2. El papel de las base de datos en las aplicaciones software

Se pueden diferenciar dos usos muy comunes en las bases de datos, como elemento de integración o bien como una aplicación única.

En el primer caso, las bases de datos permiten integrar múltiples aplicaciones, generalmente realizadas por diferentes equipos, que almacenan sus datos en una base de datos común. Esto mejora la comunicación, puesto que todas las aplicaciones operan sobre un conjunto consistente de datos persistentes. También existen desventajas para esta aproximación, dado que una estructura que está diseñada para integrar muchas aplicaciones acaba siendo muy compleja. Por ejemplo, en este contexto, cuando una aplicación quiere hacer cambios sobre los datos almacenados, entonces necesita coordinarse con el resto de las aplicaciones que usan la base de datos. Además, las diferentes aplicaciones tienen diferentes estructuras y necesidades, así, por ejemplo, una aplicación puede requerir un índice, pero este índice puede que cause un problema para realizar inserciones de datos a otra aplicación. Por otra parte, el hecho de que cada aplicación normalmente es implementada por un equipo separado también tiene como consecuencia que la base de datos no pueda confiar a las aplicaciones la actualización de los datos de forma que se preserve la integridad de la base de datos, de forma que es la propia base de datos la responsable de asegurarlo.

Otro caso de uso es como una única aplicación, de manera que hay una única aplicación que accede a la base de datos. En este caso, solo hay un equipo de desarrollo que debe conocer la estructura de la base de datos, lo que hace más fácil mantener y realizar cambios sobre la misma. Así mismo, dado que solo

existe una aplicación que interactúe con la base de datos, la integración de ambas puede realizarse en el propio código de la aplicación. Así, mismo, cualquier tema de interoperabilidad puede ser gestionado desde las interfaces de la aplicación, lo que permite una mejor interacción con los protocolos y llevar a cabo cambios en los mismos. Por ejemplo, en los últimos tiempos se ha utilizado una arquitectura orientada a servicios que se basa en servicios web. En estos casos, la comunicación se realiza vía HTTP, lo que permite un mecanismo de comunicación ampliamente utilizado. Un aspecto interesante en el uso de los servicios web como mecanismo de integración resulta en ser más flexible para la estructura de datos que es intercambiada. Si se comunica usando SQL, los datos deben estar estructurados como relaciones. Sin embargo, con un servicio web, se pueden usar estructuras de datos más ricas con registros anidados y listas, que normalmente están representados como documentos XML o JSON. En general, en las comunicaciones remotas, interesa reducir el número de comunicaciones envueltas en cada interacción, de manera que es útil poder usar una estructura de información rica en cada petición o respuesta. Hay que observar que en este contexto existe libertad para elegir la base de datos, dado que existe un desacoplamiento entre la base de datos interna y los servicios que se usan para interaccionar con el mundo exterior (no es necesario que el resto de las aplicaciones conozcan cómo se almacenan los datos).

3. Las limitaciones de las bases de datos relacionales

Las bases de datos relacionales han sido durante décadas el modelo de persistencia más utilizado en la mayoría de las aplicaciones software. En este sentido se van a revisar las ventajas y desventajas que presentan. Desde el punto de vista de los beneficios que aportan las bases de datos relacionales, se puede destacar:

- *Persistencia de datos.* Una de las misiones más importantes de las bases de datos es mantener cantidades enormes de datos persistentes. En la mayoría de las aplicaciones es necesario disponer de un almacén de datos que actúe de backup. Estos almacenes se pueden estructurar de muchas formas, como, por ejemplo, un archivo del sistema de archivos del sistema operativo. Sin embargo, la forma preferida es una base de datos, pues ofrece más flexibilidad que el sistema de archivos cuando almacena grandes cantidades de datos a la hora de permitir obtener conjuntos de información de una forma rápida y fácil.
- *Concurrencia.* En general, en las aplicaciones normales suele haber muchas personas que están trabajando sobre los mismos dados, y puede que estén modificando los mismos datos. Esta situación obliga a coordinar las diferentes interacciones sobre los mismos datos para evitar inconsistencias en estos. La gestión directa de la concurrencia es complicada y existen muchas posibilidades de cometer errores. Las bases de datos ayudan a controlar todos los accesos a los datos usando transacciones. Aunque no es un sistema que no pueda dar lugar a errores (pueden ocurrir errores en las transacciones y enton-

ces hay que retroceder la transacción), sin embargo, sí permite gestionar la complejidad que supone la concurrencia.

