto de agregado genera una problemática a resolver.

Por mi experiencia profesional en la implementación de soluciones IT en la industria bancaria, he detectado la necesidad de establecer criterios que sirvan como acelerador del diseño de proyectos orientados a dominios y la interacción entre los mismos, sobre todo cuando se trata la transaccionalidad. En el caso de la banca, es habitual la necesidad de aplicar el estándar de particionado BIAN. La realidad es que los modelos físicos de los sistemas legacy que encontramos en los clientes, dista mucho de las recomendaciones más teóricas de como implantar la separación de dominios funcionales.

Un libro de referencia en este trabajo es NoSQL Distilled (Fowler M & Sadalage P, 2012) donde se hace una introducción a la problemática comentada de dominios y agregados de información y como orientar la implementación a alto nivel de estos modelos en bases de datos NoSQL y relacionales SQL, pero sin aportar métricas de pruebas sobre instalaciones, siendo esto uno de los objetivos de este trabajo.

Next Generation Databases NoSQL, NewSQL, and Big Data (Harrison, 2015), hace referencia a patrones de implementación de soluciones distribuidas (replicación, sharding, clustering…), pero desde una perspectiva que no responde en mi opinión al reto de aplicación de dominios funcionales, ni aplicando la teoría a casos reales.

Así mismo existen otros estudios que han servido de base a este, donde hay comparaciones sino clasificación de los distintos motores (Correlation and comparison of NoSQL specimen with relational data store, 2015).

Existen otras referencias donde si se comenta la problemática de funciones agregadas (Caso de uso de negocio que se trata en este trabajo) y sé asegura que las bases de datos NoSQL analizada (MongoDB) “funciona igual o mejor que la base de datos relacional, excepto cuando se utilizan funciones agregadas”. (Parker, Poe, & Vrbsky, 2013). Por nuestra parte, pretendemos extender el estudio a otros modelos: columnares y clave-valor y no solo hacer referencia a funciones agregadas como medias, máximos, mínimos o desviaciones estándar sino al concepto de particiones de información y su impacto en desarrollos orientados a dominios.

# Situación tecnológica en la industria

Actualmente existe una enorme variedad de motores de bases de datos disponibles. Centraremos el análisis en motores de tipo OpenSource, de modo que el coste no suponga un sesgo a la hora de determinar la solución adecuada.

Tradicionalmente las grandes entidades han basado su negocio en motores relacionales o RDBMS. Instalaciones, sobre todo en banca, dominadas por bases de datos DL/I y DB2, apoyados posteriormente con sistemas relacionales en sistemas medios. Se basaban casi exclusivamente en la transaccionalidad, delegando los procesos informacionales en procesos batch que se ejecutaban en entornos mainframe. Estos sistemas son extremadamente fiables y tienen un diseño especialmente robusto en ordenamiento y tratamiento de grandes ficheros. El problema: el coste de licencias y el escalado, que solo era posible de forma vertical invirtiendo en más recursos (núcleos de procesador, memoria y discos).

Desde 2008 y de forma exponencial, ha crecido tanto el volumen de información a procesar como la necesidad de resolver problemas no relacionados directamente con el negocio core de las compañías: detección de fraude, marketing, oferta personaliza a clientes, uso de aplicaciones nativas, integración IoT o pagos por móvil. Ante la tremenda competencia, surge la obligación de procesamiento en tiempo real o near real time de los datos.

Todo lo anterior supuso la necesidad de soluciones Big Data, comenzando a proliferar el uso de BBDD no relacionales (NoSQL) con modelos desestructurados y, sobre todo, con una gran capacidad para escalar horizontalmente de forma más económica, además de resultar soluciones con una fácil migración a la nube.

Parte importante del negocio pasó a estar en el dato y el conocimiento. Es habitual que todas las entidades manejen y crucen su información en datalakes, generando sus procesos informacionales y de reporting, desplazando trabajos pesados en Batch corriendo en mainframe por modelos de aprendizaje automático y cuadros de mando avanzados. Aún no es masivo el uso de IA en los procesos, pero todas las entidades ya disponen de un CDO y equipos especializados en la explotación de modelos.

Del mismo modo, la tremenda competitividad del de proveedores cloud (Amazon, Azure, Google Cloud) y el abaratamiento de la memoria y el espacio en disco han contribuido a cambiar poco a poco este escenario, así como un aumento de implantación de soluciones OpenSource, buscando un ahorro en costo de licencia. En el sector bancario, la irrupción de bancos completamente digitales (NeoBancos) y Fintechs son un ejemplo de palanca para que todas estas tecnologías tomen impulso.

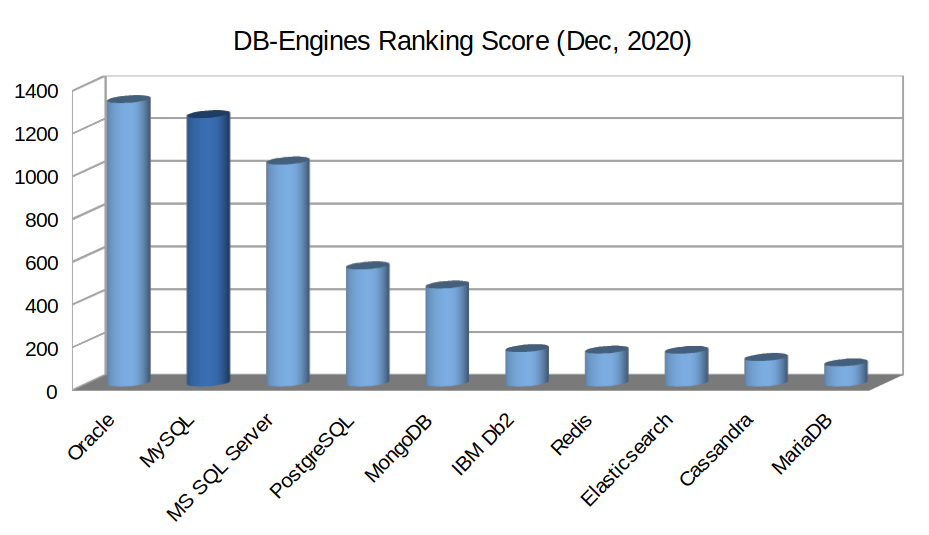


Ilustración . Ranking de las BBDD más demandadas en 2020. DB-Engines <https://towardsdatascience.com/top-10-databases-to-use-in-2021-d7e6a85402ba>

El grafico anterior muestra el ranking de las tendencias de productos más utilizados en BBDD a finales de 2020. La lista se ordena según encuestas y criterios de profesionales, así como popularidad o uso en grandes proyectos. Nos basaremos en este ranking para la selección de productos a evaluar, siempre que cumplan el criterio de ser productos de código abierto. De ellos, detallaremos sus características categorizándolos según su modelo de datos.

## Bases de datos relacionales SQL

En este apartado se analizan motores de BBDD capaces de ejecutar consultas en lenguaje estructurado SQL. Nacidas como evolución de las bases de datos jerárquicas y del modelo en red (norma Codasyl), siguen siendo la opción más utilizada.

* **MySQL**

RDMBS con licencia GPL creado en 1995 por Michael Widenius y David Axmark.

Desarrollado con enfoque de robustez y estabilidad. Posee un alto grado de madurez y penetración en la industria. Es la mas representativa muestra de BBDD SQL de tipo Opensource, utilizada por compañías líderes como Facebook o Uber. En sus últimas versiones es muy versátil, al permitir también contenido semiestructurado, lo que coloca al producto como solución todoterreno tanto para sistemas OLTP como OLAP.

Ofrece consistencia y garantiza el patrón ACID transaccional mediante el módulo InnoDB (equivalente a la instrucción BEGIN TRANSACTION-END TRANSACTION de productos como SQL Server o COMMIT de Oracle), así como Sharding mediante el módulo Shared Nothing MySQL cluster.

* **Alternativas:** Postgres

## Bases de datos NoSQL

A raíz del aumento masivo de información a procesar y la naturaleza desestructurada de la misma propias del Big Data, surgieron desafíos de negocio con continuos flujos de datos variados (redes sociales, e-commerce…), requiriendo de soluciones con procesamiento distribuido y alto rendimiento, favoreciendo la alta disponibilidad del dato pero comprometiendo la consistencia, pasando de la búsqueda de aplicaciones que cumplieran las propiedades ACID y “resultando en aplicaciones BASE (real readily available smooth-state choices with Eventual-consistency)” (Correlating NoSQL Databases With a Relational Database: Performance and Space, 2018)

Existen diferentes modelos de datos aplicables:

### Tipo clave-valor

Los elementos se almacenan siguiendo un patrón clave-valor (ejemplo duplas DNI-Nombre y apellidos). Es posible almacenar datos más complejos en el valor, pero sin capacidades de consulta avanzada como en SQL. Son muy rápidas en la búsqueda por clave, pudiendo ser o bien sintética, a elección del diseñador, o bien autogenerada. La transaccionalidad se delega en la aplicación.

No fueron diseñadas para un buen rendimiento en consultas de partes del dato o en operaciones de actualización. Su limitada capacidad de modelado hacen que sean desestimadas en este estudio.

* **Redis**

Creada en 2009 por  [Salvatore Sanfilippo](https://twitter.com/antirez), es una base de datos distribuida en memoria, soporta varios tipos de estructuras de datos, lo que le convierte en muy buena opción para aplicaciones de cache. Tiene licencia BSD 3-Clause y soporta replicación y sharding mediante el Cluster Redis.

* **Alternativas:** Etcd, EhCache, Riak KV

### Orientadas a documentos

Similar al patrón clave -valor, difiriendo en que el valor es un objeto complejo semiestructurado (XML, JSON, BSON) denominado documento.

