Dominio de las BD relacionales debido a

- Filtrado eficiente de los datos persistentes.
- Control de acceso concurrente. Garantiza consistencia con transacciones ACID.
- Buena integración con aplicaciones.
- Su estadariación: basadas en el modulo relacional y usan SQL como lenguaje. Esto permite no tener que recodificar aplicaciones al cambiar la BD.

Modelando para el acceso a datos

Debemos tener en cuenta cómo va a ser el acceso a datos. Como alternativa podemos dividir el agregado y usar referencias (en BD documentales no es necesario almacenar las referencias en ambas direcciones. Debemos calcular el agregado para hacer consultas analícas (excepto en documentales, pero no es eficiente).

Problemas con las BD relacionales

Impedancia entre modelos de memoria y disco

Los datos en disco se almacenan como tuplas simples (1NF), mientras que en memoria se manejan estructuras complejas (anidadas). Como soluciones se plantean:

- BD orientadas a objetos, pero no llegaron a triunfar.
- El mapeado objeto-relacional no resuleve el problema del todo, pero induce nuevos problemas, como la ineficiencia al evitar el uso del SGBD.

Integración de aplicaciones a través de una BD centralizada

Este tipo de BD es compleja y requiere de mecanismos de verificación de consistencia. Además, los cambios en la BD requieren coordinación entre las aplicaciones. Como alternativa, se plantea que cada aplicación tenga su propia BD (incluso así, el modelo relacional domina). Esto permite:

- El equipo de cada app conoce, mantiene y evoluciona su BD.
- La verificación de integridad se realiza en cada aplicación.
- Integración a través de interfaces como servicios web, con uso de XML o JSON.
- Permite usar distintos modelos de BD, dependiendo de las necesidades de cada app.

Necesidad el uso de cluster

Surge del incremento en la escala de las aplicaciones modernas. Como soluciones se plantean:

- Escalado vertical con máquina más potentes (limitado y costoso).
- Escalado horizontal con *cluster* de máquinas más pequeñas. Esto da mayor resiliencia y abarata costes.

Surgen problemas ya que las BD relacionales no están diseñadas para arquitecturas distribuidas. Implementar entonces una BD sobre sistemas de archivos induce un único punto de fallo, y el almacenamiento fragmentado requiere que la aplicación controle la distribución de datos. Aquí surgen alternativas a tener en cuenta como Big Table de Google o Dynamo de Amazon.

NoSQL

No hay una definición clara del término. Inicialmente era “BD no relaciones, distribuidas y de código abierto”.

Aparición

- No usan SQL (aunque algunos lenguajes sean muy parecidos).
- Funcionan sobre *clusters*: importan sobre el modelo de datos y sobre la solución al problema de consistencia entre ACID y *cluster* (excepto las de grafos).
- Se adaptan a las nuevas necesidades y operan sin esquema, aunque se suelen usar como BD de aplicaciones, no de integración.
- Abre opciones para el almacenamiento de datos, sin restringir la incorporación de nuevos datos.
- Hay que considerar NoSQL cuando el tamaño y rendimiento hagan necesarios el uso de un *cluster* y por temas de productividad, ya que dan una interacción más natural con los datos.

Agregados

La gran diferencia de NoSQL con las BD relacionales es el uso de agregados, ya que el relacional no permite anidar estructuras. Se diseñan las estructuras pensando en cómo va a accederse a ellas: se agrupan y replican los datos en un único documento, minimizando los accesos (desnormalización). Como consecuencia, la agregación puede beneficiar unas consultas y perjudicar otras. Además, al trabajar contra un *cluster*, es necesario determinar qué datos deben ser físicamente próximos, para minimizar el acceso a varios nodos.

Las BD NoSQL, en general, no soportan ACID en trasacciones que se expander por varios agregados. Hay 4 grandes tipos de modelos.

Características comunes BD NoSQL

No tener esquema; no es necesario definir un esquema para almacenar datos. Esquema en la aplicación.

- **Clave-valor:** Cualquier valor se puede insertar para una clave.
- **Documentales:** No hay restricciones sobre el contenido de cada documento.
- **Column-family:** Cualquier dato se puede almacenar sobre una columna de una fila.
- **Grafos:** las propiedades de nodos y arcos son libres.

Modelos Key-Value

- Estos modelos funcionan mediante pares (Clave, Agregado), donde cada clave única tiene asociado un agregado (que es opaco para la BD, y no tiene restricciones en el contenido).
- Solo se puede consultar por clave, y no tiene esquema ni estructura.

Modelos Documentales

- Las BD entienden la estructura del agregado. Se permite ver y consultar dentro del agregado.
- No tienen esquema y los datos se almacenan en forma de documentos, generalmente JSON o XML.