- *Integración*. Con frecuencia, las aplicaciones requieren interaccionar con otras aplicaciones de forma que compartan los mismos datos, y unas y otras utilizan los datos que han sido modificados por el resto de las aplicaciones. Una forma de implementar esta interacción consiste en usar una integración basada en compartir una base de datos, es decir, las diferentes aplicaciones que interaccionan almacenan sus datos en una única base de datos. Usar una única base de datos permite que todas las aplicaciones usen los datos de las restantes de una forma fácil. Además, el sistema de control de la concurrencia de las bases de datos permite que existan múltiples aplicaciones usando la misma base de datos.
- *Modelo estándar*. Las bases de datos relacionales han tenido éxito debido a que proporcionan los beneficios clave de constituir un sistema estándar. Una persona puede aprender unos conocimientos generales sobre el modelo relacional que son comunes a las diferentes bases de datos relacionales y podrá aplicarlos en cualquier caso particular. Es decir, los mecanismos nucleares son los mismos y las diferencias serán, por ejemplo, la existencia de dialectos de SQL o pequeñas diferencias de funcionamiento de las transacciones, pero esencialmente el funcionamiento es el mismo.

Sin embargo, las bases de datos relacionales no son perfectas. Uno de los principales problemas hace referencia a la diferencia entre el modelo relacional y las estructuras de datos en memoria que algunos autores denominan **«impedancia».** El modelo relacional organiza los datos en una estructura de tablas y filas (también denominado «relaciones y tuplas»). En el modelo relacional, una

tupla es un conjunto de pares nombre-valor y una relación es un conjunto de tuplas. Todas las operaciones en SQL consumen y retornan relaciones de acuerdo con el algebra relacional. Este uso de las relaciones proporciona cierta elegancia y simplicidad, pero introduce un conjunto de limitaciones. En particular, los valores en una tupla relacional tienen que ser simples (no pueden tener estructura tal como un registro anidado o una lista). Esta limitación no se da en las estructuras de datos en memoria, donde existe un conjunto de tipos de estructuras mucho más ricas que las relaciones. Como consecuencia, si se quiere usar una estructura de datos más compleja que la relación, hay que traducirla a una representación relacional para poderla almacenar en disco.

Este problema llevó a pensar que las bases de datos relacionales serían sustituidas por bases de datos que replicaran las estructuras de datos en memoria. Y en este sentido la aparición de los lenguajes orientados a objetos, y junto con ellos de las bases de datos orientadas a objetos, hizo pensar que estas últimas acabarían ocupando el lugar de las bases de datos relacionales. Así, los lenguajes de programación orientados a objetos triunfaron, sin embargo, las bases de datos orientadas a objetos quedaron en el olvido. La razón del éxito de las bases de datos relacionales se ha debido a que constituyen un mecanismo de integración muy sólido basado en la existencia de un lenguaje estándar de manipulación de datos, el lenguaje SQL, y en la existencia de dos roles en los equipos de desarrollo: desarrolladores y los administrados de las bases de datos, que permiten llevar a cabo esta integración y coordinación. Hay que observar por otro lado que en los últimos tiempos ha aparecido un conjunto de frameworks que mapean información del mundo relacional al mundo de la orientación de objetos aliviando el problema de la impedancia. Sin embargo, el mapeo tiene también sus propios problemas cuando al hacer uso

del mismo se obvia la existencia de una base de datos y se llevan a cabo consultas que hacen descender de una forma importante el rendimiento de la base de datos.

Otra limitación presente en las bases de datos relacionales es su integración en un **clúster.** Un clúster es un conjunto de máquinas que permite implementar soluciones para escalar una aplicación (escalado horizontal). Es una solución más resistente que disponer de una única máquina más potente (escalado vertical), dado que, cuando se produce un fallo en un clúster, la aplicación podrá continuar funcionando, proporcionando, así, una alta fiabilidad. Desde el punto de vista de las bases de datos relacionales, existe un problema, dado que estas no están diseñadas para ejecutarse sobre un clúster. Existen algunas bases de datos relacionales que funcionan sobre el concepto de subsistema de disco compartido, usando un sistema de archivos compatible con un clúster que escribe sobre un subsistema de disco de alta disponibilidad, pero eso significa que el clúster tiene el subsistema de disco como un único punto de fallo. Por otra parte, las bases de datos relacionales podrían ejecutarse en servidores separados para diferentes conjuntos de datos, implementando lo que se denomina «una solución de **sharding** de la base de datos». Sin embargo, esta solución plantea el problema de que todo el sharding tiene que ser controlado por la aplicación, por lo que deberá mantener el control sobre qué servidor debe ser preguntado para cada tipo de datos. Además, se pierden determinadas características acerca de las consultas que se pueden realizar, integridad referencial, transacciones o control de la consistencia. Otro problema adicional es el coste: normalmente las bases de datos relacionales están pensadas para ejecutarse en un único servidor, de manera que si se tienen que ejecutarse en un clúster requieren de varias instancias de la base de datos, lo que aumanta, así, el coste económico.

