

Apuntes-SI1-Tema-3.pdf

[**SalvaGrados**](https://wuolah.com/perfil/SalvaGrados)

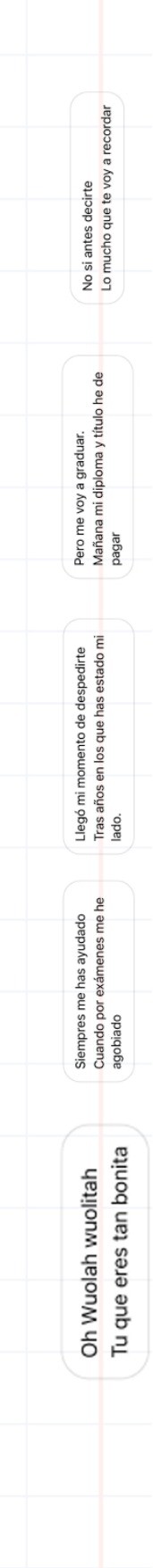
**Sistemas Informaticos I**

**3º Grado en Ingeniería Informática**

**Escuela Politécnica Superior Universidad Autónoma de Madrid**

Reservados todos los derechos.

**No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.**



GESTIÓN DE DATOS DISTRIBUIDOS

**3.1 INTRODUCCIÓN**

**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:

* El modelo más habitual de gestores de bases de datos es el que sigue el modelo relacional • El lenguaje más utilizado actualmente para modelar y recuperar la información es SQL (Structured Query Language) • Aunque hay una definición ISO para SQL el lenguaje soportado por cada SGBD suele ser ligeramente diferente

**SOBRE EL ACCESO A LOS DATOS**

Originalmente las aplicaciones se construían sobre un conjunto de ficheros en los que se

almacenaban los datos de gestionaban

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.

**3.2 BASES DE DATOS RELACIONALES EN ENTORNOS DISTRIBUIDOS**

**MODELO RELACIONAL**

Informalmente, una relación puede considerarse una tabla (conceptualmente)

• Una base de datos que se ajusta al modelo relacional puede representarse como un conjunto de tablas

• Convertir un diagrama E-R a tablas es el primer paso para obtener una base de datos relacional

• Normalmente cada entidad y cada relación muchos a muchos da lugar a una tabla

• Cada tabla tienen un conjunto de columnas que suelen corresponderse con los atributos



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

**DDL y DML**

DDL: Creación y destrucción de tablas, definición de restricciones. Lenguaje de definición de datos (Data Definition Language)

* 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. Lenguaje de manipulación de datos (Data Manipulation Language)
* 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.

Texto

Descripción generada automáticamente

**AGREGACIONES Y AGRUPACIONES**

• Operadores que calculan un valor único a partir de una columna de valores: SUM, AVG, MIN, MAX, COUNT

SELECT COUNT(\*) FROM tabla;

SELECT COUNT(campo) FROM tabla;

SELECT COUNT(DISTINCT campo) FROM tabla;

Diagrama

Descripción generada automáticamente• Las agregaciones se puede realizar sobre todos los campos que cumplen los criterios de filtrado o agrupando registros según alguno de sus atributos

SELECT campo1, …, campo\_n SUM(campo) FROM table

GROUP BY campo1, …, campo\_n HAVING condicion

**CRUCES DE TABLAS**

• Gran parte de la potencia de las bases relacionales se basa en la posibilidad de combinar/cruzar dos (o más) relaciones ◊ producto (join)

• En un cruce se toman dos o mas relaciones y se obtiene otra relación, en principio con todos los atributos de las relaciones que se han cruzado

• El tipo de cruce más simple es el producto cartesiano: todos con todos

• Enumerando cada relación en la clausula FROM

• Otros cruces tipos de cruces presentan sentencias específicas (depende del SGBD)

**COMBINACIÓN DE RELACIONES**

• Para poder combinar dos o más relaciones deben ser compatibles:

• UNION: unión de relaciones

• INTERSECT: intersección de relaciones. Dependiente de SGBD

• EXCEPT: resta relaciones. Dependiente de SGBD • Estos operadores eliminan los duplicados

• Si se usa ALL los duplicados no se eliminan: p.ej., UNION ALL

• Ejemplo: Actores comunes a las películas Star Trek IV y Star Trek V

Texto, Carta

Descripción generada automáticamente

**SUBCONSULTAS**

• Hasta ahora las condiciones en WHERE involucraban valores escalares, pero pueden aparecer subconsultas como parte de la condición descrita en una clausula WHERE

• Pueden devolver un valor (S)

• s op ALL R, op = {,<>,=, …}: cierto si se cumple para todos los valores de R

• s op ANY R, op = {,<>,=, …}: cierto si se cumple para al menos un valor de R

• Pueden devolver una relación (R) que será procesada valor por valor:

• S IN R: cierto si S está en R

• EXISTS R: cierto si R es una relación no vacía

• También es posible usar una subconsulta en lugar de una relación ya almacenada en la clausula FROM

• Dependiendo del SGBD puede ser obligatorio proporcionar un alias para la subconsulta