Modelos Column-Family

- No confundir con el almacenamiento columnar !
- Eficientes en lectura.
- Evoluciona del modelo clave-valor y añade estructura (muy poca). El esquema es el nombre de las familias, lo único que declaramos. La ventaja de estos modelos son las familias.
- Tiene una estructura tabular (no es una tabla) con columnas dispersas.
- Es un clave-*column family*-valor: para cada clave de una fila (cada fila es un agregado) tenemos *column-families* (cada *column-family* define un tipo de registro, y es un bloque dentro del agregado). Para cada familiar tenemos pares clave-valor.

Cassandra

- Aquí una fila solo puede pertenecer a una *column-family*, y una *column-family* puede tener columnas anidadas (supercolumnas, equivalentes a las *column-family* de HBase o Big Table).
- La estructura y almacenamiento de las filas de una tabla puede ser:
  - *Skinny row:* pocas columnas. Mismas columnas en casi todas las filas.
  - *Wide row:* muchas columnas (miles). Filas con columnas muy variadas, qe modelan una lista.

Modelos de grafos

- Motivado por registros simples pero con muchas relaciones entre ellos.
- Modelo compuesto por nodos y arcos, pudiendo tener datos en ambos.
- Las consultas suelen centrarse en acceder a un nodo y mover por los arcos desde el mismo.
- Son rápidas en navegación, pero lentas en inserción.
- Ejemplo Neo4J, que tiene objetos java como propiedades de los nodos y arcos (no tiene esquema). Suele usarse para redes sociales, recomendaciones, etc.

Relaciones

- La BD no conoce la existencia de la relaciones entre datos, se almacenan como una referencia a una clave.
- Las BD relacionales proporcionan mecanismos explícitos para gestionar las relaciones mediantes transacciones ACID o FK. Son poco eficientes si hay que seguir muchas relaciones, ya que se deben hacer muchas *join*.
- Las NoSQL no suelen soportar la modificación de varios agregados en una misma transacción.

Trabajar sin esquema

Ventajas

- No hay que hacer asunciones a priori.
- Facilidad para incorporar cambios en los datos y para trabajar con datos no uniformes.
- Tener esquema resulta inflexible en la inserción de datos, es común un campo de texto done va cualquier cosa.

Desventajas

- Las aplicaciones suelen necesitar cierto formato y semántica en los datos, cierto esquema implicito en los datos. Por esto, hay que minimizar el uso de literales, para que el código se pueda cambiar y mejorar de forma fácil.
- Tener el esquema codificado en las aplicaciones puede dar problemas, ya que la BD no puede utilizarlo para mejorar la eficiencia.
- Los esquemas tienen valor. Una BD sin esquema solo traslada el problema del esquema a la aplicación (es como usar un lenguaje de tipado débil).

Soluciones

- Integrar todo el acceso a la BD en una única app que proporciona servicios web para las demás.
- Delimitar diferentes partes de cada agregado para el acceso de apps diferentes.

Los esquemas en BD relacionales son "flexibles", permiten añadir y quitar columnas por ejemplo. El problema de almacenar los datos de formas nuevas en BD NoSQL es que las apps deben funcionar con datos nuevos y antiguos. Esto se puede arreglar añadiendo una columna y conviviendo con dos columnas, una con nulos hacia arriba y otra con nulos hacia abajo.

Cambios en el esquema

Se le da importancia a los métodos ágiles para cambiar fácilmente de esquema. En NoSQL se pueden hacer cambios rápidos, pero hay que tener cuidado con las migraciones de esquema.

BD relacional

- Un cambio de esquema puede ser un proyecto en si mismo, ya que necesita *scripts* para la migración de datos.
- Con los proyectos nuevos simplemente almacenamos los cambios con los *scripts* de migración de datos.
- Con proyectos *legacy* extraemos el esquema de la BD y procedemos con los proyectos nuevos. Es importante mantener la compatibilidad hacia atrás y tener en cuenta que hay una fase de transición, donde ambos esquemas conviven.

BD NoSQL

- Se intenta no tener que realizar cambios de esquema.
- El esquema está en la aplicación ! Debe *parsear* los datos obtenidos de la BD, por lo que hay que cambiar el código que la app lee y escribe. Si no cambiamos la aplicación, el error de esquema que daría la BD lo dará la aplicación.
- Se hace una migración incremental, ya que mover todos los datos puede ser muy costoso. Transicionamos a partir e la propia aplicación, lee archivos de ambos esquemas (viejo y nuevo) pero escribe en el nuevo. De esta forma, hay datos que no llegan nunca a migrarse.
- En el caso de una BD de grafos, se pueden definir nuevos arcos con el nuevo esquema.