Tarea 1

Modelo Orientado a Objetos

¿Qué es?

Un modelo de base de datos orientado a objetos (Object-Oriented Database Model, OODBMS) almacena datos como *objetos*, en lugar de tablas como en los modelos relacionales. Cada objeto incluye:

- atributos (estado)
- métodos (comportamiento)
- relaciones entre objetos
- herencia, polimorfismo, encapsulación etc.

Los objetos tienen identificadores únicos (OID) asignados automáticamente.

Ventajas

1. **Correspondencia con programación orientada a objetos**: Si usas lenguajes OO (Java, C++, Python, etc.), el modelo de objetos en la base de datos puede mapear directamente con los objetos en el código, reduciendo la necesidad de traducción entre modelo de datos y modelo en la aplicación.
2. **Manejo eficiente de datos complejos**: Bueno para estructura de datos ricas, con muchas relaciones, objetos anidados, colecciones, etc.
3. **Reutilización de código**: Herencia, polimorfismo, modularidad.
4. **Integridad de la identidad del objeto (OID)**: los objetos tienen identidad propia independiente de sus atributos. Permite referencias directas. jot.fm+1

Desventajas

1. **Poca adopción / ecosistema pequeño**: Menos herramientas, menos soporte, menos comunidad en comparación con modelos relacionales o NoSQL ampliamente usados.
2. **Curva de aprendizaje**: Para quienes vienen de modelos relacionales puede ser difícil adaptarse. Lenguajes de consulta menos estándar.

3. **Menos compatibilidad con SQL/reportes estándar**: Muchas herramientas de BI, de reporting, etc., están diseñadas para SQL. Puede ser más difícil integrarse.
4. **Potencialmente costoso para consultas simples / overhead**: Si los datos no son tan complejos, la estructura orientada a objetos puede introducir sobrecarga, lentitud o complejidad innecesaria.

Casos de uso

- Sistemas de ingeniería, CAD/CAM donde los objetos tienen muchas propiedades, métodos, relaciones complejas.
- Aplicaciones científicas o de simulación con estructuras de datos complejas.
- Sistemas donde la correspondencia entre objetos del dominio y los objetos almacenados sea importante.
- Multimedia, gráficos, juegos, modelado 3D, donde los objetos no son simples filas de datos.

Modelos NoSQL

NoSQL = Not Only SQL. Son bases de datos no relacionales, que no usan el esquema rígido de tablas y que están diseñadas para escalabilidad, flexibilidad, grandes volúmenes de datos, datos no estructurados o semiestructurados.

Hay principalmente **cuatro tipos**:

1. Clave-valor (Key-Value)
2. Documentales (Document stores)
3. Columnar o Familia de columnas (Wide-column / Column-oriented)
4. De grafos (Graph databases)

Tipos, ventajas, desventajas y casos de uso

Aguilera Ferrusca Patricio
Bases de datos
Tarea 1

Tipo	Descripción	Ventajas principales	Desventajas principales	Casos de uso
Clave-valor	Almacenan pares clave-valor; la aplicación busca con la clave. El valor puede ser cualquier estructura.	+ Muy simple, rendimiento alto (lectura/escritura) + Escalabilidad horizontal fácil + Buena para cachés, sesiones, datos temporales	− Consultas complejas difíciles (filtrar por atributos, rangos, relaciones) − No hay esquemas ricos − A veces poca flexibilidad para agregaciones o relaciones	Cachés, almacenamiento de sesiones, configuración de usuario, datos simples que se acceden por clave
Documentales	Datos almacenados como documentos (JSON, BSON, XML, etc.). Colecciones de documentos; cada documento puede tener estructura distinta.	+ Flexibilidad de esquema: documentos pueden variar en estructura + Soporte para búsquedas y consultas dentro de documentos + Indexación sobre atributos, subdocumentos + Adecuados para aplicaciones ágiles donde los requisitos cambian rápido	− Consistencia transaccional puede no ser tan fuerte − Consultas muy complejas / join entre colecciones pueden ser costosas o difíciles − Redundancia de datos (denormalización) puede requerir más espacio − Gestión de esquemas heterogéneos puede dificultar mantenimiento	Aplicaciones web / móvil, CMS, e-commerce, catálogos de productos, logs, aplicaciones con datos semiestructurados
Columnar / familia de columnas	Organizan los datos por columnas (o familias de columnas), no por filas. Ej: Apache Cassandra, HBase.	+ Excelente para grandes cantidades de datos, lecturas/escrituras en masa + Buen rendimiento para consultas que solo requieren algunas columnas + Alta escalabilidad	− No tan eficientes para transacciones complejas − Mayor latencia para accesos a muchas columnas si no se diseña bien − Modelado puede requerir pensar cuidadosamente cuál	Big data, análisis de logs, métricas, series temporales, aplicaciones que necesitan altas tasas de escritura y lectura parciales, data warehouses distribuidos