Han tenido mucha acogida y permiten operaciones de consulta complejas (aunque sin el nivel de complejidad de SQL).

* **MongoDB**

Surgida en 2009, fue la primera de base de datos de tipo documento. Ofrece escalabilidad horizontal (sharding) con réplica de nodos primarios y secundarios. Tiene una aproximación al patrón ACID mediante Snapshots. Dispone de un potente lenguaje de consulta usando pipelines de agregación y capacidades básicas para problemas map-reduce.

* **Alternativas:** DocumentDB, CouchDB, MarkLogic

### Basadas en grafos

Orientadas a problemas donde los elementos almacenados tienen un alto grado de relación entre ellos en forma de grafo, donde. Ejemplo de ellos son aplicaciones de redes sociales. Por su impacto en la industria actualmente, no son de consideración en el estudio.

* **Principales productos** Neo4J, ArangoDB

### Columnares

Se trata de motores de base de datos que siguen un modelo orientado a columna, donde de forma similar a las RMDBS, cada columna de una tabla almacena un dato, pero en este tipo de productos, cada columna puede tener como dato otras columnas.

Las lecturas y escrituras se realizan sobre toda la columna, al contrario de sobre las filas como las relacionales. Los RMDBS almacenan físicamente la información de una fila como bytes de forma secuencial en disco, mientras que las orientadas a columna almacenan toda la columna de forma secuencia. Si se diseñan con criterio, suelen ser más rápidas en operaciones de lectura y en operaciones que requieran de agregación sobre los datos (cálculo de medias, sumatorios, conteos, máximos, mínimos o desviación estándar, por ejemplo).

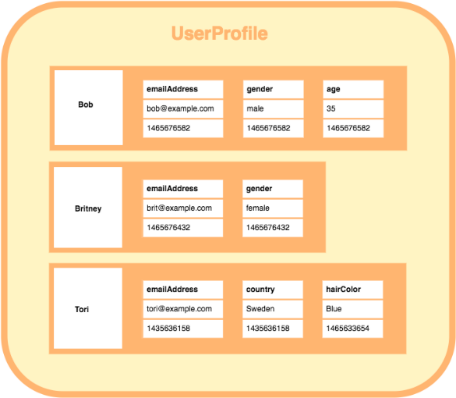


Ilustración Ejemplo de estructura BBDD columnar (<https://database.guide/nosql-database-types/>)

* **Cassandra**

Proyecto de código abierto mantenido por Apache, es un DBMS NoSQL distribuido. Diseñado para manejar altos volúmenes de información en servidores distribuidos ofreciendo soporte masivo para clusters.

Entre sus principales características: Soporte de replicación en multidatacenter, funciones de map-reduce para gestión de operaciones agregadas y dispone de un lenguaje propio denominado CQL (Cassandra Query Language) como alternativa a SQL en consultas más complejas

* **Alternativas:** Hbase

## Bases de datos NewSQL

Desde hace unos años están apareciendo nuevos motores de bases de datos SQL, escalables y de alto rendimiento denominadas NewSQL, término que acuño por primera vez Matthew Aslett en 2011 que detallo posteriormente como “nueva clase de sistemas de gestión de bases de datos (DBMS) […] con capacidad para escalar cargas de trabajo modernas de procesamiento de transacciones en línea (OLTP) de una manera que no es posible con sistemas legacy.” (What's Really New with NewSQL?, 2016).

También se denominan ScalableSQL para diferenciarlos de los productos de bases de datos relacionales existentes. El objetivo es dotar de capacidades transaccionales ACID y posibilidad de realizar consultas complejas en bases de datos con la potencia de soluciones distribuidas a las que aplica los principios CAP.

No es del alcance de este estudio estos productos, debido a su aún baja adopción por parte de proyectos de envergadura.

* **Principales productos:** ClustrixDB, NuoDB, CockRoachDB, MemSQL.

## Comparativa NoSQL respecto a CAP

Como se ha comentado previamente, el coste del alto rendimiento y el tratamiento de datos masivo y multiservidor provoca limitaciones detalladas en el teorema CAP, dejando atrás conceptos como consistencia estricta por consistencia eventual, por ejemplo. “Incluso sin las consideraciones del teorema CAP, las transacciones ACID eran cada vez más insostenibles en sitios web distribuidos a gran escala. Esto se relaciona más con el rendimiento que con la disponibilidad” (Harrison, 2015)

De los sistemas seleccionados que se utilizarán para las pruebas de este estudio, esta es su relación respecto al teorema CAP.

Por un lado, tenemos a MySQL como sistema SQL relacional, que asegura transacciones ACID t por tanto cumple con las propiedades CAP de consistencia estricta y disponibilidad en cuanto el dato leído en cualquier momento es fiable. Sin embargo, un problema de comunicación entre nodos provocaría indisponibilidad.

Otra agrupación identifica a Redis y MongoDB como sistemas muy orientados a la escalabilidad horizontal y por tanto con un fuerte soporte a errores si cae algún nodo (tolerancia a la partición). Así mismo y dependiendo de la configuración de la instalación se prioriza o bien consistencia o bien disponibilidad.

Por último, Cassandra escribe datos en los nodos que estén activos, se seguirá dando servicio, pero si alguno ha perdido comunicación con el resto y se lee de él, no estaría cumpliéndose el principio de consistencia.

|  |  |  |
| --- | --- | --- |
| Producto | Modelo datos | Funciones CAP |
| Redis | Clave-valor | CP (Consistencia y tolerancia a la partición) |
| MongoDB | Documental | CP (Consistencia y tolerancia a la partición) |
| Cassandra | Columnar | AP (Disponibilidad y tolerancia a la partición) |
| MySQL | SQL | CA (Consistencia y disponibilidad) |

Tabla . Comparativa CAP (Elaboración propia)

Las nuevas bases de datos NewSQL parten de modelos relacionales, pero enfoque distribuido típico de las bases de datos NoSQL más avanzadas, lo que sobre el papel haría cumplir los tres principios del teorema.



# Definición del modelo de información

Se han seleccionado una muestra de entidades de negocio bancario como referencia de particionamiento de dominios según el estándar BIAN. El estándar BIAN no detalla cómo deben implementarse los modelos lógicos, con lo que nos basamos de inicio en modelos de datos adaptados por los recomendados en el estándar OpenBanking (basado así mismo en OpenAPI y OpenData).

El modelo lógico que relaciona las entidades (sin incluir los parámetros) y los dominios BIAN que implementan, se puede observar en el siguiente esquema:

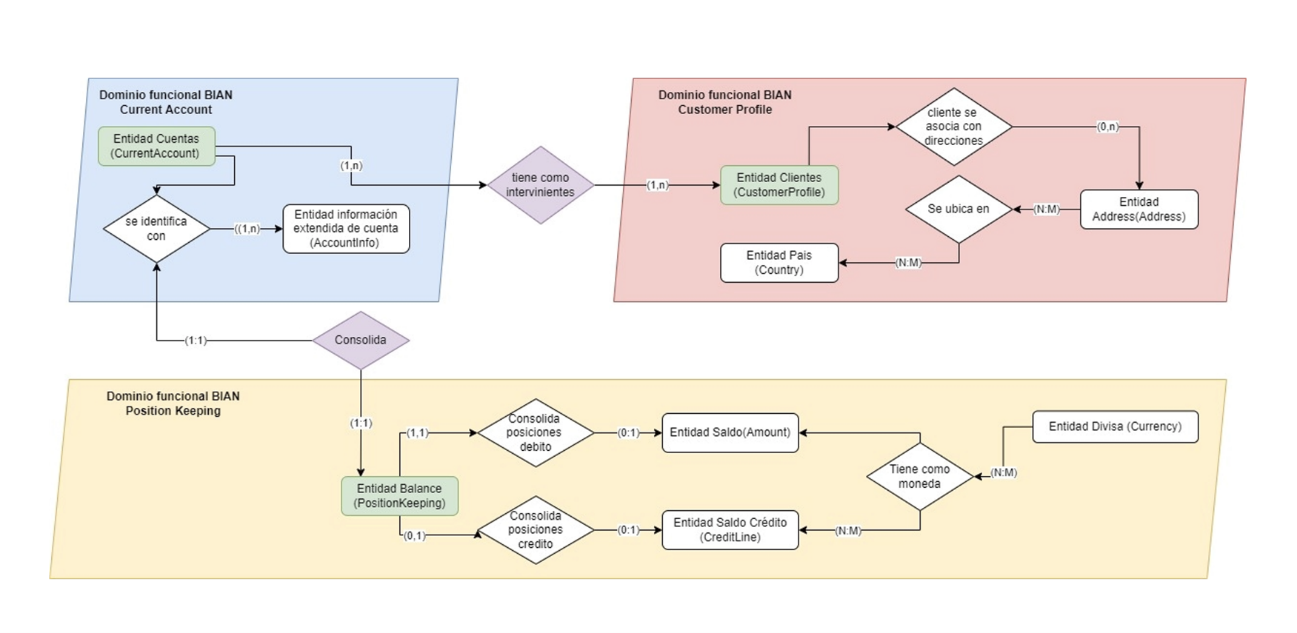


Ilustración 3 Diagrama modelo lógico (fuente. Elaboración propia)

Estas entidades son simplificadas y trasladadas a un modelo E/R inicial tomando como ejemplo la definición de recursos OpenBanking en su versión 3.1.5:

* Accounts <https://openbankinguk.github.io/read-write-api-site3/v3.1.5/resources-and-data-models/aisp/Accounts.html>)
* Balances <https://openbankinguk.github.io/read-write-api-site3/v3.1.5/resources-and-data-models/aisp/Balances.html>)
* Party <https://openbankinguk.github.io/read-write-api-site3/v3.1.5/resources-and-data-models/aisp/Parties.html>).

