Tarea 1

Gonzalez Jimenez Victor Yotecatl

313173743

Bases de datos Gpo. 1

Profesor: Arreola Franco Fernando

• *Modelo orientado a objetos*

Por **definición una base de datos orientada a objetos** es una base de datos (BDOO) en la que la información está representada mediante objetos, como los presentes en la programación orientada a objetos. Al integrarse las características de una base de datos con las de un lenguaje de programación orientado a objetos (POO), se obtiene como resultado un sistema gestor de base de datos orientada a objetos (ODBMS), que hace que los objetos de la base de datos aparezcan como objetos de un lenguaje de programación (pudiendo dar soporte a uno o más de este tipo de lenguajes, como por ejemplo, Visual Basic, C++ o Java).

Aunque fue en los noventa cuando vivieron su primer apogeo, las BDOO nacieron en los años 60, de mano del doctor Nygaard, un especialista en la elaboración de sistemas informáticos noruego. Su idea base era crear un software diseñado en paralelo al objeto físico, de manera que si el objeto físico tenía 50 componentes, el software tendría a su vez 50 módulos. Para poder operar con este sistema, Nygaard creó también un lenguaje de apoyo, el Simula-67.

Se siguió trabajando en el desarrollo de este **tipo de base datos orientada a objetos** y unos años después, Alan Kay y Xeros tomaron de referencia el trabajo de Sumila-67 para crear otro lenguaje parecido, Smalltalk, que en los 80 daría paso a C++. Actualmente, el uso de BDOO a vuelta a cobrar importancia por la necesidad de satisfacer las necesidades de nuevas

aplicaciones que emplean lenguajes de programación orientados a objetos y a la actividad de las comunidades de software libre relacionadas con ellas y los POO.

Características

Las principales **características de la base de datos orientadas a objetos** se pueden dividir en tres grupos:

- Mandatorias, son aquellas características que deben estar en la BDOO de forma obligatoria, es decir, los requisitos imprescindibles que el sistema debe tener:
    - Debe soportar objetos complejos.
    - Los objetos deben tener un identificador al margen de los valores de sus atributos.
    - Encapsulación, es decir, los datos e implementación de los métodos están ocultos en los objetos.
    - El esquema de la BDOO tiene un conjunto de clases.
    - Concurrencia.
    - Recuperación.
    - Completación computacional.
    - Persistencia y manejador de almacenamiento secundario.
    - Facilidad de query.
- Opcionales cuando no es necesario incluirlas, pero si la BDOO cuenta con ellas, hará que el sistema sea mejor, entre otras:
    - Herencia múltiple.
    - Diseño de transacciones y versiones.
    - Comprobación de clases e inferencia de la distribución.

- Abiertas, son las características que el diseñador puede poner y que están relacionadas con la programación. Existen diferentes opciones, como por ejemplo, la representación del sistema.

Aparte de estas características, las BDOO cuentan con conceptos propios y clave del modelo de objetos, entre los que encuentran estas propiedades:

- La encapsulación oculta información al resto de objetos, de manera que pueden impedir los conflictos o los accesos incorrectos. Consiste en unir en la clase las variables (características) y los métodos (comportamientos), de manera que solo se tiene una unidad, de la que se conoce su comportamiento, pero no los detalles internos.
- La **herencia en base de datos orientada a objetos** hace referencia a que los objetos heredan comportamientos dentro de una jerarquía de clases, es decir, una clase se deriva de otra de manera que extiende su funcionalidad. La clase de la que se hereda puede llamarse clase base, clase padre, superclase, clase ancestro, etc. (dependiendo del lenguaje de programación que se esté usando)
- El polimorfismo es la propiedad que permite que una operación pueda aplicarse a objetos de distinta tipología.

Ventajas y desventajas

Como en otros tipos de bases de datos, el **modelo de base de datos orientado a objetos tiene ventajas y desventajas**, que debemos sopesar a la hora de implementarlo, teniendo en cuenta las necesidades que nuestra empresa, negocio o proyecto pueda tener respecto al empleo de una base de datos u otra.