• La sintaxis para proporcionar el alias también puede variar ligeramente dependiendo del SGBD

Escala de tiempo

Descripción generada automáticamente con confianza baja

Obtener todos los actores que han sido el actor principal en al menos 10 películas ordenado por

numero de veces que han sido la estrella

Texto

Descripción generada automáticamente

**PROCESAMIENTO DE UNA CONSULTA**

1. Crear el producto cartesiano descrito en FROM

2. Aplicar las restricciones descritas en WHERE

3. Si no hay GROUP BY, proyectar la relación (2) según describe SELECT ◊ Fin

4. Agrupar las tuplas por valores tal y como especifica por GROUP BY

5. Aplicar HAVING 6. Aplicar SELECT –> Fin

Texto

Descripción generada automáticamente

## 3.3 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.

• Las aplicaciones no acceden a datos almacenados en BBDD relacionales a través de SQL interactivo, sino a través de un middleware, el 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 de conexión común:

• Conexión a la base de datos

• Manipulación/acceso a datos. Hay flujo de información:

• Desde el lenguaje de alto nivel hacia la base de datos

• Desde la base de datos hacia el lenguaje de alto nivel

• Desconexión de la base de datosTipos:

**EVOLUCIÓN DE LOS MECANISMOS DE ACCESO A DATOS**

* 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).

**SQL EMBEBIDO EN PHP**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza media

Algunos métodos implementados por la clase PDOStatement:

• fetch: retorna la próxima fila • fetchAll: retorna un array con todas las filas

• fetchColumn: retorna un solo campo de la próxima fila

• fetchObject: retorna la próxima fila como un objeto • execute: ejecuta una sentencia preparada

• bindColumn: liga una columna a una variable PHP

• Los resultados pueden ser devueltos (fila a fila) de distinta forma según se especifique con el método setFetchMode

• PDO::FETCH\_NUM: array indexado numéricamente

• PDO::FETCH\_ASSOC: como array asociativo

• PDO::FETCH\_OBJ: como un objeto Texto

Descripción generada automáticamente

• Para cerrar la conexión hay que destruir el objeto ◊ eliminar toda referencia al objeto

• Automáticamente al finalizar la ejecución del script

• También se puede cerrar asignado un valor null a la referencia

Gráfico

Descripción generada automáticamente

SQL EMBEBIDO EN JAVA

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente



Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

• java.sql.DataSource = factoría de conexiones al origen de datos físico

• Incluido en la JDK a partir de la versión 1.4

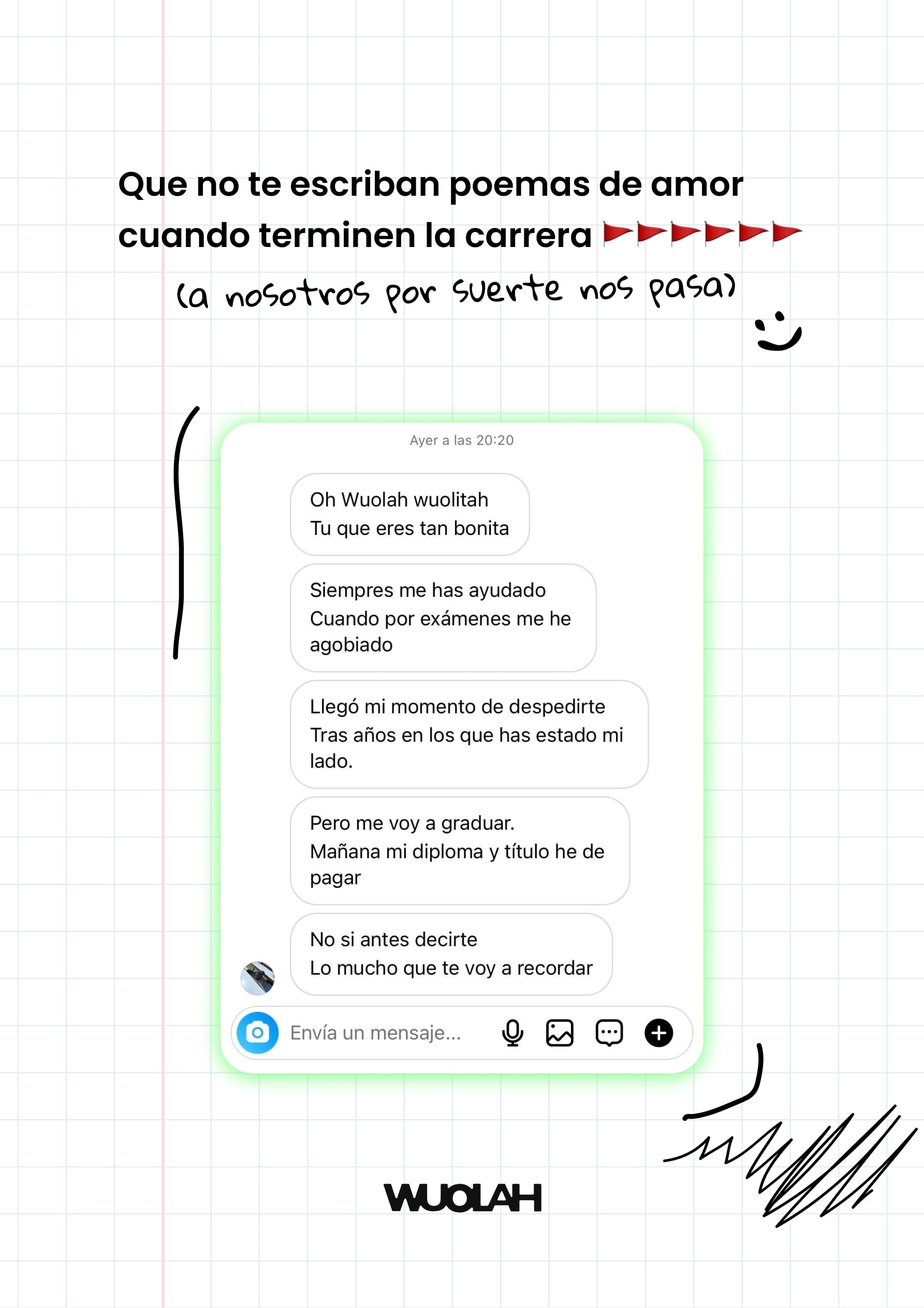
• Es la forma recomendada de obtener una conexión con la base de datos ◊ ofrece ventajas frente a obtenerla del DriverManager (depende de la implementación del driver)