## Casos de uso de pruebas

Se han definido los casos de uso que serán objeto de pruebas, siempre con acciones en bases de datos que involucren a varios dominios funcionales:

* **Consulta** de todos los atributos de cuentas (*CurrentAccount*), saldo (*PositionKeeping.[Amount].Amount*) de cada una de ellas y listado de intervinientes relacionados (*CustomerProfile.PartyId* y *CustomerProfile.FullLegalName*).
* **Modificación de agregado de varios dominios**: Modificación del campo “*Currency\_Code*” (código de divisa) a valor *“GBP”* de todos los balances (*PositionKeeping.CreditLine.Currency.Code* y *PositionKeeping.Amount.Currency.Code*) de cuentas y balance cuyo campo *CurrentAccount.[AccountInfo].SchemeName* cuyas primeras tres posiciones del registro sean “*UK.*”.
* **Inserción de datos con distribución entre varios dominios**: Nuevo cliente y sus direcciones (*CustomerProfile y CustomerProfile.[Address]*) y alta simultanea de cuenta nueva (*CurrentAccount*) además de un ingreso mínimo inicial en saldo debito euros (*PositionKeeping*).
* **Cálculo de funciones de agregación**: máximo (**MAX**) y media (**AVG**) sobre campo numérico Amount en los registros de crédito (*PositionKeeping.CreditLine.Amount*) de cuentas activas (*CurrentAccount.Status* sea. “*Enabled*)”.

Las casuísticas de prueba que impliquen modificación o inserción de datos entre varios dominios, serán tratados como una transacción atómica.

Dado que los modelos de datos han sido diseñados para mantener una separación lógica y física entre dominios (esquemas, colecciones y keyspaces), es necesario coordinar esta transacción mediante la capa de aplicación.

## Definición de entidades relacionales

En el modelo relacional y con el fin de simular un entorno de datos separado por dominios tanto en su forma lógica como física, se definen tres esquemas diferentes de datos, coincidiendo en nombre con los dominios BIAN (*CurrentAccountDomainSchema, CustomerProfileDomainSchema y Position KeepingDomainSchema*).

La definición de productos contratados por un cliente de tipo cuenta se gestionan, según BIAN, mediante el dominio funcional “*Current Account*” (<https://bian.org/semantic-apis/current-account/>). El atributo AccountId actúa como clave primaria y se utiliza una versión simplificada de la entidad OBReadAccount5 de OpenBanking para este estudio. Hay una referencia a PartyId como identificador del dominio de clientes ([ver diccionario de datos](#_OBReadAccount5)). Se define una relación con entidad AccountInfo donde residen datos descriptivos de una cuenta y su segmentación según el estándar en el atributo SchemeName.

BIAN describe el dominio funcional Customer Profile (<https://bian.org/semantic-apis/customer-profile/>). Este dominio engloba los datos personales del cliente y datos relacionados como sus diferentes direcciones de residencia y contacto. El atributo PartyId actúa como clave única y primaria y se utiliza en este trabajo una definición adaptada de OBParty2 de OpenBanking ([ver diccionario de datos](#_OBParty2)) como implementación base de esta entidad. Se normaliza el listado de países y sus códigos internacionales en la entidad Country para dar profundidad al estudio.

Cada cuenta tiene un balance contable, que según BIAN es responsabilidad del dominio funcional Position Keeping (<https://bian.org/semantic-apis/position-keeping/>) y que para su implementación se basa en el recurso del estándar OpenBanking “Balance”. AccountId se establece como clave foránea de la relación entre cuentas y el balance de estas. El atributo Type define si son balances de crédito o débito y dependiendo de ello el saldo se describe o por la entidad Amount (debito) o por la entidad CreditLine (crédito).

Para ampliar la profundidad del modelo de datos y que esto enriquezca el estudio, se normaliza la tabla de divisas (Currency) con descripción y código internacional de las mismas.

La implementación E/R para las entidades queda definida de la siguiente forma separada por los tres esquemas independientes:

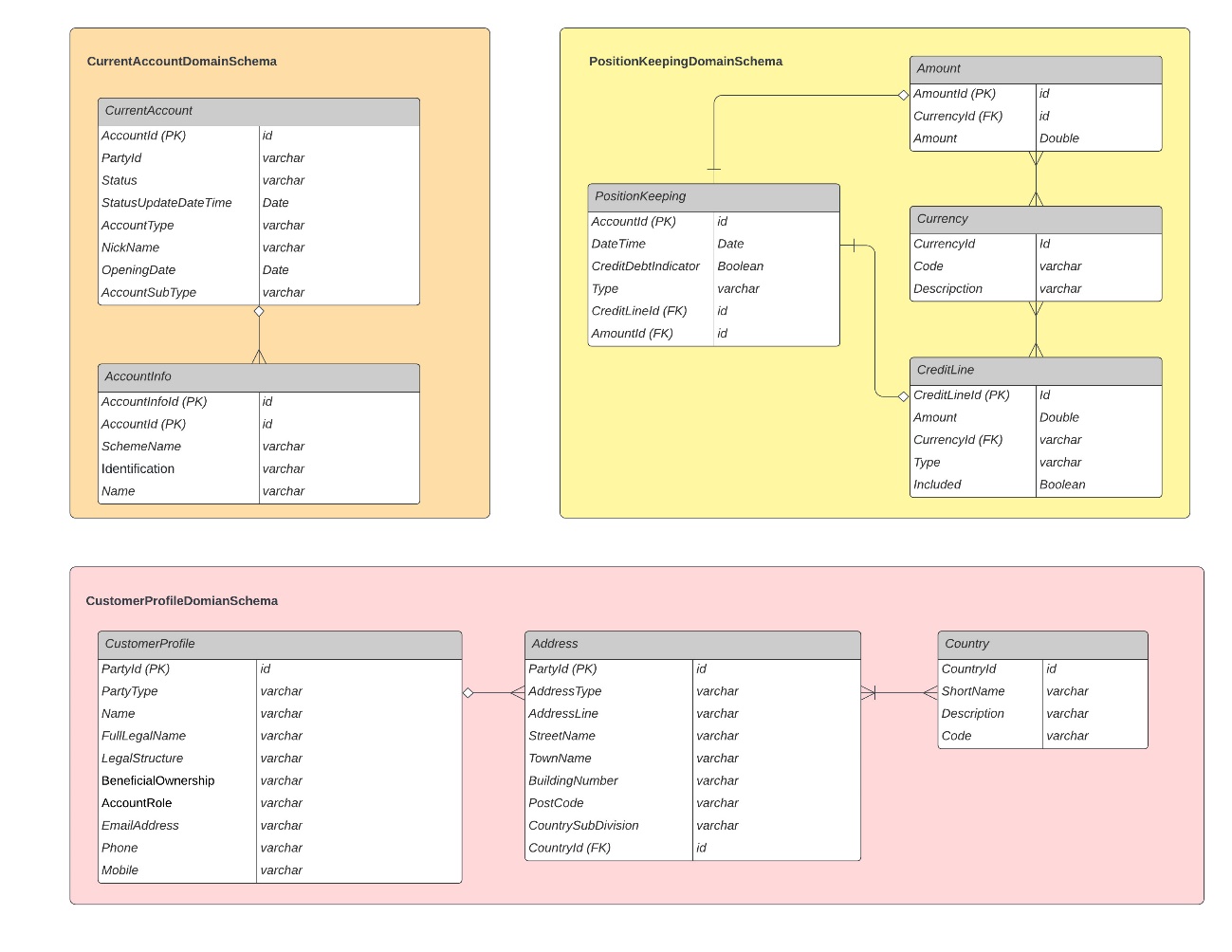


Ilustración 4. Diagrama E/R por dominios funcionales. (fuente. [Elaboración](https://openbankinguk.github.io/read-write-api-site3/v3.1.5/resources-and-data-models/aisp/Accounts.html) propia)

## Definición del modelo de datos documental

La forma adecuada de organizar este modelo para lograr la máxima eficiencia en bases de datos documentales, seria agrupar todos los elementos en un único documento y una única colección, donde el identificador sea AccountId y disponga de subdocumentos embebidos con la información de balances (Position Keeping) y datos del cliente (Customer Profile), así como el resto de entidades:

{    //Current Account

    "\_id": "31820", //AccountId

    "Status": "Enabled",

    "StatusUpdateDateTime": "2018-01-01T06:06:06+00:00",

    "AccountType": "Personal",

    "AccountSubType": "CurrentAccount",

    "Nickname": "Household",

    "AccountInfo": [

        {

            "SchemeName": "UK.OBIE.SortCodeAccountNumber",

            "Identification": "80200110203348",

            "Name": "Mr Kevin" },

        {   "SchemeName": "UK.OBIE.IBANAccountNumber",

            "Identification": "UK30802001102033489987",

            "Name": "Mr Kevin for external" } ],

    //Position Keeping

      "CreditDebitIndicator": "Credit",

      "Type": "InterimAvailable",

      "DateTime": "2017-04-05T10:43:07+00:00",

      "Amount": {

        "Amount": "300.00",

        "Currency": {"code" : "GBP", "description": "Libras esterlinas"} },

      "CreditLine": {

          "Included": false,

          "Amount": {

            "Amount": "500.00",

            "Currency": {"code" : "GBP", "description": "Libras esterlinas"} },

          "Type": "Available"},

     //Customer profile

    "PartyId": "PABC123",

    "PartyType": "Sole",

    "Name": "Semiotec",

    "FullLegalName": "Semiotec Limited",

    "LegalStructure": "UK.OBIE.PrivateLimitedCompany",

    "BeneficialOwnership": true,

    "AccountRole": "UK.OBIE.Principal",

    "EmailAddress": "contact@semiotec.co.jp",

    "Address": [

{

            "AddressType": "Business",

            "StreetName": "Street",

            "BuildingNumber": "15",

            "PostCode": "NW1 1AB",

            "TownName": "London",

            "Country": { "Code": "GB", "ShortName": "GreatBrit", "Description": "Gran Bretiania" } },