Algunos ejemplos conocidos de estas bases de datos incluyen:

1. **db4o (Database for Objects)**: Una base de datos orientada a objetos pura, muy utilizada en entornos embebidos y de Java o .NET. Permite almacenar objetos directamente, sin necesidad de un mapeo relacional.

2. **ObjectDB**: Orientada a Java, esta base de datos soporta las especificaciones de JPA (Java Persistence API). Es útil en aplicaciones que necesitan una base de datos embebida con un enfoque orientado a objetos.
3. **GemStone/S**: Una base de datos orientada a objetos que soporta Smalltalk y Java. Es conocida por sus capacidades de escalabilidad y procesamiento de transacciones distribuidas.
4. **Versant Object Database**: Una solución empresarial que permite trabajar directamente con objetos en lenguajes como Java y C++. Está orientada a aplicaciones críticas que requieren alta disponibilidad.
5. **InterSystems Caché**: Aunque también puede ser utilizada como una base de datos relacional, permite el almacenamiento y manipulación de objetos. Se utiliza en industrias como la sanitaria, financiera y logística.
6. **ZODB (Zope Object Database)**: Una base de datos para Python que permite almacenar objetos directamente sin un mapeo a tablas relacionales. Es utilizada principalmente con el framework Zope, aunque puede ser usada en cualquier aplicación Python.

• *Modelos NoSQL (clave-valor, documentales, grafos) descripción, ventajas, desventajas, casos de uso…*

Modelado de datos en NoSQL

Una de las mayores diferencias entre las bases de relacionales y no relacionales radica en el enfoque que adoptamos para el **modelado de datos**. Las BBDD NoSQL no siguen un esquema rígido y predefinido. Esto permite a los desarrolladores elegir libremente el modelo de datos en función de las características del proyecto.
El objetivo fundamental es mejorar el rendimiento de las consultas, eliminando la necesidad de estructurar la información en tablas complejas. Así, NoSQL admite una gran variedad de datos desnormalizados como documentos JSON,

valores clave, columnas y relaciones de grafos.
Cada **tipo de base de datos NoSQL** está optimizado para facilitar el acceso, consulta y modificación de una clase específica de datos. Las principales son:

- **Clave-valor:** Redis, Riak o DyamoDB. Son las BBDD NoSQL más sencillas. Almacenan la información como si fuera un diccionario basado en pares de clave-valor, donde cada valor está asociado con una clave única. Se diseñaron con la finalidad de escalar rápidamente garantizando el rendimiento del sistema y la disponibilidad de los datos.
- **Documentales:** MongoDB, Couchbase. Los datos se almacenan en documentos como JSON, BSON o XML. Algunos las consideran un escalón superior de los sistemas clave-valor ya que permiten encapsular los pares de clave-valor en estructuras más complejas para realizar consultas avanzadas.
- **Orientadas a columnas:** BigTable, Cassandra, HBase. En lugar de almacenar los datos en filas como lo hacen las bases de datos relacionales, lo hacen en columnas. Estas a su vez se organizan en familias de columnas ordenadas de forma lógica en la base de datos. El sistema está optimizado para trabajar con grandes conjuntos de datos y cargas de trabajo distribuidas.
- **Orientadas a grafos:** Neo4J, InfiniteGraph. Guardan los datos como entidades y relaciones entre entidades. Las entidades se llaman "nodos" y las relaciones que unen los nodos son los "bordes". Son ideales para gestionar datos con relaciones complejas, como redes sociales o aplicaciones con ubicación geoespacial.

MongoDB

Es un SGBD de tipo documental desarrollado por 10gen en 2007. Es de código abierto y ha sido creado en lenguajes de programación como C++ C y JavaScript.