4. Bases de datos NoSQL

Debido a las limitaciones mencionadas en la sección anterior, algunas empresas decidieron usar alternativas a las bases de datos relacionales. Este fue el caso de Google y Amazon, que desarrollaron sistemas de almacenamiento basado en el uso de clústeres: las Big Tables de Google y Dynamo de Amazon. Estos sistemas permitían gestionar grandes cantidades de datos en ambientes distribuidos y llevar a cabo su procesamiento, por lo que se consideran el origen de las denominadas **«bases de datos NoSQL».** Aunque muchas empresas no gestionaban las escalas de datos de las empresas antes mencionadas, sin embargo, muchas de ellas empezaron a diseñar sus aplicaciones para ejecutarse en estos ambientes distribuidos y usar este tipo de bases de datos al descubrir las ventajas que ofrecían:

- *Productividad del desarrollo de aplicaciones.* En los desarrollos de muchas aplicaciones se invierte un gran esfuerzo en realizar mapeos entre las estructuras de datos que se usan en memoria y el modelo relacional. Las bases de datos NoSQL proporcionan un modelo de datos que encaja mejor con las necesidades de las aplicaciones simplificando así la interacción, lo que resulta en tener que codificar, depurar y evolucionar menos las aplicaciones.
- *Datos a gran escala.* Las organizaciones han encontrado muy valiosa la posibilidad de tener muchos datos y procesarlos rápidamente. En general, es caro (en el caso de ser posible) hacerlo con una base de datos relacional. La primera razón es que las bases de datos relacionales están diseñadas para ejecutarse en una única máquina, y por otro lado es mucho más barato ejecutar muchos datos y procesarlos si se encuen-

tran cargados sobre clústeres de muchas maquinas pero más pequeñas y baratas. La mayoría de las bases de datos NoSQL están diseñadas para ejecutarse sobre clústeres, por lo que encajan mejor para estas situaciones.

Algunas de las características compartidas por las bases de datos NoSQL son:

- No usan SQL como lenguaje de consultas, sin embargo, algunas de ellas utilizan lenguajes de consultas similares a SQL, tales como CQL en Cassandra.
- En general se trata de proyectos de código abierto.
- Muchas de las bases de datos NoSQL nacieron por la necesidad de ejecutarse sobre clúster, lo que ha influido sobre su modelo de datos como su aproximación acerca de la consistencia. Las bases de datos relacionales usan transacciones ACID para manejar la consistencia de la base de datos, sin embargo, esto choca con un entorno de clúster, de manera que ofrecen diversas opciones para implementar la consistencia y la distribución. Sin embargo, no todas las bases de datos NoSQL están orientadas a correr sobre clúster.
- Las bases de datos NoSQL no tienen un esquema fijo.

De todas las características sobre las bases de datos NoSQL, destaca la no existencia de un esquema. Cuando se van almacenar datos en una base de datos relacional, lo primero que hay que hacer es definir un esquema que indique que tablas existen, qué columnas tiene cada tabla y qué tipo de datos tiene cada columna. Sin embargo, las bases de datos NoSQL operan sin esquema, lo que permite añadir libremente campos a los registros de la base de datos sin tener que redefinir la estructura. Esto es particular-

mente útil cuando se trata con campos de datos personalizados y no uniformes (datos donde cada registro tiene un conjunto de campos diferentes) que fuerzan a las base de datos relacionales a usar campos que son difíciles de procesar y entender, dado que un esquema relacional pondrá todas las filas de una tabla en un esquema rígido,[1] lo que se complica cuando se tienen diferentes tipos de datos en diferentes filas, de manera que al final se tienen muchas columnas que son nulas (una tabla dispersa) o bien columnas que no tienen significado. Cuando no se usa esquema, se evita esto, dado que se permite que cada registro pueda contener solo aquello que necesita. Los defensores de no usar esquemas apuntan a este grado de libertad y flexibilidad. Con un esquema es necesario saber antes lo que se necesita almacenar, lo cual en ocasiones puede no conocerse. Sin embargo, cuando no se tiene que cumplir un esquema, se puede almacenar fácilmente lo que se necesite, permitiendo cambiar[2] los datos almacenados según se va conociendo más acerca del proyecto. También, si hay información que no es necesaria seguir almacenando, se puede dejar de almacenar sin preocuparse por perder datos antiguos, como si se borrasen columnas en un esquema relacional. En este