        {

            "AddressType": "Particular",

            "StreetName": "Street",

            "BuildingNumber": "14",

            "PostCode": "NW1 1AB",

            "TownName": "London",

            "Country": { "Code": "GB", "ShortName": "GreatBrit", "Description": "Gran Bretiania" }} ] }

Teniendo en cuenta el requisito de separación por dominios tanto en forma física como lógica del problema que tratamos, es necesario dividir los datos en colecciones MongoDB independientes, donde cada una de ellas haga referencia a los diferentes ámbitos o dominios funcionales (ver Tabla 2. Trazabilidad entidades modelo documental (fuente. Elaboración propia)). Cada documento serán objetos JSON como los de estos ejemplos agrupados por sus colecciones correspondientes:

**Collection CurrentAccount – Ejemplo documento**

{    //Current Account

    "AccountId": "31820", //AccountId

"PartyId": “PABC123”, //Referencia al Customer Profile (Party)

    "Status": "Enabled",

    "StatusUpdateDateTime": "2018-01-01T06:06:06+00:00",

    "AccountType": "Personal",

    "AccountSubType": "CurrentAccount",

    "Nickname": "Household",

    "AccountInfo": [

        {

            "SchemeName": "UK.OBIE.SortCodeAccountNumber",

            "Identification": "80200110203348",

            "Name": "Mr Kevin" },

        {   "SchemeName": "UK.OBIE.IBANAccountNumber",

            "Identification": "UK30802001102033489987",

            "Name": "Mr Kevin for external" }

]

}

**Collection PositionKeeping – Ejemplo documento**

{//Position Keeping

      "AccountId": "31820", //AccountId

"CreditDebitIndicator": "Credit",

      "Type": "InterimAvailable",

      "DateTime": "2017-04-05T10:43:07+00:00",

      "Amount": {

        "Amount": "300.00",

        "Currency": {"code" : "GBP", "description": "Libras esterlinas"}

      },

      "CreditLine": {

          "Included": false,

          "Amount": {

            "Amount": "500.00",

            "Currency": {"code" : "GBP", "description": "Libras esterlinas"} },

          "Type": "Available"}

}

**Collection CustomerProfile – Ejemplo documento**

{  //Customer profile

    "PartyId": "PABC123",

    "PartyType": "Sole",

    "Name": "Semiotec",

    "FullLegalName": "Semiotec Limited",

    "LegalStructure": "UK.OBIE.PrivateLimitedCompany",

    "BeneficialOwnership": true,

    "AccountRole": "UK.OBIE.Principal",

    "EmailAddress": "contact@semiotec.co.jp",

    "Address": [

        {

            "AddressType": "Business",

            "StreetName": "Street",

            "BuildingNumber": "15",

            "PostCode": "NW1 1AB",

            "TownName": "London",

            "Country": { "Code": "GB", "ShortName": "GreatBrit", "Description": "Gran Bretiania" }

        },

        {

            "AddressType": "Particular",

            "StreetName": "Street",

            "BuildingNumber": "14",

            "PostCode": "NW1 1AB",

            "TownName": "London",

            "Country": { "Code": "GB", "ShortName": "GreatBrit", "Description": "Gran Bretiania" }

        }

]

}

Existirán por tanto documentos y listados de documentos embebidos: AccountInfo y Address (relaciones *one-to-many*) o Country y Currency (*many-to-many*). Estas agregaciones no comprometen la separación por ámbitos de dominio funcionales al ser información relacionada con los artefactos BIAN con los que estamos trabajando.

## Definición del modelo de información columnar

En una definición estándar de modelo columnar, se recomienda siempre diseñar el modelado respecto a las ejecuciones sobre los datos que deseamos consultar para obtener de este modo el mejor rendimiento y las aprovechar las ventajas de la tecnología ([ver casos de uso de pruebas](#_Casos_de_uso)).

Es necesaria una correcta estrategia de Partitions Key y Clustering Keys tanto para optimizar el rendimiento en distribución de los datos en los nodos del cluster como para el acceso a disco de las consultas. Es una buena practica orientar las consultas a obtener una única partición (clave primaria), teniendo en consideración que Cassandra tiene una limitación de filas por partición (2000 millones), además de un descenso pronunciado en rendimiento cuanto mas altos son estos valores. Esto también afecta críticamente al espacio en disco necesario.

En instalaciones habituales de bases de datos columnares es buena práctica replicar datos en diferentes tablas y columnas, definiendo específicamente tablas para consultas repetitivas, todo ello en un único keySpace para optimizar la distribución en los nodos. Por ejemplo, para cubrir las necesidades de los [casos de uso de negocio del análisis](#_Casos_de_uso_1) en un entorno no orientado a dominios, podríamos definir tablas de la siguiente forma:

* **Tabla maestra de cuentas, clientes y saldos (almacén de información no orientado a consultas)**

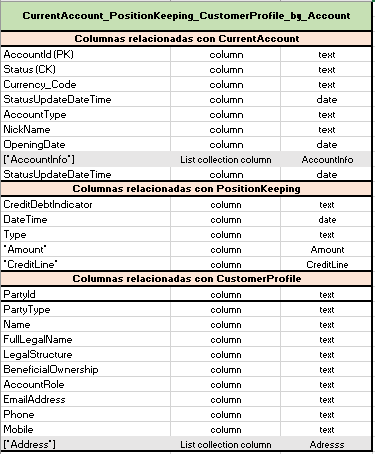


Ilustración . Diagrama modelo datos columnar datos maestros (fuente. Elaboración propia)

* **Tablas de tipos de datos**

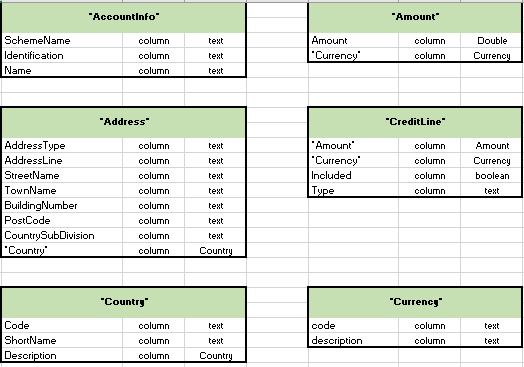


Ilustración . Diagrama modelo datos columnar tipos de dato (fuente. Elaboración propia)

* **Tabla orientada a consulta: Consulta de cuentas, saldo e intervenientes agrupados por moneda**

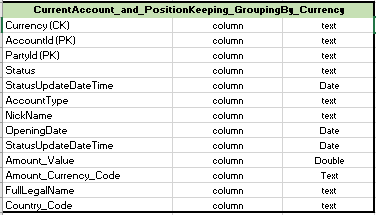


Ilustración . Tabla columnar por consulta de moneda (fuente. Elaboración propia)

* **Tabla orientada a consulta: Cálculo de funciones de agregación máximo (MAX) y media (AVG) sobre saldo tipo debito para cuentas activas.**

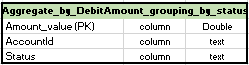


Ilustración . Tabla columnar por consulta de agregados de importe (fuente. Elaboración propia)

Los tipos de datos de usuario definidos en las tablas del modelo columnar que aparecen en este documento, se representan con \* (asterisco). Estos tipos de datos propios no se definen con claves primarias ni de clustering.

La replicación de datos en tablas adaptadas a las consultas conlleva la implementación de otros mecanismos de sincronización y coherencia del dato. Estos mecanismos requieren de una infraestructura compleja para asegurar la integridad del dato. En los entornos bancarios a los que se orienta este estudio es de vital importancia una sincronización atómica en tiempo real.

En nuestro problema a resolver, cada dominio de información es responsable de sus propios datos debiéndose separar la información en keyspaces diferentes. En instalaciones reales bancarias estos datos no solo están separados por dominios lógicos sino en muchas ocasiones también en servidores o incluso empresas diferentes.

Se definen tipos de usuario para el equivalente a las tablas CreditLine, Address, AccountInfo, Currency y Amount del modelo relacional y , a su vez, documentos embebidos en el modelo documental.