MongoDB es uno de los sistemas más populares para bases de datos distribuidas. Redes sociales como LinkedIn, empresas de telecomunicaciones como Telefónica o medios informativos como Washington Post utilizan MongoDB.

Veamos algunas de sus características principales.

- Almacenamiento en BBDD con MongoDB: MongoDB almacena los datos en documentos BSON (JSON binario). Cada base de datos se compone de una colección de documentos. Una vez que MongoDB está instalado y la Shell está en ejecución, podemos crear la BBDD simplemente indicando el nombre que queremos usar. Si la BBDD aún no existe, MongoDB la creará automáticamente al añadir la primera colección. De manera similar, una colección se crea automáticamente al almacenar un documento en ella. Sólo tenemos que agregar el primer documento y ejecutar la sentencia "insert" y MongoDB creará un campo ID asignándole un valor del tipo ObjectID que es único para cada máquina en el momento en el que se ejecuta la operación.
- Particionado en BBDD con MongoDB: MongoDB facilita la distribución de datos en múltiples servidores utilizando la función de sharding automático. La fragmentación de los datos se produce a nivel de colección, distribuyendo los documentos entre los distintos nodos del clúster. Para efectuar esta distribución se emplea una "clave de partición" definida como campo en todos los documentos de la colección. Los datos se fragmentan en "chunks" que tienen por defecto un tamaño de 64 MB y se almacenan en diferentes shards dentro del clúster, procurando que exista un equilibrio. MongoBD monitoriza continuamente la distribución de los chunks entre los nodos del shard y si fuera

necesario, efectúa un rebalanceo automático para asegurarse de que la carga de trabajo que soportan los nodos esté equilibrada.

- Replicado en BBDD con MongoDB: MongoDB utiliza un sistema de replicación basado en la arquitectura maestro-esclavo. El servidor maestro puede realizar operaciones de escritura y lectura, pero los nodos esclavos únicamente realizan lecturas (replica set). Las actualizaciones se comunican a los nodos esclavos mediante un log de operación llamado oplog.
- Consultas en BBDD con MongoDB: MongoDB cuenta con una potente API que permite acceder y analizar los datos en tiempo real, así como realizar consultas ad-hoc, es decir, consultas directas sobre una base de datos que no están predefinidas. Esto proporciona a los usuarios la posibilidad de realizar búsquedas personalizadas, filtrar documentos y ordenar los resultados por campos específicos. Para llevar a cabo estas consultas, MongoDB emplea el método "find" sobre la colección deseada o "findAndModify" para consultar y actualizar los valores de uno o más campos simultáneamente.
- Indexación en BBDD con MongoDB: MongoDB utiliza árboles B+ para indexar los datos almacenados en sus colecciones. Se trata de una variante de los árboles B con nodos de índice que contienen claves y punteros a otros nodos. Estos índices almacenan el valor de un campo específico, permitiendo que las operaciones de recuperación y eliminación de datos sean más eficientes.
- Coherencia en BBDD con MongoDB: A partir de la versión 4.0 (la más reciente es la 6.0), MongoDB soporta transacciones ACID a nivel de documento. La función "snapshot isolation" ofrece una visión coherente de los datos y permite realizar operaciones atómicas en múltiples documentos dentro de una sola transacción. Esta característica es especialmente relevante para las bases de datos NoSQL, ya que plantea soluciones a diferentes problemas relacionados con la consistencia, como escrituras concurrentes o consultas que devuelven versiones obsoletas

de un documento. En este aspecto, MongoDB se acerca mucho a la estabilidad de los RDMS.

- Seguridad en BBDD con MongoDB: MongoDB tiene un nivel de seguridad alto para garantizar la confidencialidad de los datos almacenados. Cuenta con varios mecanismos de autenticación, configuración de accesos basada en roles, cifrado de datos en reposo y posibilidad de restringir el acceso a determinadas direcciones IP. Además, permite auditar la actividad del sistema y llevar un registro de las operaciones realizadas en la base de datos.

