Apuntes-SI1-Tema-3.pdf

SalvaGrados

Sistemas Informaticos I

3º Grado en Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid

GESTORES DE BASES DE DATOS

Sistema que se encarga de la organización, almacenamiento, gestión y recuperación (eficiente) de la información. También llamado SGBD (Sistema de Gestión de Base de Datos)**.**

Incluye:

- Estructuras de almacenamiento de la información optimizadas para trabajar con un gran volumen de datos.
- DDL: Data Definition Language. Un lenguaje para modelar la información de acuerdo a un determinado modelo (DDL,)
- DML: Data Manipulation Language. Un lenguaje para recuperar/manipular la información almacenada mediante búsquedas dirigidas.
- Mecanismos adecuados que le permitan integrarse en un sistema de acceso con control transaccional

Características:

- Normalmente utiliza el **modelo relacional**, utilizando el lenguaje SQL (Es un estándar no un lenguaje, hay muchas implementaciones).

Problemas que resuelve:

- Volumetría: se pueden manejar muchos datos, no como con ficheros.
- Redundancia: al ser relacional.
- Acceso a datos ineficiente: no hay información duplicada.
- Datos no aislados: no hay que modificar el programa para modificar datos.
- Integridad: Pueden ponerse restricciones en la propia base de datos (Ej: edad > 0).
- Atomicidad: Las modificaciones son atómicas.
- Acceso simultáneo: por varios usuarios.
- Seguridad.

DDL y DML

DDL: Creación y destrucción de tablas, definición de restricciones.

- Definición de esquemas de relación.
- **Borrado** de relaciones.
- **Creación** de índices.
- **Modificación de esquemas** de relación.
- Órdenes para la definición de vistas.
- Órdenes para especificar las **restricciones de integridad** que deben cumplir los datos almacenados en la base de datos.
- Órdenes para especificar **derechos de acceso** para las relaciones y vistas.
- Check, Not Null, Unique, Primary Key, Foreign Key

DML: Modificación de tablas, Consultas CRUD sencillas.

- Incluye un lenguaje de **consultas**, basado en el álgebra relacional (y en el cálculo de tuplas)
- Incluye órdenes para **insertar, borrar, modificar y seleccionar (CRUD)** tuplas de la base de datos.
- INSERT, SELECT, UPDATE, DELETE/TRUNCATE

ACCESO A SQL DESDE APLICACIÓN

Diferentes tipos de acceso:

SQL Interactivo:

Uso de SQL en el cliente del SGBD. Sirve para probar consultas, definir la estructura, etc. No une con el programa.

Middleware:

Driver de la base de datos (ODBC, JDBC). Específico del SGBD y el lenguaje de alto nivel en el que está programada la aplicación. Define una API para comunicarse entre el lenguaje de alto nivel y la base de datos, y de su conexión y desconexión.

Tipos:

- SQL embebido:
  Las sentencias SQL están incrustadas dentro del propio código. Se construyen como Strings (dinámicos) que se pasan al SGBD a través del driver. Poco seguro.

- Sentencias preparadas:
  Sentencia SQL precompilada que acepta parámetros. Mejora el tiempo de respuesta y/o la seguridad. Útil cuando una misma sentencia se utiliza muchas veces. Reduce el envío de datos desde la aplicación

- DataSources lógicos.
  Sustituye código SQL por código alto nivel. Como patrón de diseño mejora la reusabilidad/mantenibilidad. Como herramienta de acceso a datos mejora los tiempos de acceso.

- Separación de responsabilidades (separation of concerns) en el diseño
  Mejora de la legibilidad y mantenibilidad del código. Frameworks y bibliotecas que permiten que las sentencias SQL no aparezcan inmersas en el código funcional.

- ORM: Object-Relational Mapping
  Abstracción del acceso a datos. Frameworks de persistencia que almacenan y recuperan objetos de una BBDD relacional de forma transparente para el programador de la lógica de negocio. Los programas invocan a la capa de persistencia como a cualquier otro elemento de la lógica de negocio (el código SQL no existe).

Ejemplos:

- MyBatis: ORM
  Framework de persistencia que soporta SQL, procedimientos almacenados y mapeos avanzados. MyBatis elimina casi todo el código JDBC, puede configurarse con XML o anotaciones y permite mapear mapas y POJOs (Plain Old Java Objects).

- Hibernate ORM: ORM
  Hibernate ORM enables developers to more easily write applications whose data outlives the application process.

WUOLAH + BBVA

1/6	BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.
Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.	