Para mantener coherencia con la disposición definida en modelo documental y relacional, nuestro modelo de información queda distribuido de este modo:

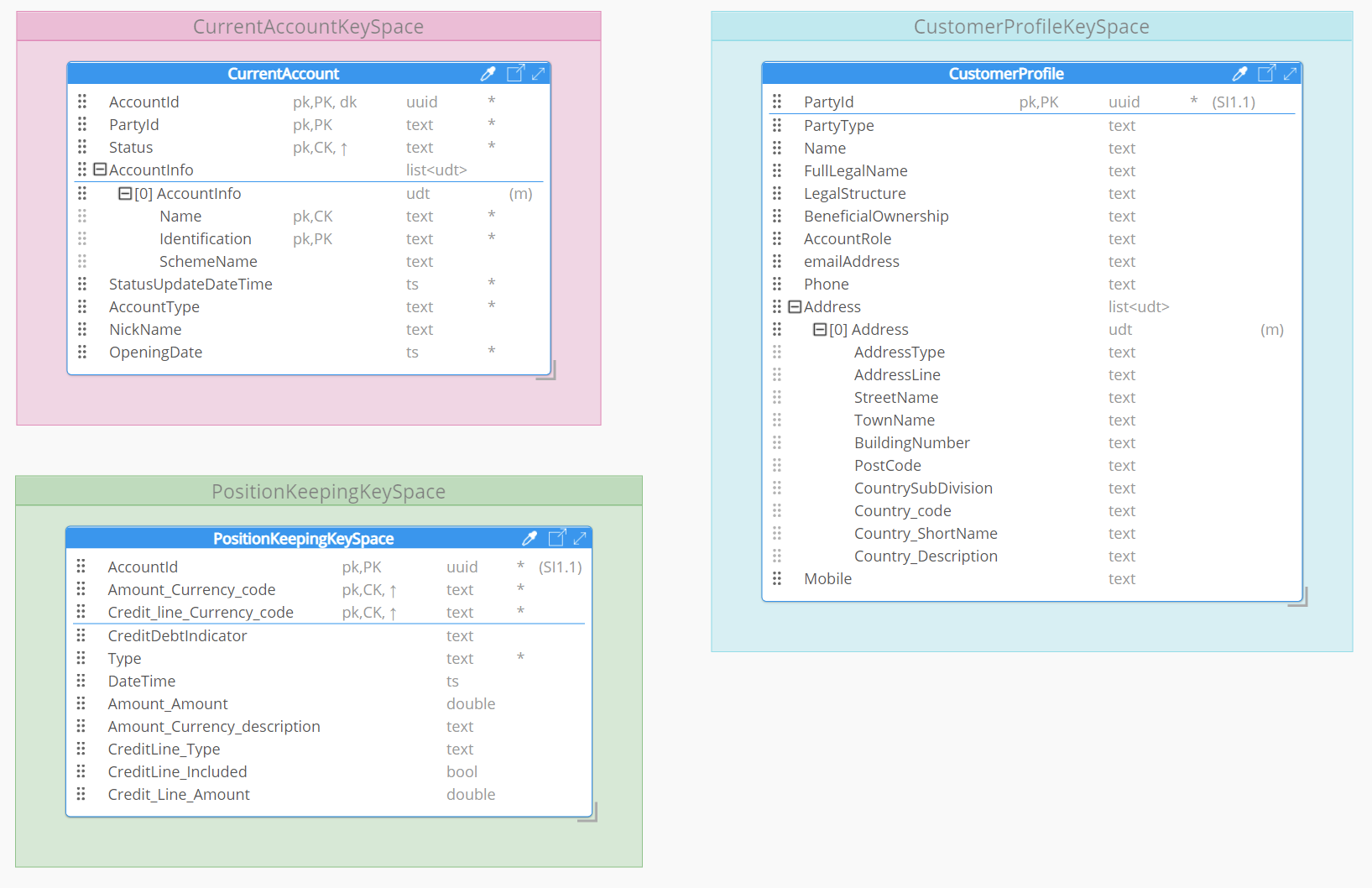


Ilustración . Diagrama modelo datos columnar por dominios (fuente. Elaboración propia)

Se configuran los keySpaces con un factor de réplica a 1 y una estrategia de ubicación de datos simple (SimpleStrategy). Las tablas se crean con la configuración por defecto para los atributos de cache.

CREATE KEYSPACE CurrentAccountKeySpace WITH REPLICATION = {'class': 'SimpleStrategy',

'replication\_factor': 1};

CREATE KEYSPACE PositionKeepingKeySpace WITH REPLICATION = {'class': 'SimpleStrategy',

'replication\_factor': 1};

CREATE KEYSPACE CurrentProfileKeySpace WITH REPLICATION = {'class': 'SimpleStrategy',

'replication\_factor': 1};

En cualquier caso, debido a la tecnología de Cassandra, ninguno de los datos entre KeySpace mantendrá una integridad referencial entre sí, de modo que en actualizaciones y borrados será necesario controlar este detalle mediante la capa aplicativa. Existe la opción de implementar el concepto de *batch* en Cassandra para agrupar inserciones y modificaciones en grupos de transacciones, pero no es aconsejado en las buenas practicas del producto, por la sobrecarga de trabajo a los nodos, quedando fuera de este trabajo.

Las siguientes tablas resumen la trazabilidad de entidades, con sus estructuras lógicas referentes, así como los dominios BIAN relacionados, el objeto OpenBanking correspondiente (el digito final de la entidad OpenBanking hace referencia a la versión del diccionario de datos del estándar) y otros aspectos destacados según la tecnología.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Entidad | | Dominio BIAN | | Entidad OpenBanking | | Esquema | | Claves primarias y foraneas |
| CurrentAccount | | Current Account | | OBReadAccount5 | | currentAccountDomainSchema | | AccountId (PK),PartyId (FK) |
| AccountInfo | | Current Account | | OBReadAccount5 | | currentAccountDomainSchema | | AccountInfo (PK), AccountId (PK) |
| PositionKeeping | | Position Keeping | | OBReadBalance1 | | positionKeepingDomainSchema | | AccountId (PK), AmountId (FK), CreditLineId (FK) |
| Amount | | Position Keeping | | OBReadBalance1 | | positionKeepingDomainSchema | | AmountId (PK) , CurrencyId (FK) |
| Currency | | Position Keeping | | Normalización OBReadBalance1 | | positionKeepingDomainSchema | | CurrencyId (PK) |
| CreditLine | | Position Keeping | | OBReadBalance1 | | positionKeepingDomainSchema | | CreditLineId (PK) , CurrencyId (FK) |
| CustomerProfile | Customer profile | | OBParty2 | | customerProfileDomainSchema | | PartyId (PK) | |
| Address | Customer profile | | OBParty2 | | customerProfileDomainSchema | | PartyId (PK), CountryId (FK) | |
| Country | Customer profile | | Normalización OBParty2 | | customerProfileDomainSchema | | CountryId (PK) | |

Tabla 3. Trazabilidad entidades modelo relacional (fuente. Elaboración propia)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Entidad | Dominio BIAN | Entidad OpenBanking | Colección Documental | Identificador (\_id) |
| Account | Current Account | OBReadAccount5 | currentAccount | AccountId |
| Balance | Position Keeping | OBReadBalance1 | positionKeeping | AccountId |
| Party | Customer profile | OBParty2 | customerProfile | PartyId |

Tabla 4. Trazabilidad entidades modelo documental (fuente. Elaboración propia)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Entidad | Dominio BIAN | Entidad OpenBanking | ColumnFamily | Keyspace | Partition key | Clustering key |
| CurrentAccount | Current Account | OBReadAccount5 | currentAccount | CurrentAccountKeySpace | AccountId, PartyId | Status |
| Balance | Position Keeping | OBReadBalance1 | positionKeeping | PositionKeepingKeySpace | AccountId | Amount\_Currency.Code, CreditLine\_Currency\_code |
| Party | Customer profile | OBParty2 | customerProfile | CustomerProfileKeySpace | PartyId | N/A |

Tabla 5. Trazabilidad entidades modelo columnar (fuente. Elaboración propia)

# Metodología y entornos de pruebas

Se definen instancias de las bases de datos seleccionadas para el estudio con las siguientes versiones: MySql 8.X, MongoDB 4.X y Cassandra 3.X.

El objetivo del estudio es comprobar principalmente el rendimiento respecto a velocidad en que se ejecutan las consultas y transacciones, así como el uso de memoria.

Todos los clústeres de las bases de datos se despliegan mediante contenedores Docker. En el caso de MySQL se realiza una instalación de cluster InnoDB en modo sandbox desplegada en la plataforma de test con 3 nodos en grupos de replicación.

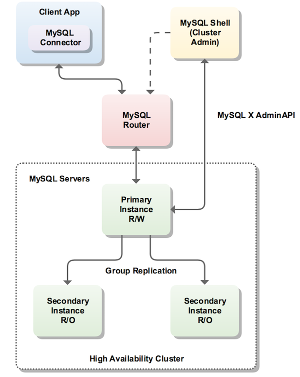
.

Ilustración . Arquitectura cluster InnoDB MySQL sandbox (fuente. <https://github.com/wwwted/MySQL-InnoDB-Cluster-local-sandbox>)

La instalación del cluster MongoDB se realiza igualmente en la misma plataforma de test, mediante una arquitectura de replica set con 3 nodos, estableciéndose un quorum de réplica y coherencia de dato a dos ([commitQuorum](https://docs.mongodb.com/manual/reference/method/db.collection.createIndex/" \l "std-label-createIndex-method-commitQuorum) = 2).

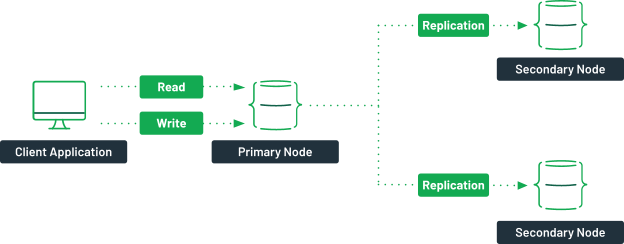


Ilustración . Arquitectura cluster de pruebas MongoDB (fuente. <https://www.mongodb.com/basics/clusters/mongodb-cluster-setup>)

En el caso de Cassandra el valor de estrategia de réplica se configura como “simpleStrategy”, con escrituras durables (durable writes) activadas para los tres KeySpaces que se definen en el modelo.

La arquitectura que se ha diseñado para ejecutar las pruebas se basa en una capa de aplicación con endpoints HTTP para el acceso a datos agrupados en un modelo de microservicio, de forma análoga a como se expone la información en arquitecturas hexagonales.