1 Aunque se crítica a los esquemas relacionales por tener que definir a priori los esquemas y ser inflexibles, esto no es del todo cierto, puesto que los esquemas pueden ser cambiados en cualquier momento con determinados comandos de SQL. Así, por ejemplo, se pueden crear nuevas columnas para almacenar datos no uniformes. Sin embargo, en muchas ocasiones la no uniformidad en los datos es una buena razón para usar una base de datos sin esquema. En este sentido, la no existencia de esquema tiene un gran impacto sobre cambios en la estructura a lo largo del tiempo, especialmente en datos más uniformes.

2 Aunque no se haga habitualmente, los cambios en un esquema de bases de datos relacionales pueden ser hechos de forma controlada, y de forma similar se debe controlar cuándo se cambia la forma en que se almacenan los datos en una base de datos sin esquema de forma que sea fácil acceder tanto a los datos nuevos como a los antiguos.

sentido es muy interesante usar soluciones sin esquemas, pero también tienen desventajas. Muchas veces con los datos se hacen procesamientos en los que es necesario conocer el nombre de los campos, el tipo de los datos que tiene cada campo, etc., de manera que la mayoría de las veces los programas que acceden a los datos descansan sobre alguna forma implícita de esquema que asume que ciertos nombres de campos llevan determinados datos de un tipo de datos con un cierto significado. Así, aunque la base de datos no tenga esquemas, existe un esquema implícito, que es el conjunto de suposiciones acerca de la estructura de los datos en el código que manipula los mismos. Tener el esquema implícito en el código de la aplicación produce algunos problemas, como, por ejemplo, para saber qué datos están presentes, hay que consultar el código de la aplicación, de forma que si está bien estructurado será fácil de deducir el esquema, de lo contrario, puede convertirse en una tarea tediosa. Por otro lado, la base de datos permanece ignorante del esquema y no puede usarlo para decidir cómo almacenar y recuperar datos de manera eficiente, no puede aplicar sus propias validaciones sobre los datos de manera que se asegure que diferentes aplicaciones no manipulan datos de una forma inconsistente. Estas son algunas de las razones por las que las bases relacionales han fijado sus esquemas, y el valor que tienen los mismos. Esencialmente, las bases de datos sin esquemas cambian el esquema dentro del código de la aplicación que lo accede. Esto se convierte en problemático si existen múltiples aplicaciones desarrolladas por diferentes personas que acceden a la misma base de datos. Estos problemas pueden ser mitigados bien encapsulando toda la interacción de la base de datos en una única aplicación e integrarlo con otras aplicaciones usando servicios web o bien estableciendo claramente diferentes puntos de acceso a los datos almacenados por parte de las aplicaciones.

5. Modelos de bases de datos NoSQL orientados hacia agregados

Aunque resulta complicado realizar una categorización de las bases de datos NoSQL puesto que en muchos casos algunas bases de datos comparten características de varias familias, sin embargo, de acuerdo con el modelo de datos se pueden distinguir:[3]

- Bases de datos clave-valor: Riak, Redis, Dynamo, Voldemort.
- Bases de datos orientadas a documento: MongoDB, CouchDB.
- Bases de datos basadas en columnas: Cassandra, Hypertable, HBase, SimpleDB
- Bases de datos de grafos: Neo4J, Infinite Graph.