Te regalamos

15€

1

Abre tu Cuenta Online **sin comisiones ni condiciones**

2

Haz una compra igual o superior a 15€ **con tu nueva tarjeta**

3

BBVA te devuelve un **máximo de 15€**

- SQLAlchemy: ORM
  SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

- Django: ORM
  Clases que heredan de django.db.models.Models. Representa una entidad y definen todos sus campos y relaciones. El modelo de datos lógico se convierte automáticamente en un modelo de datos físico en el SGBD. Una vez cargado en la base de datos, django ofrece una funcionalidad estándar para trabajar con los datos mediante un patrón DAO

APLICACIONES DENTRO DE LA BASE DE DATOS

Se puede añadir funcionalidad en la base de datos. No hay un estándar, cada SGBD lo hace diferente. Consideramos dos formas:

- Funciones y procedimientos almacenados: Se ejecutan a petición del usuario.
- Triggers: Se ejecutan cuando ocurre un evento asociado a una tabla (p.ej., una inserción o una actualización)

Al igual que cualquier otra metaestructura se gestionan con los comandos:

- CREATE
- ALTER (habitualmente se usa DROP + CREATE)
- DROP.

Ventajas:

- Mejoras de rendimiento frente a la ejecución de comandos SQL desde la aplicación.
- Su acceso está controlado por los mecanismos de seguridad.
- Aceptan parámetros de entrada.
- Los procedimientos se invocan, las funciones se incluyen dentro de una sentencia SQL.
- La sintaxis se valida en tiempo de ejecución, no durante la creación.
- Al procesarse la funcionalidad en la BD, pueden ahorrarse varias llamadas (ej necesitas el resultado de una consulta para pedir otra).

Triggers:

Tipo especial de procedimiento almacenado. Se invoca de forma automática en respuesta a una modificación de datos en una tabla. A la hora de crear un trigger se debe especificar el evento que los dispara.

VOLUMETRÍA EN EL ACCESO A DATOS

Cada vez que ejecutamos una sentencia SQL en un SGBD, éste crea el plan de ejecución (explain plan) de la sentencia.

Explain Plan:

- Define la forma en que el SGBD busca o inserta los datos: Formas de cruzar tablas, uso de índices, orden de ejecución de subconsultas, etc.
- La información suministrada por el explain plan permite identificar **cuellos de botella** en la ejecución de una consulta. Índices, cruces y estructura de sentencias.

Funcionamiento del Explain Plan:

1. ANALYZE: Comando para actualizar la información que el motor tiene sobre el contenido de las tablas. Se hace cada vez que hay una gran actualización.
2. EXPLAIN: Devuelve el plan de acceso de una sentencia SQL en forma de tabla.
   Cada fila contiene información de las tablas (físicas o no) empleadas en la consulta.
   El orden de las tablas indica el orden en el que se procesarian en la consulta (a tener en cuenta de cara a la optimización)
3. SHOW WARNINGS: Mensajes adicionales del optimizador
4. EXPLAIN ANALYZE: Indica la estrategia de acceso.

Estrategias de acceso:

- Directo/Constante (CONST)
  Tablas con un solo registro.
  Por valor en índice.
  Select en tabla con primary key.
- Cruce por clave única (EQ_REF)
- Clave no única (REF)
- Merge de índices (INDEX_MERGE)
- Clave única en subconsulta (UNIQUE_SUBQUERY)
- Clave no única en subconsulta (INDEX_SUBQUERY)
- Rango en índice (RANGE)
  =, <>, >, >=, <, <=, IS NULL, BETWEEN, LIKE o IN
- Full index scan (INDEX)
- Full table scan, secuencial (ALL)

Mejoras al rendimiento:

- Insert todo a la vez en vez de varios inserts separados.
- Select añadiendo antes una Primary Key (Índice implícito):
  Pasa de Full Table Scan a Constant.
- Select añadiendo un Índice Explícito:
  Pasa de Full Table Scan a Clave no única.
- Select añadiendo un Índice Explícito y buscando por rango (Ej 'Indiana Jones %' para buscar todas las películas de Indiana Jones):
  Mejora sólo si no hay rango a la izquierda (Ej '% Jones %') porque no estaría ordenado.
  Pasa de Full Table Scan a Clave no única + Rango en Índice.
- En vez de un Join (Que a lo mejor tiene que hacer un full table scan para comparar) hacer una subquery que primero disminuya los elementos a buscar y luego el full table scan de esos elementos que son menos.