Para el modelado relacional se ha utilizado la herramienta [MySQL Workbench](https://www.mysql.com/products/workbench/). De esta herramienta se han generado los ficheros DDL de creación de la estructura de la base de datos. Tanto los scripts, como el proyecto del workbench, pueden encontrarse en el repositorio del proyecto.

Para la definición del modelo columnar se ha utilizado la herramienta [Hackolade](https://hackolade.com/) generándose los ficheros CQL de definición de la base de datos Cassandra.

Para tener un juego de datos adecuado, se han definido datos sintéticos a partir del modelo relacional mediante la herramienta online [fillDB](http://filldb.info/). Esta herramienta tiene una limitación de registros generados a fichero csv, con lo que se ha recurrido a definir a un programa python encargado de simular más registros aleatorios basados en registros ya creados para cada entidad principal (un millón) a modo de dataset con la librería **pandas** y **numpy**. Se respetan las secuencias relativas a claves primarias. Para este estudio no es primordial una coherencia exhaustiva de los datos sino unas pruebas relacionadas con el volumen de los datos.

Estos ficheros, se adecuan mediante cuadernos de Jupyter Notebook para cada modelo (relacional, columnar y documental) según las especificaciones referidas en puntos anteriores, ya sea en un formato JSON o con una distribución diferente de elementos y atributos en ficheros csv.

Se han definido los siguientes segmentos de volúmenes de registros 10000, 100000 y 1000000. Se insertan en las bases de datos (dependiendo del volumen de registros a evaluar) mediante programas python también desarrollados en cuadernos Jupyter. Esos registros se corresponden con el número de clientes y de cuentas. Los parámetros variables como numero de direcciones asociadas a un cliente, detalles de una cuenta o posiciones de saldo, variarán de forma aleatoria en un numero razonable (1-5 registros).

La capa de aplicación se implementa con la librería [Flask](https://flask.palletsprojects.com/en/2.1.x/) de Python y como framework de test se utilizará la librería [Locust](https://locust.io/) ,orientada a test de carga ,desde la que se simularán las peticiones de usuarios en segmentos de 10, 100 y 1000 clientes concurrentes conectando con cada uno de los endpoints HTTP que acceden a las consultas de datos cada uno de los casos de negocio del estudio. Cada uno de estos endpoints accede a los conectores Python de las distintas tecnologías (mysql-connector-python, cdata.cassandra y PyMongo), delegando en este microservicio la orquestación de llamadas y cruce de información entre dominios.

Todos estos recursos pueden encontrarse en el [repositorio](https://github.com/danielherranzsegundo/TFM_2022) público del proyecto.

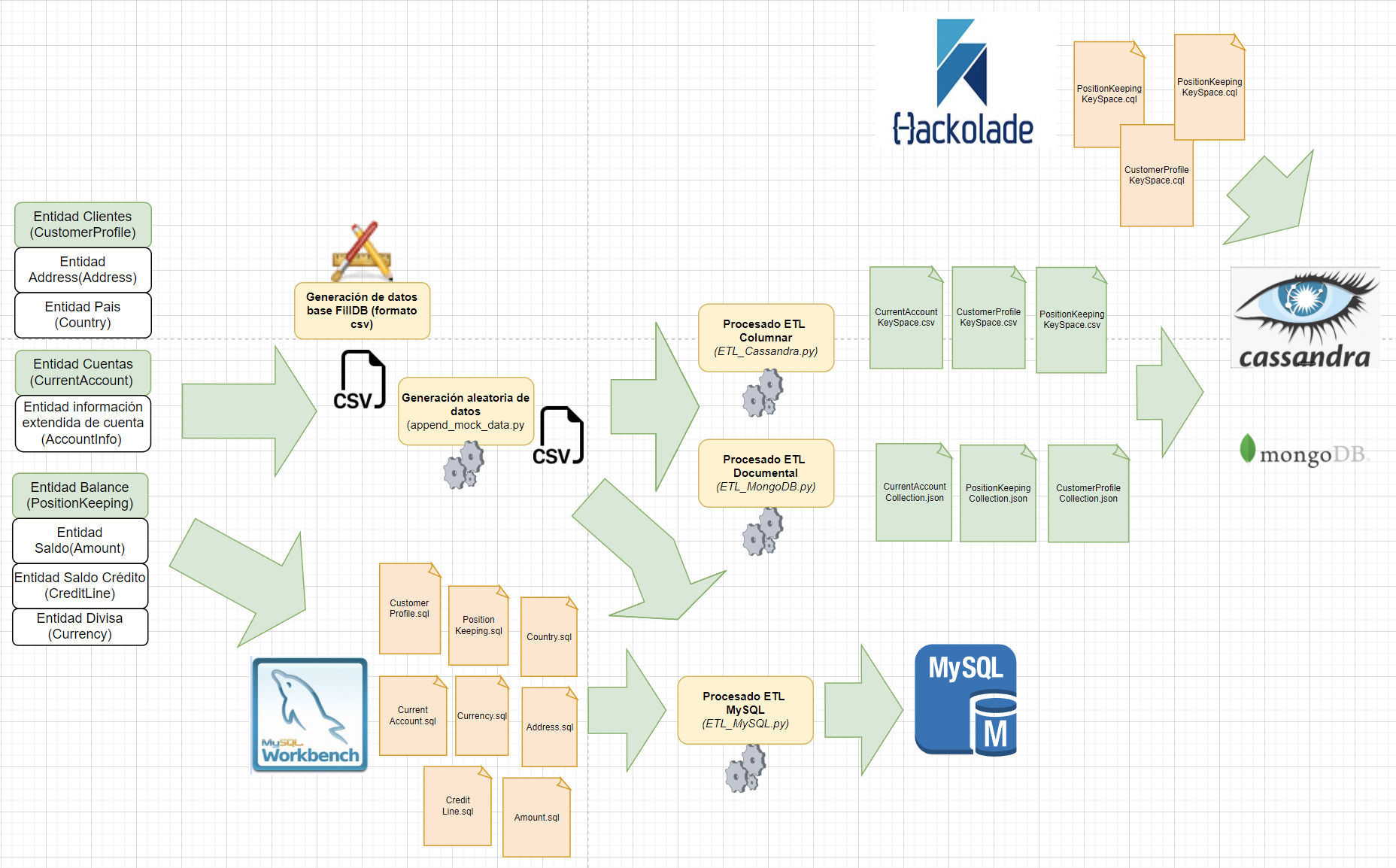


Ilustración . Modelo ETL del proyecto (fuente. Elaboración propia)

Los casos de negocio anteriormente listados serán implementados como consultas y transacciones de forma masiva con ejecuciones repetidas durante un minuto con una concurrencia en tres rangos 10,100 y 1000 peticiones, sobre la que se obtendrá la media (eliminando del cálculo los resultados en percentiles superiores a 95% e inferiores a 5%).

En la siguiente imagen podemos observar la arquitectura desplegada con los elementos que forman parte del ecosistema de pruebas:

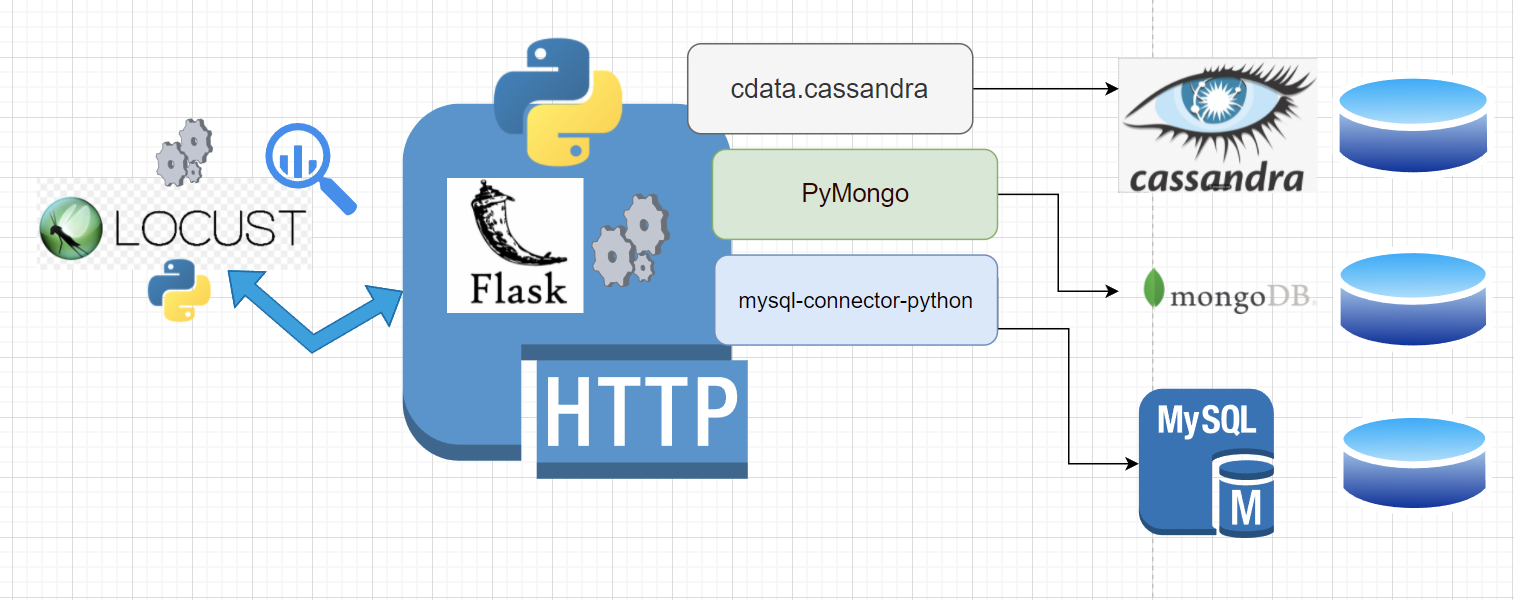


Ilustración 11. Arquitectura ecosistema pruebas (fuente. Elaboración propia)

Se ha elegido Python y Flask como tecnologías de integración ya que consumen pocos recursos del sistema, liberando estos para trabajos pesados (mayores volúmenes de datos y de peticiones concurrentes)



## Plataforma de test

El equipo standalone de test se compone de un Intel(R) Core(TM) i7-8700 CPU corriendo a 3.20GHz con 32GB de Random Access Memory (RAM) y 5TB de almacenamiento. El sistema operativo es Ubuntu 20.10, 64-bit.