Apache Cassandra

Es un SGBD orientado a columnas que fue desarrollado por Facebook para optimizar las búsquedas dentro de su plataforma. Uno de los creadores de Cassandra es el informático Avinash Lakshman que trabajó anteriormente con Amazon, formando parte del grupo de ingenieros que desarrolló DynamoDB. Por este motivo, no es extraño que comparta algunas características con este otro sistema.
En el año 2008 fue lanzado como proyecto open source y en 2010 se convirtió en un proyecto top-level de la Fundación Apache. Desde entonces Cassandra continuó creciendo hasta ser uno de los SGBD NoSQL más populares.
Aunque a día de hoy Meta utiliza otras tecnologías, Cassandra sigue formando parte de su infraestructura de datos. Otras empresas que lo utilizan son Netflix, Apple o Ebay. En términos de escalabilidad está considerada como una de las mejores bases de datos NoSQL.

Veamos algunas de sus características más destacadas:

- Almacenamiento en BBDD con Apache Cassandra: Cassandra utiliza un modelo de datos tipo "Column Family", que es similar a las bases de datos relacionales, pero más flexible. No se refiere a una estructura jerárquica de columnas que contengan otras columnas, sino más bien a una colección de pares clave-valor, donde la clave identifica una fila y el valor es un conjunto de columnas. Es un diseño pensado para almacenar grandes cantidades de datos y realizar operaciones de escritura y lectura más eficientes.
- Particionado en BBDD con Apache Cassandra: Para la distribución de datos Cassandra utiliza un particionador que reparte los datos en diferentes nodos del clúster. Este particionador usa el algoritmo "consistent hashing" para asignar una clave de partición única a cada fila de datos. Los datos que poseen la misma clave de partición estarán juntos en los mismos nodos. También admite nodos virtuales (vnodes), lo que significa que un mismo nodo físico puede tener varios rangos de datos.

- Replicado en BBDD con Apache Cassandra: Cassandra propone un modelo de replicado basado en Peer to peer en el que todos los nodos del clúster aceptan lecturas y escrituras. Al no depender de un nodo maestro para procesar las solicitudes, la posibilidad de que se produzca un cuello de botella es mínima. Los nodos se comunican entre sí y comparten datos utilizando un protocolo de gossiping.
- Consultas en BBDD con Apache Cassandra: Al igual que MongoDB, Cassandra también admite consultas ad-hoc, pero estas tienden a ser más eficientes si están basadas en la clave primaria. Además, dispone de su propio lenguaje de consulta llamado CQL (Cassandra Query Language) con una sintaxis similar a SQL, pero que en lugar de utilizar joins apuesta por la desnormalización de los datos.
- Indexación en BBDD con Apache Cassandra: Cassandra utiliza índices secundarios para permitir consultas eficientes sobre columnas que no forman parte de la clave primaria. Estos índices pueden afectar a columnas individuales o a varias columnas (SSTable Attached Secondary Index). Se crean para permitir consultas complejas de rango, prefijo o búsqueda de texto en un gran número de columnas.
- Coherencia en BBDD con Apache Cassandra: Al utilizar una arquitectura Peer to Peer Cassandra juega con la consistencia eventual. Los datos se propagan de forma asíncrona en múltiples nodos. Esto quiere decir que durante un breve periodo de tiempo puede haber discrepancias entre las diferentes réplicas. Sin embargo, Cassandra proporciona también mecanismos para configurar el nivel de consistencia. Cuando se produce un conflicto (por ejemplo, si las réplicas tienen versiones diferentes), utiliza la marca de tiempo (timestamp) y da por válida la versión más reciente. Además, realiza reparaciones automáticas para mantener la coherencia y la integridad de los datos si se presentan fallos de hardware u otros eventos que causan discrepancias entre las réplicas.
- Seguridad en BBDD con Apache Cassandra: Para utilizar Cassandra en un entorno seguro es necesario realizar configuraciones, ya que muchas opciones no están habilitadas por defecto. Por ejemplo, debemos activar

el sistema de autenticación y establecer permisos para cada rol de usuario. Además, es fundamental encriptar los datos en tránsito y en reposo. Para la comunicación entre los nodos y el cliente se pueden cifrar los datos en tránsito utilizando SSL/TLS.