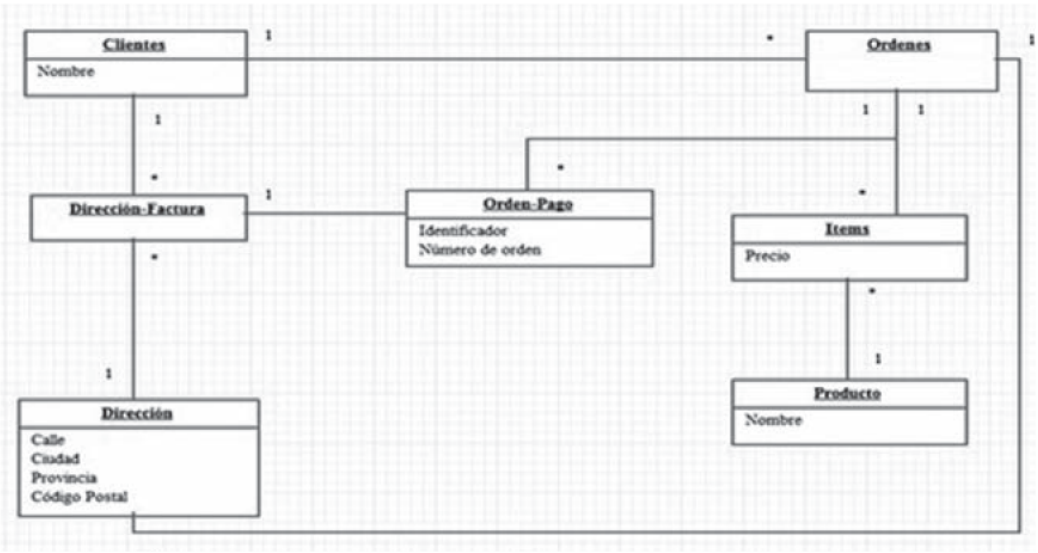
Las tres primeras familias de bases de datos NoSQL mencionadas comparten una característica común en sus modelos de datos y se trata de modelos orientados hacia agregados. En el modelo relacional se toma la información que se quiere almacenar y se divide en tuplas (filas). Una tupla es una estructura de datos limitada, dado que captura un conjunto de valores de manera que no se puede anidar una tupla dentro de otra para conseguir registros anidados, ni se puede poner una lista de valores o tuplas dentro de otra. Esta simplicidad permite pensar cuando se opera que se toman tuplas y se retornan tuplas. Sin embargo, en la orientación agregada hay un cambio de aproximación, y se parte del hecho de que a veces interesa operar con estructuras más complejas que un conjunto de tuplas tales

3 Hay que observar que esta clasificación no es exhaustiva, y se puede encontrar una clasificación más detallada en la siguiente dirección: http://nosql-database.org.

como un registro complejo que puede contener listas y otras estructuras de registro que están anidadas dentro del mismo. Estos registros complejos se van a denominar **«agregados»,** y van a representar un conjunto de datos relacionados que son tratados como una unidad. En particular son una unidad para la manipulación de datos y para la gestión de la consistencia, es decir, se actualizarán agregados con operaciones atómicas y se interaccionarán con la base de datos en términos de agregados. También el uso de agregados facilita que las bases de datos puedan operar sobre clústeres, pues el concepto de agregado se convierte en una unidad natural para la replicación y la distribución. Asimismo desde el punto de vista del programador, es más fácil operar con agregados, pues habitualmente operan con datos a través de estructuras agregadas.

Para fijar las ideas, se va a comparar el modelo relacional con la agregación de datos en un escenario en el que se quiere gestionar información de una tienda virtual en la que se venden productos de algún tipo y hay que almacenar información acerca de los usuarios, el catálogo de productos, órdenes de compra, direcciones, etc. Esta información podría modelarse tanto usando el modelo relacional como usando agregación. Así, si se usa un modelo relacional, se podría modelar usando el diagrama UML de la figura 1. En este modelo supuestamente normalizado, no hay repeticiones de datos, hay integridad referencial, etc., y se requieren siete tablas.

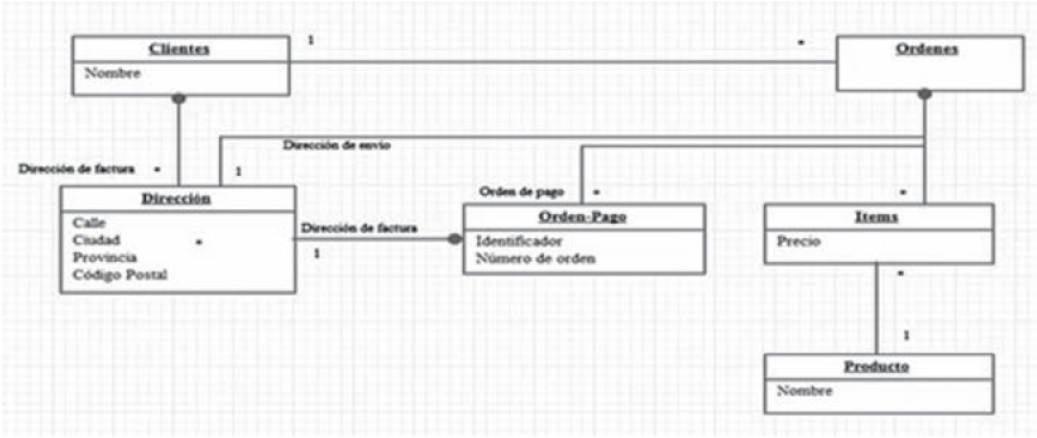
Figura 1. Solución relacional al supuesto planteado.