Como gestor de contenedores se utiliza Docker. En todos los casos se utilizan redes virtuales por cada cluster con autenticación básica en todos los casos.

# Desarrollo del proyecto

Una vez definidos los diferentes modelos de información según la tecnología, se procede a la instalación de las diferentes instancias de los motores de bases de datos. Para más información del proceso consultar el [Anexo I](#_Anexos_I).

La siguiente tarea fue la definición del modelado físico de los datos. Para ello se ha utilizado la herramienta [MySql workbech](https://www.mysql.com/products/workbench/) para construir los scripts de lenguaje de definición de datos DDL de los diferentes objetos (esquemas, tablas, índices…) del modelo relacional. Para los modelos columnares y documentales, se ha hecho uso de la herramienta [Hackolade](https://hackolade.com/) en su versión Community, para la definición de los keySpaces, User-defined types y diferentes tablas del modelo columnar de Cassandra, así como los documentos ejemplo JSON correspondientes a cada una de las colecciones de MongoDB. Todos los recursos comentados pueden encontrarse en el [repositorio](https://github.com/danielherranzsegundo/TFM_2022) del proyecto.

# Resultados

# Conclusión y trabajos futuros

En lo personal este proyecto me ha ayudado a afrontar retos relacionados con diseños basados en los dominios y tener una capacidad de discusión con los diferentes actores que suelo encontrar en los proyectos tecnológicos en los que participo.

Las bases de datos solían ser un ámbito bastante estático, pero en los últimos años hay una gran cantidad y variedad de soluciones. Esto desgraciadamente ha provocado no tener que dejar fuera del alcance a algunas de las ultimas en aparecer.

El trabajo tiene una clara continuación en los modelos de datos no tratados. Por un lado, se debería expandir a bases de datos NewSQL que prometen soluciones ACID con las bondades de la escalabilidad de las NoSQL.

Serian así mismo necesarias mas pruebas sobre instalaciones en la nube, probando los distintos tipos de proveedores (AWS, Azure o Google Cloud). Ese análisis requeriría de mayores recursos de tiempo y coste. En estudios venideros, se recomienda profundizar aún mas en mecanismos avanzados de afinamiento del rendimiento de las bases de datos (índices compuestos, configuraciones de bloques de memoria, o cache de datos).

También podría ser de suficiente interés analizar los mismos escenarios aquí tratados en arquitecturas basadas en DIH (Digital Integration Hub), donde se desacopla el aplicativo y la explotación, de la capa de registro de datos. Esta arquitectura se basa computación on-memory y permite utilizar cualquier modelado de datos (SQL, NoSQL, NewSQL) incluso una combinación de varias de ellas. La computación on-memory asegura mejorar en cien veces el rendimiento de bases de datos tradicionales, eso es debido al uso de RAM como almacenamiento y el procesamiento en paralelo siendo una posible solución conjunta a problemas OLTP y OLAP analítica. Aún tecnologías incipientes pero que considero con bastante recorrido.

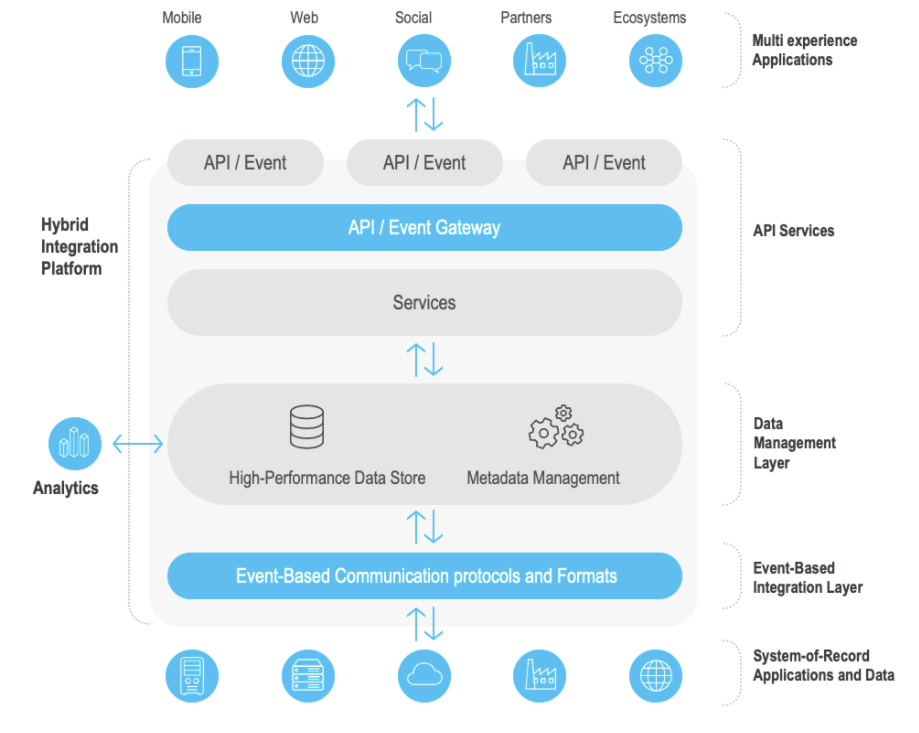


Ilustración . Arquitectura DIH (https://www.gigaspaces.com/blog/digital-integration-hub-dih)

# Referencias

Fowler M, & Sadalage P. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence* (Addison Wesley ed.). Addison Wesley.

Harrison, G. (2015). *Next Generation Databases.* Berkeley, CA: Apress.

Domdouzis K, Lake P, Crowther P (2013)*. Concise Guide to Databases. A Practical Introduction (Undergraduate Topics in Computer Science).* Springer

Pokorný J. (2015). *Database technologies in the world of big data*. <https://doi.org/10.1145/2812428.2812429>

Schieferdecker I, Urgen Großmann J, Wendland. (2012) *Model-Based Testing: Trends*. <https://doi.org/10.1081/E-ESE-120046903>

Tauro C. (2013, 3 agosto). *A Comparative Analysis of Different No SQL Databases on Data Model*. Proceedings of International Conference on Emerging Research in Computing, Information, Communication and Applications, ERCICA 2013.

Zhao G, Huang W,Tang Y. (2013). *Modeling MongoDB with relational model*. <https://doi.org/10.1109/EIDWT.2013.25>

Parker Z, Poe S, Vrbsky S. (2013). *Comparing NoSQL MongoDB to an SQL DB*. <https://doi.org/10.1145/2498328.2500047>

Sangeeta G, Narsimha G. (2015). *Correlation and comparison of NoSQL specimen with relational data store*. IJRET: International Journal of Research in Engineering and Technology.

Nance C, Losser T, Harmon G. (2013, 18 mayo). *NoSQL Vs RDBMS. Why there is room for both*. Association for Information Systems AIS Electronic Library (AISeL).

Gilbert S, Lynch N. (2012, febrero). *Perspectives on the CAP Theorem*. Computer 45, no. 2: 30-36.

Pamina J, Shankar K, Beschi Raja J. (2018). *Correlating NoSQL Databases With a Relational Database: Performance and Space*. International Journal of Pure and Applied Mathematics, (2018), 235-244, 118(7)

Bassil Y. (2012). *A Comparative Study on the Performance of the Top DBMS Systems*. JCSCR Journal of Computer Science & Research Journal of Computer Science & Research, 20-31, 1(1)

Helland P, Haderle D. (2013). *Engagements: Building Eventually ACiD Business Transactions*. CIDR

Pavlo A, Aslett. (2016). *What's Really New with NewSQL?*. MACM SIGMOD Record, 45(2), 45-55

Yassien A, Desouky A. (2016). RDBMS, NoSQL, Hadoop: A Performance-Based Empirical Analysis. <http://dx.doi.org/10.1145/2944165.2944174>

Hasan R, Gandon F. (2014). A machine learning approach to SPARQL query performance prediction <http://dx.doi.org/10.1109/WI-IAT.2014.43>

# Anexos I – Codigo fuente e instalaciones

En este anexo se detallarán los procesos de instalación y donde localizar los diferentes artefactos construidos para el estudio. Los scripts de creación de base de datos y de la generación de registros en las mismas, así como el código fuente de la aplicación Spring utilizada para las consultas, puede encontrarse en el github del proyecto mediante licencia OpenSource Apache 2.0: [danielherranzsegundo/TFM\_2022: Codigo y scripts generados en el desarrollo del TFM de Daniel Herranz Segundo (github.com)](https://github.com/danielherranzsegundo/TFM_2022)

## **Instalación MySQL cluster InnoDBsandbox**

Para la instalación del cluster MySQL en la plataforma de test, se han seguido las recomendaciones de <https://github.com/wwwted/MySQL-InnoDB-Cluster-local-sandbox>