BIG DATA

Significa obtener información útil a partir de volúmenes de datos tan grandes que hasta hace pocos años no era posible su adquisición y procesamiento. "Big Data" es similar a "Small Data", solo que con mayores volúmenes de datos. La diferencia de tamaño requiere de soluciones diferentes.

Razones:

- Incremento exponencial en la cantidad de datos generados y disponibles
- Revolución tecnológica/social: Web 2.0 y 3.0 4.0, Smartphones, Internet de las cosas
- Aparición de tecnologías de bajo coste que permiten su almacenamiento y procesamiento

3 V 's del Big Data: Velocidad, Variedad y Volumen.
5 V 's del Big Data: Velocidad, Variedad, Volumen, Veracidad y Valor.

Problemas:

- Volumetría de datos: En un futuro habrá demasiados datos.
- Los modelos tradicionales de procesamiento y almacenamiento (vigentes desde los 70) no son suficientes para algunas casuísticas.
- Surgen dos necesidades, interrelacionadas:
    - Procesamiento distribuido.
    - Almacenamiento distribuido.
      Entre otras cosas, implica que los datos pueden estar en sitios físicos distintos y posiblemente replicados.

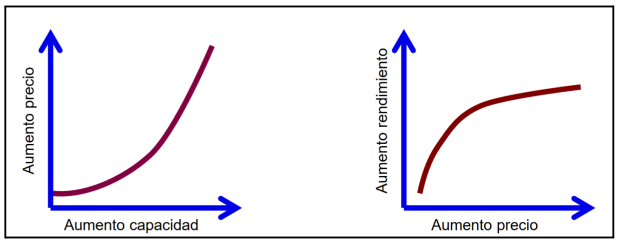
ESCALABILIDAD

V de volumen y V de velocidad. Aumentar la capacidad de almacenamiento y procesamiento de los sistemas.

Tipos:

- Vertical (scale up)
  Incrementar la potencia de la máquina en la que se ejecuta el software, ya sea nuevos y mejores componentes o cambiar el ordenador completo.
  Ventajas:
    - ❖ Es más simple si el software está preparado

  Desventajas
    - ❖ Por más preparado que esté el software, más temprano que tarde encontraremos limitaciones.
    - ❖ Es muy caro.

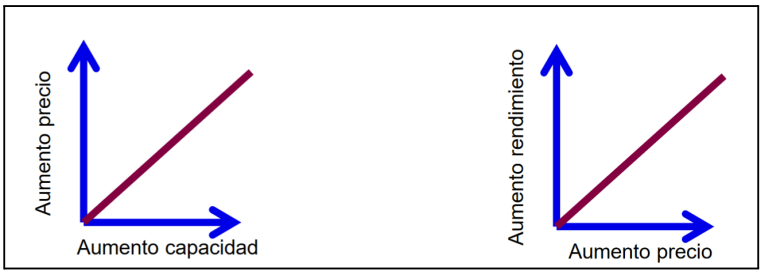


- Horizontal (scale out)
  Distribuir la carga de trabajo entre varios ordenadores conectados entre sí. Normalmente estos ordenadores son de gama baja/media (más baratos).
  Ventajas:
    - ❖ Los límites son mucho más altos (potencialmente miles de ordenadores conectados)

- ❖ Cuando más capacidad, más barato respecto a la escalabilidad vertical
- ❖ Normalmente escalabilidad lineal: si duplico el número de ordenadores, duplicó el rendimiento (predictibilidad)

Desventajas:

- ❖ Requiere de software específicamente diseñado e implementado para ejecutarse en varios ordenadores a la vez (procesamiento distribuido)



TEOREMA CAP

La escalabilidad horizontal implica cierta probabilidad de que falle uno de los nodos conectados o la comunicación entre ellos.

Tres propiedades deseables:

- Consistencia: para resumir, que todos los nodos contengan valores consistentes entre sí en todo momento
- Disponibilidad (Availability): garantía de que cada petición a un nodo reciba una confirmación de si ha sido o no resuelta satisfactoriamente
- Tolerancia al Particionado: que pueda fallar un nodo o conexión y el sistema siga funcionando

El teorema CAP establece la imposibilidad de que un sistema ofrezca las tres propiedades simultáneamente, solo pudiendo cubrirse simultáneamente dos de ellas:

- Sistemas CA: SGBDR.
- Sistemas CP: mayoría de BBDD NoSQL (Ej.: MongoDB).
- Sistemas AP: Apache Cassandra.