Sin embargo, si se usa un modelo agregado, se podría modelar de acuerdo con la figura 2. En este modelo se tienen dos agregados: «clientes» y «órdenes». El cliente contiene una lista de direcciones de facturación y las órdenes contienen una lista de productos vendidos, direcciones de envío y órdenes de pago. Las órdenes de pago en sí mismas contienen una dirección de facturación para ese pago. Asimismo el registro de la dirección aparece tres veces, pero, en vez de usar ID, se trata como un valor y se copia cada vez (esto es coherente con el dominio que se modeliza en el que no se quiere que se cambie ni la dirección de envío ni la dirección de pago). En el modelo relacional, se aseguraría que las filas de la dirección no son actualizadas para ese caso haciendo una nueva fila en su lugar. Con los agregados se puede copiar la estructura entera de la dirección en el agregado tal como se necesita. Por otro lado, el enlace entre el cliente y la orden es una relación entre agregados. De la misma forma, se podría haber definido un enlace entre los ítems y los productos, siendo los productos un agregado separado, aunque en este caso no se ha hecho y el nombre del producto se muestra como parte

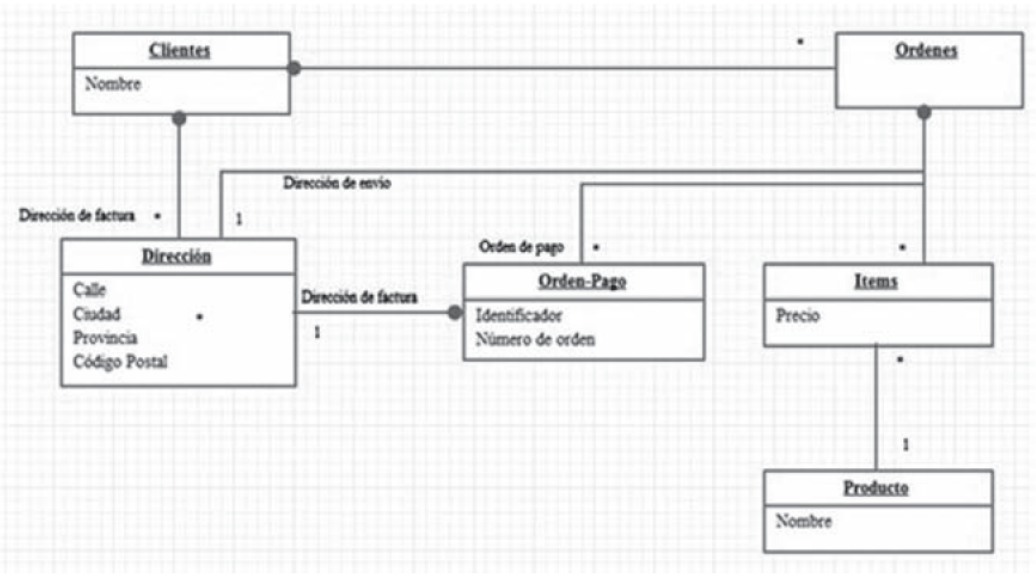
del ítem con el objetivo de minimizar el número de agregados a los que hay que acceder.

Figura 2. Solución agregada al supuesto planteado.



Hay que observar que en el caso agregado, cuando se modeliza hay que pensar en cómo se va acceder a los datos, lo cual afectará al desarrollo de la aplicación que use ese modelo de datos. En este sentido igual de válido habría sido pensar un modelo de datos en el que todas las ordenes son parte de un agregado denominado «clientes» (figura 3). En este sentido no existe un modelo mejor que otro, todo depende de la forma en la que se accede a los datos. Si, por ejemplo, se prefiere que al acceder a un cliente se acceda a todas las órdenes asociadas al cliente, se optaría por este último modelo. Sin embargo, si se prefiere gestionar una orden cada vez, es preferible el primer modelo, y tener varios agregados.

Figura 3. Otra solución agregada al supuesto planteado.