Tipo	Descripción	Ventajas principales	Desventajas principales	Casos de uso
		horizontal + A menudo tolerancia a fallos, replicación distribuida	será el patrón de acceso	
Grafos	Se centran en relaciones entre entidades: nodos y aristas (edges). Ideal para datos conectados.	+ Excelentes para consultas que atraviesan muchas relaciones (ej: "¿qué amigos de mis amigos visitaron X?") + Modelado natural de redes sociales, dependencias, recomendaciones + Alta expresividad para relaciones	− Escalan peor que modelos más simples en algunos casos − Consultas globales pueden ser costosas en grafos enormes − Operaciones de escritura pesadas pueden afectar rendimiento − Aprendizaje y herramientas menos universales que los modelos relacionales / documentales	Redes sociales, sistemas de recomendación, detección de fraude, análisis de relaciones, seguimiento de rutas, knowledge graphs

---

Comparativa OO vs NoSQL

- **Estructura vs flexibilidad**: OO es bueno para estructuras muy definidas (aunque permite herencia etc.), NoSQL permite esquemas flexibles y cambios rápidos.
- **Consulta de relaciones complejas**: Los grafos de NoSQL y OO tienen ventaja para relaciones complejas; los documentos también pueden, pero con límites.
- **Escalabilidad**: NoSQL suele estar diseñado para escalabilidad horizontal (varios nodos), mientras OO puede tener limitaciones para crecer mucho dependiendo de la implementación.
- **Consistencia / transacciones**: OO tiende a soportar mejor ACID; muchos NoSQL sacrifican algo de consistencia para disponibilidad o

particionamiento (CAP theorem) aunque algunos proporcionan transacciones distribuidas actualmente. MongoDB+1

- **Adopción y herramientas**: NoSQL tiene gran masa de adopción, comunidad creciente; OO más de nicho.

Cuando usar qué

Aquí unos criterios/principios para decidir:

- Si la aplicación tiene muchas relaciones complejas entre objetos, comportamientos, lógica del dominio orientado a objetos: considerar OO.
- Si necesitas almacenar grandes volúmenes de datos semiestructurados o no estructurados, que cambian con el tiempo, con acceso rápido, NoSQL puede ser mejor.
- Si requieres transacciones fuertes, consistencia estricta, integridad referencial, posiblemente OO o base de datos relacional + OO.
- Si la escalabilidad, alta disponibilidad, tolerancia a fallos, particionamiento geográfico son importantes → NoSQL quizá sea mejor.
- También hay casos híbridos: combinar modelos, usar "multi-modelo" o usar OO para parte de la aplicación y NoSQL para otras.

Bibliografía

[2] Pandora FMS, "Bases de datos NoSQL: Guía con las ventajas y desventajas," *Blog Pandora FMS*, 2022. [En línea]. Disponible en: https://pandorafms.com/blog/es/bases-de-datos-nosql/

[3] Arsys, "Bases de datos NoSQL: qué son, tipos y ventajas," *Blog Arsys*, 2023. [En línea]. Disponible en: https://www.arsys.es/blog/bases-de-datos-nosql-que-son-tipos-y-ventajas

[4] Actian, "Tipos de bases de datos, ventajas e inconvenientes y casos de uso," *Blog Actian*, 2023. [En línea]. Disponible en: https://www.actian.com/es/blog/databases/types-of-databases-pros-cons/

Aguilera Ferrusca Patricio
Bases de datos
Tarea 1