PROCESAMIENTO BIG DATA

El procesamiento distribuido en un contexto big data requiere el uso de modelos computacionales no estándar. El primero en hacerlo fue Hadoop, utilizando el modelo MapReduce.

Características Hadoop:

- Cubre necesidades de almacenamiento y procesamiento masivo de datos
- Las tareas se ejecutan en una red (cluster hadoop) de ordenadores conectados entre sí (nodos) que se reparten la tarea.
    - La suma total de capacidad de proceso y almacenamiento es igual a la suma de capacidad de proceso y almacenamiento de cada uno de sus nodos. Esto dota al sistema de escalabilidad horizontal y capacidad de crecer según las necesidades.
- HDFS: HADOOP DISTRIBUTED FILE SYSTEM

Permite aprovechar y trabajar con la capacidad total de almacenamiento de todos los ordenadores a la vez, mostrándola como si fuera uno solo.
Es un sistema de almacenamiento tolerante a fallos (mediante replicación).

MapReduce:
Un problema objetivo debe ser paralelizable divisible en subtareas que puedan ser ejecutadas por separado

1. Primero se divide el problema en problemas menores (etapa Map)
2. Luego los problemas más pequeños son resueltos paralelamente
3. Finalmente, el conjunto de soluciones a los problemas menores es sintetizado en una solución al problema original (etapa Reduce).

Limitaciones de MapReduce:

- Complejidad: Aunque reduce la dificultad de la programación paralela, su implementación
- es a bajo nivel y no trivial.
- Rigidez: Las soluciones siempre se deben expresar en dos etapas con semántica muy estricta. En el extremo, cierto tipo de problemas no pueden ser solucionados con este método.
- Antigüedad

Una alternativa más moderna a Hadoop es Apache Spark.

BASES DE DATOS NO SQL
Características:

- Bases de datos sin esquemas
- Mayormente utilizan interfaces distintas al SQL
- En general dan soporte al almacenamiento de grandes cantidades de datos mediante escalabilidad horizontal
- Operan sobre infraestructuras distribuidas, como Hadoop
- Tratamiento de la información rápido y flexible.

Tipos de BBDD NoSQL:

- Basadas en pares clave-valor:
  Utilizan combinaciones nombre:valor, normalmente en memoria y de acceso rápido.
  Ejemplo: Redis.
- Basadas en grafos:
  Utilizan un grafo para establecer conexiones entre datos, así como mecanismos de consulta más eficientes.
  Ejemplos: Neo4j y JanusGraph.
- Basadas en columnas:
  Utilizan filas y columnas, pero con nombres y formatos variables entre filas. Pueden verse como BBDD basadas en clave-valor bidimensionales.
  Ejemplos: Cassandra y Hbase.
- Basadas en documentos:

Utilizan documentos en vez de tablas, y se caracterizan por su gran flexibilidad y capacidad de almacenamiento. Ejemplos: CouchBase y MongoDB.

Características de las basadas en documentos:

- Modelo clave-valor, pero permitiendo el uso de metadatos para aportar mayor expresividad.
- La unidad organizativa de la información es el documento, que goza de alta flexibilidad. Cada uno consta de un ID único.
- Los datos se agrupan en colecciones y documentos, que serían como las tablas y las filas, respectivamente, en las BBDD Relacionales.
- Suelen basarse en el formato JSON preferentemente, si bien pueden usar también XML.
- La principal ventaja es la flexibilidad, ya que no tienen estructuras predefinidas. Podemos tener documentos diferentes entre sí. Esto permite:
    - Cambios ágiles, sin tener que modificar estructuras internas predefinidas.
    - Consultas más naturales
    - Reducción de la verbosidad
- Propensas a errores de introducción de datos, por lo que es necesario implementar métodos de saneado y limpieza de datos.
- Ejemplo: MongoDB. Funciona bien para grandes cantidades de datos almacenados, escalado horizontal sencillo. No es adecuado para transacciones complejas.

Características de las basadas en grafos: Neo4J

- La representación y almacenado de los datos es en forma de grafo
- Se dice que es “whiteboard friendly”: lo que dibujas como cajas y líneas en una pizarra lo almacenas directamente en Neo4j.
- Neo4J se centra más en las relaciones entre los datos que en los aspectos comunes entre conjuntos de datos (tales como tablas de filas o colecciones de documentos).
- Define un lenguaje propio (Cypher) para manipulación de los datos, pero existen varios lenguajes capaces de interactuar con Neo4J: Java code, REST, Ruby console, Gremlin y otros.