Según ejemplo descrito se puede observar que el modelo relacional captura los diferentes elementos de datos y las relaciones que existen entre ellos sin usar ninguna noción de agregación entre ellos (en el ejemplo se podría deducir que una orden está constituida por un conjunto de ítems, una dirección de envío y una orden de pago), sino que lo expresa en términos de claves ajenas en las relaciones (pero no existe nada para distinguir relaciones que representen agregaciones de aquellas que no lo son, perdiendo el conocimiento acerca de la estructura de agregación para almacenar y distribuir los datos). Por esta razón se dice que las bases de datos relacionales ignoran las agregaciones. Un modelo que ignora las agregaciones permite fácilmente ver los datos desde distintas perspectivas, de manera que será una buena elección cuando no existe una estructura clara de manipulación de los datos.

Cuando se trabaja con bases de datos orientadas a la agregación, se tiene una semántica más clara respecto a la agregación al tratarse de la unidad de interacción con el sistema de almacenamiento. No es una propiedad lógica de los datos, sino que tiene

que ver con cómo los datos están siendo usados por las aplicaciones. A veces es complicado definir agregados si los mismos datos van a ser usados en diferentes contextos, pues una estructura de agregación puede ser útil con algunas interacciones de datos pero ser un obstáculo para otros casos. Sin embargo, la razón básica para usar una orientación agregada es cuando se quiere utilizar un clúster. En este caso, hay que minimizar el número de nodos[4] que se necesitan consultar cuando se están recopilando datos. Así, cuando se definen agregaciones, se ofrece información acerca de qué conjuntos de datos deben ser manipulados juntos, y, por tanto, deberían encontrarse en el mismo nodo.

Otro aspecto clave sobre los modelos orientados a agregaciones se refiere al tema de las transacciones. Las bases de datos relacionales permiten manipular cualquier combinación de filas de cualquier tabla en una única transacción ACID (atómica, consistente, aislada y durable).[5] En general, es cierto que las bases de datos orientadas a agregados no soportan las transacciones ACID para múltiples agregados, sin embargo, soportan manipulaciones atómicas de un único agregado cada vez. Eso significa que, si se necesita manipular múltiples agregados de una manera atómica, habrá que hacerlo desde el código de la aplicación. En la práctica, se puede ver que muchas veces se puede mantener la atomicidad dentro de una única agregación, de hecho, esta situación es parte de la consideración que se debe hacer a la hora de dividir los datos en agregados.

4 En un clúster un nodo es cualquier instancia de una base de datos que se está ejecutando de forma distribuida con otras instancias.

5 La atomicidad hace referencia a que muchas filas que abarcan muchas tablas son actualizadas en una sola operación que tiene éxito o bien falla de manera completa. Además, las operaciones concurrentes son aisladas una de cada otra de manera que no puede realizarse una actualización parcial.

6. El modelos de distribución de datos de las bases de datos NoSQL

Cuando la cantidad de datos que manejar es enorme, se hace complicado y costoso escalar las bases de datos, y una posible solución consiste en la compra de un gran servidor que ejecute la base de datos (escalado vertical). Sin embargo, una opción mejor consiste ejecutar la base de datos en un clúster de servidores. Por esta razón, las bases de datos NoSQL son interesantes por su capacidad[6] para ejecutarse sobre clústeres de grandes dimensiones.

Dependiendo del modelo de distribución que se elija, se pueden obtener ventajas del tipo: capacidad de manejar grandes cantidades de datos, capacidad de procesar un mayor tráfico de lectura/escritura, mayor disponibilidad cuando se producen ralentizaciones en la red o un fallo de la red, etc. Sin embargo, la distribución conlleva un coste en términos de complejidad.

Esencialmente, hay dos modelos de distribución de datos: replicación y sharding. La replicación toma los mismos datos y los copia en múltiples servidores, mientras que el sharding distribuye diferentes datos a través de múltiples servidores de manera que cada servidor actúa como una única fuente para un subconjunto de datos. En este sentido se trata de dos modelos ortogonales, de manera que puede usarse en solitario uno de ellos o bien ambos.

La replicación puede ser de dos formas: maestro-esclavo o peer-to-peer. La replicación maestro-esclavo hace de un nodo la copia autorizada que soporta las escrituras mientras que

6 Capacidad basada en el concepto usado para organizar la información en forma de agregados de datos.

los nodos secundarios se sincronizan con el nodo primario y soportan las lecturas. En el caso de la replicación peer-to-peer, se permite la escritura sobre cualquier nodo, de manera que los nodos se coordinan para sincronizar las copias de los datos. Por una parte, la replicación maestro-esclavo reduce las posibilidades de conflictos de actualización, y la replicación peer-to-peer evita cargar las escrituras sobre un único punto de fallo.