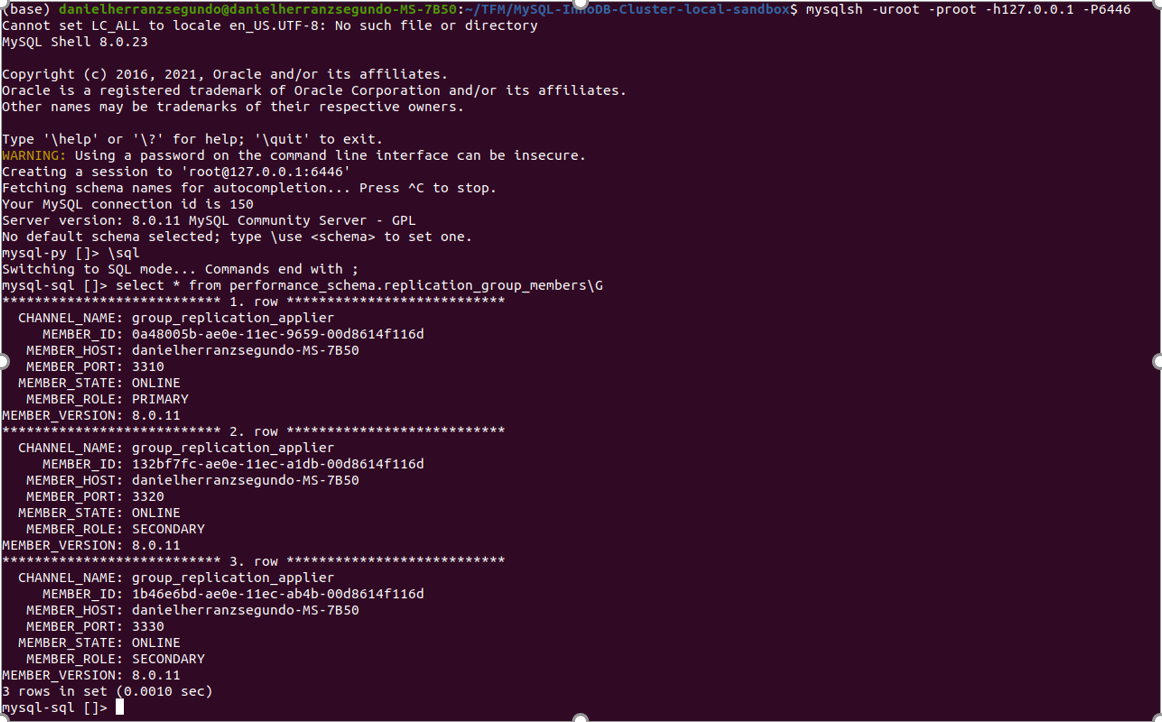
En cualquier caso, ante diversos problemas encontrados durante la instalación, se recomienda tambien instalar el core de MySQL server en su versión 8.0 con el gestor de paquetes apt

*sudo apt install mysql-server-core-8.0*

Siguiendo los pasos anteriormente descritos se debe haber desplegado correctamente el cluster (evidencias)



Y los nodos deben estar correctamente replicando información.



## **Instalación MySQL MongoDB cluster**

## **Instalación Cassandra docker cluster**

**Manual sysbench para MySQL**

<https://imysql.com/wp-content/uploads/2014/10/sysbench-manual.pdf>

tar xf sysbench-1.0.20.tar.gz cd sysbench-1.0.20

./autogen.sh

./configure make -j

**Configuración sysbench para definir DML**

<https://www.percona.com/blog/2019/04/25/creating-custom-sysbench-scripts/>

**Instalación del software sysbench en plataforma Linux**.

Mas información <http://www.howtoforge.com/how-to-benchmark-your-system-cpu-file-io-mysql-with-sysbench>

**Instalación**: sudo apt-get install sysbench

**Preparación del entorno**: sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test --mysql-user=root --mysql-password=yourrootsqlpassword prepare

**Evaluación y bechmark**: sysbench --test=oltp --oltp-table-size=1000000 --mysql-db=test --mysql-user=root --mysql-password=yourrootsqlpassword --max-time=60 --oltp-read-only=on --max-requests=0 --num-threads=8 run

# Anexos II – Diccionario de datos

### 

En este anexo se detallarán los diferentes tipos de datos utilizados durante el estudio.

### **OBReadAccount5**

| **Atributo** | **Cardinalidad** | **XPath** | **Tipo** |
| --- | --- | --- | --- |
| AccountId | 1..1 | OBReadAccount5/OBReadDataAccount5/Account/AccountId | int |
| Status | 0..1 | OBReadAccount5/OBReadDataAccount5/Account/Status | Texto (45) |
| StatusUpdateDateTime | 0..1 | OBReadAccount5/OBReadDataAccount5/Account/StatusUpdateDateTime | ISODateTime |
| AccountType | 1..1 | OBReadAccount5/OBReadDataAccount5/Account/AccountType | Texto (45) |
| AccountSubType | 1..1 | OBReadAccount5/OBReadDataAccount5/Account/AccountSubType | Texto (45) |
| Nickname | 0..1 | OBReadAccount5/OBReadDataAccount5/Account/Nickname | Texto (70) |
| OpeningDate | 0..1 | OBReadAccount5/OBReadDataAccount5/Account/OpeningDate | ISODateTime |
| AccountInfo | 0..n | OBReadAccount5/OBReadDataAccount5/Account/Account | OBCashAccount5 |
| SchemeName | 1..1 | OBReadAccount5/OBCashAccount5/Account/AccountInfo/SchemeName | Texto (70) |
| Identification | 1..1 | OBReadAccount5/OBCashAccount5/Account/AccountInfo /Identification | Texto (256) |
| Name | 0..1 | OBReadAccount5/OBCashAccount5/Account/ AccountInfo /Name | Texto (70) |

Tabla 6. Definición datos entidad OBReadAccount5 (fuente. Elaboración propia)

### **OBReadBalance1**

| **Name** | **Cardinalidad** | **XPath** | **Tipo** |
| --- | --- | --- | --- |
| AccountId | 1..1 | OBReadBalance1/AccountId | id |
| CreditDebitIndicator | 1..1 | OBReadBalance1/CreditDebitIndicator | Texto (45) |
| Type | 1..1 | OBReadBalance1/Type | Text(45) |
| DateTime | 1..1 | OBReadBalance1/DateTime | ISODateTime |
| Amount | 0..1 | OBReadBalance1/Amount | Amount |
| CreditLine | 0..1 | OBReadBalance1/CreditLine | CreditLine |

Tabla 7. Definición datos entidad OBReadBalance1 (fuente. Elaboración propia)

### **Currency**

| **Name** | **Cardinalidad** | **XPath** | **Tipo** |
| --- | --- | --- | --- |
| CurrencyId | 1..1 | Currency/CurrencyId | id |
| Code | 1..1 | Currency/Code | Texto (3) |
| Description | 0..1 | Currency/Description | Texto (70) |

### **Amount**

| **Name** | **Cardinalidad** | **XPath** | **Tipo** |
| --- | --- | --- | --- |
| CurrencyId | 1..1 | Amount/CurrencyId | Id |
| Amount | 1..1 | Amount/Amount | Double |
| AmountId | 1..1 | Amount/AmountId | id |

### **CreditLine**

| **Name** | **Cardinalidad** | **XPath** | **Tipo** |
| --- | --- | --- | --- |
| CreditLineId | 1..1 | CreditLine/CreditLineId | Id |
| CurrencyId | 1..1 | CreditLine /CurrencyId | Id |
| Amount | 1..1 | CreditLine /Amount | Double |
| Type | 1..1 | CreditLine/Type | Texto (45) |
| Included | 1..1 | CreditLine/Included | Boolean |

### **OBParty2**

| **Atributo** | **Cardinalidad** | **XPath** | **Tipo** |
| --- | --- | --- | --- |
| PartyId | 1..1 | OBParty2/PartyId | Text(45) |
| PartyNumber | 0..1 | OBParty2/PartyNumber | Text(45) |
| PartyType | 0..1 | OBParty2/PartyType | Text(4) |
| Name | 0..1 | OBParty2/Name | Text(70) |
| FullLegalName | 0..1 | OBParty2/FullLegalName | Text(256) |
| LegalStructure | 0..1 | OBParty2/LegalStructure | Text(256) |
| BeneficialOwnership | 0..1 | OBParty2/BeneficialOwnership | boolean |
| AccountRole | 0..1 | OBParty2/AccountRole | Text(45) |
| EmailAddress | 0..1 | OBParty2/EmailAddress | Text(256) |
| Phone | 0..1 | OBParty2/Phone | Text(12) |
| Mobile | 0..1 | OBParty2/Mobile | Text(12) |
| Address | 0..n | OBParty2/Address | Address |
| AddressType | 0..1 | OBParty2/Address/AddressType | Text(4) |
| AddressLine | 0..1 | OBParty2/Address/AddressLine | Text(70) |
| StreetName | 0..1 | OBParty2/Address/StreetName | Text(70) |
| BuildingNumber | 0..1 | OBParty2/Address/BuildingNumber | Text(16) |
| PostCode | 0..1 | OBParty2/Address/PostCode | Text(16) |
| TownName | 0..1 | OBParty2/Address/TownName | Text(45) |
| CountrySubDivision | 0..1 | OBParty2/Address/CountrySubDivision | Text(45) |
| Country | 1..1 | OBParty2/Address/Country | Text(4) |

Tabla 8. Definición datos entidad OBParty2 (fuente. Elaboración propia)

### **Country**

| **Name** | **Cardinalidad** | **XPath** | **Tipo** |
| --- | --- | --- | --- |
| CountryId | 1..1 | Country/CountryId | id |
| ShortName | 1..1 | Country/ShortName | Texto (35) |
| Description | 0..1 | Country/Description | Texto (256) |
| Code | 1..1 | Country/ | Texto (3) |