Ventajas de las bases de datos NoSQL:

1. Escalabilidad horizontal:
    - Las bases de datos NoSQL están diseñadas para escalar horizontalmente, lo que significa que puedes agregar más servidores para distribuir la carga. Esto es ideal para aplicaciones que manejan grandes volúmenes de datos.
2. Flexibilidad en el esquema:
    - No requieren un esquema fijo como las bases de datos relacionales, lo que permite almacenar datos sin necesidad de definir previamente las estructuras (tablas). Esto es útil cuando los datos cambian con frecuencia o no tienen una estructura consistente.
3. Alto rendimiento:
    - Las bases de datos NoSQL están optimizadas para el rendimiento con operaciones rápidas de lectura y escritura, especialmente en aplicaciones que necesitan manejar grandes cantidades de datos en tiempo real.
4. Adecuadas para datos no estructurados o semiestructurados:
    - Son ideales para almacenar datos como documentos JSON, archivos multimedia, registros de logs, etc., que no se ajustan fácilmente a un modelo relacional.
5. Distribución y disponibilidad:
    - Están diseñadas para funcionar en entornos distribuidos, lo que permite una alta disponibilidad y tolerancia a fallos. Son utilizadas en aplicaciones que requieren estar siempre disponibles.

6. Variedad de modelos de datos:
    - NoSQL ofrece diferentes tipos de modelos de datos (clave-valor, columnares, grafos, documentos) que permiten ajustar la base de datos al tipo de aplicación específica.

Desventajas de las bases de datos NoSQL:

1. Falta de estandarización:
    - No existe un estándar unificado para las bases de datos NoSQL, lo que significa que las características y las APIs pueden variar significativamente entre distintos proveedores y sistemas.
2. Consistencia eventual:
    - Muchas bases de datos NoSQL priorizan la disponibilidad sobre la consistencia (modelo de consistencia eventual). Esto puede causar problemas si la aplicación necesita una consistencia de datos estricta.
3. Mayor complejidad en las consultas:
    - A diferencia de SQL, las bases de datos NoSQL no tienen un lenguaje de consulta estándar universal (como SQL). Esto puede hacer que las consultas complejas sean más difíciles de implementar y menos eficientes.
4. Falta de transacciones ACID completas:
    - Aunque algunos sistemas NoSQL ofrecen algún nivel de transacciones, generalmente no garantizan las propiedades ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad) de manera completa, lo que puede ser problemático para aplicaciones que requieren transacciones consistentes y seguras.
5. Curva de aprendizaje:
    - Migrar o comenzar a trabajar con una base de datos NoSQL puede requerir una curva de aprendizaje importante, ya que el

diseño y la optimización son muy diferentes de las bases de datos relacionales.

6. Compatibilidad limitada con sistemas existentes:
    - Muchas aplicaciones empresariales están diseñadas para trabajar con bases de datos relacionales, por lo que puede ser difícil integrar una base de datos NoSQL en entornos legados sin realizar cambios significativos.

 [1] "Bases de Datos NoSQL: Qué son, Tipos y Características," **Pandora FMS**, Jul. 15, 2021. [Online]. Available: https://pandorafms.com/blog/es/bases-de-datos-nosql/. [Accessed: Aug. 24, 2024].

 [2] "Bases de Datos Orientadas a Objetos: Qué son, Ejemplos y Ventajas," **Ayuda Ley Protección Datos**, [Online]. Available: https://ayudaleyprotecciondatos.es/bases-de-datos/orientas-a-objetos/#google_vignette. [Accessed: Aug. 24, 2024].