7. La consistencia de los datos en las bases de datos NoSQL

El modelo de ejecución distribuido propio de las bases de datos NoSQL plantea varios tipos de problemas de consistencia en los datos:

- *Consistencia en las escrituras.* Cuando dos usuarios tratan de actualizar el mismo dato al mismo tiempo se produce un conflicto de escritura-escritura. Cuando las escrituras llegan al servidor, las serializa y entonces decide aplicar una y a continuación la otra. De esta forma, uno de los usuarios sufre una pérdida de actualización, puesto que la última escritura que se aplica sobrescribe la primera escritura. Existen varias aproximaciones para mantener la consistencia. Una aproximación pesimista[7] consiste en tener bloqueos de escritura, de manera que, si se quiere cambiar un valor, es necesario

7 Las aproximaciones pesimistas a menudo degradan severamente la capacidad de respuesta de un sistema hasta el grado de no ser útiles para su propósito. Este problema empeora por el peligro de errores, debidos a interbloqueos que son difíciles de prevenir y depurar.

adquirir un bloqueo. El sistema asegura que solo un cliente puede conseguir un bloqueo a la vez. Otra aproximación más positiva consiste en la actualización condicional donde algún cliente que hace una actualización testea el valor justo antes de actualizarlo para ver si ha sido cambiado desde su última lectura. Ambas aproximaciones descansan sobre una serialización consistente de las actualizaciones. Si existe un único servidor, tiene que elegir una y a continuación la otra. Sin embargo, si existe más de un servidor, varios nodos podrían aplicar las actualizaciones en orden diferente, por lo que resultará un valor diferente actualizado. Por esta razón se habla de consistencia secuencial, que consiste en asegurar que todos los nodos apliquen las operaciones en el mismo orden. Existen otras aproximaciones,[8] pero en cualquiera de ellas el objetivo es mantener el equilibrio entre evitar errores tales como actualizaciones conflictivas y la rapidez de respuesta al usuario. Por otra parte, la replicación hace mucho más probable encontrarse con conflictos de escritura-escritura. Si diferentes nodos tienen diferentes copias de algunos datos que pueden ser actualizados de manera independiente, se llegará a conflictos a menos que se tomen medidas específicas para evitarlos. Usando un único nodo para todas las escrituras de algunos datos hace que sea más fácil mantener la consistencia de las actualizaciones.

- *Consistencia en las lecturas*. Una inconsistencia de lectura o conflicto de lectura-escritura se produce cuando un usuario reali-

8 Existe otro camino optimista para tratar con los conflictos escritura-escritura, guardar todas las actualizaciones y registrar que están en conflicto. El siguiente paso consiste en mezclar las dos actualizaciones. El problema es que cualquier mezcla automatizada de conflictos escritura-escritura es específica del dominio y necesita ser programada para cada caso particular.

za una lectura en la mitad de la escritura de otro sobre los mismos datos que se están leyendo. Para evitar una inconsistencia lógica[9] debido a un conflicto lectura-escritura, en las bases de datos relacionales se usa el concepto de transacción, que asegura que en estas situaciones el usuario que lee o bien lee los datos antes de la escritura o bien los datos después de la escritura. En general se afirma que las bases de datos NoSQL no soportan transacciones, sin embargo, no es cierto que todas las bases de datos NoSQL no soporten transacciones ACID (por ejemplo, las orientadas hacia grafos suelen soportarlas) y además las bases de datos orientadas hacia agregados soportan actualizaciones atómicas sobre un único agregado (por lo que consiguen consistencia lógica dentro de un agregado pero no entre agregados). En la replicación aparece un nuevo tipo de consistencia denominada «consistencia de replicación», que consiste en asegurar que los mismos datos tienen el mismo valor cuando son leídos desde diferentes réplicas. En este sentido se dice que los nodos son «eventualmente consistentes», pues cuando se produce una actualización en un nodo puede que exista algún período en el que algunos nodos tengan inconsistencias, pero, si no se realizan más actualizaciones con el tiempo, al final, todos tomarán el mismo valor. Aunque la consistencia de replicación es independiente de la consistencia lógica, la replicación puede dar lugar a una inconsistencia lógica alargando su ventana de inconsistencia.[10]

9 Se denomina «consistencia lógica» aquella que asegura que los diferentes datos tienen sentido juntos.

10 La longitud de tiempo en la que una inconsistencia está presente se denomina «ventana de inconsistencia». La presencia de una ventana de inconsistencia significa que diferentes usuarios verán diferentes cosas al mismo tiempo. En los sistemas NoSQL, la ventana de inconsistencia generalmente es bastante pequeña.