• Facilidades para el registro y la inyección de dependencias

• Pool de conexiones • Transacciones distribuidas

Texto

Descripción generada automáticamente



SEPARACIÓN DE RESPONSABILIDADES

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.

Res

* 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

dent

**3.4 PROCEDIMIENTOS ALMACENADOS Y TRIGGERS**

## 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)

• Conjunto de sentencias SQL y lógica de programa compilado, verificado y almacenado en el SGBD

• Tratado por el servidor como cualquier otro objeto de la BBDD y almacenado en el catálogo de la misma

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

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

• No todos los SGBD los soportan:

• No hay estándares: implementación propia de cada fabricante

• PL/SQL (Procedural Language for SQL) definición de lenguaje incorporado en Oracle 6 que posteriormente ha sido asumida por otros fabricantes (p.ej., IBM DB2)

• PostgreSQL soporta funcionalidad dentro de la base de datos desde 1997

• El cuerpo de la función es una cadena de caracteres que se pasa al SGBD indicándole el lenguaje en el que está escrito

• Soportados por omisión:

• SQL

• PL/pgSQL

• C, solo usuarios privilegiadas

• A nivel administrativo se pueden cargar módulos con otros lenguajes:

• PL/PERL

• PL/TCL

• PL/Python

• …

**FUNCIONES Y PROCEDIMIENTOS ALMACENADOS**

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).

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

FUNCIONALIDAD EN POSTGRESQL

Imagen que contiene Diagrama

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Escala de tiempo

Descripción generada automáticamente

Texto, Carta

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Escala de tiempo

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

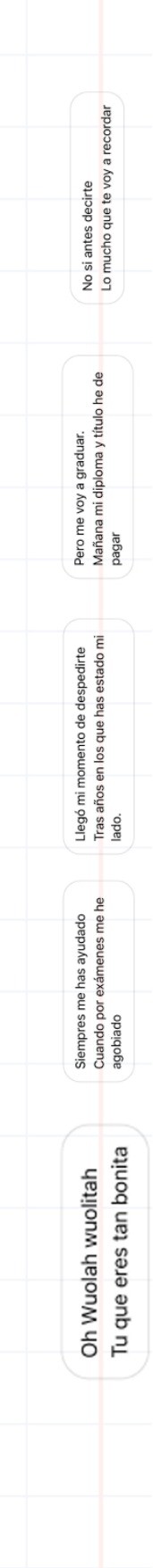
Imagen que contiene Tabla

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza media





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.

Características:

• Mecanismo alternativo para validar la integridad de los datos

• Ofrecen una funcionalidad equivalente a un planificador de tareas dentro de la propia BBDD

• Mecanismo sencillo para realizar una auditoría de datos independiente de la aplicación

• Se pueden considerar un tipo especial de procedimiento almacenado

• La principal diferencia es que un trigger 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

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

**3.5 OPTIMIZACIÓN DE CONSULTAS**

## 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.

Res

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)

* 1. SHOW WARNINGS: Mensajes adicionales del optimizador
  2. 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.

• Analizamos en detalle el caso de MySQL, pero hay comandos equivalentes en prácticamente todos los SGBD

• 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 procesarían en la consulta (a tener en cuenta de cara a la optimización)

• SHOW WARNINGS ◊ mensajes adicionales del optimizador

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente• EXPLAIN ANALYZE



BD DE PRUEBA

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza media

Texto, Carta

Descripción generada automáticamente

Tabla

Descripción generada automáticamente

JOINS

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Texto, Carta

Descripción generada automáticamente

Texto, Carta

Descripción generada automáticamente

Texto, Carta

Descripción generada automáticamente

**3.6 ENTORNOS DE COMPUTACIÓN DISTRIBUIDA PARA GRANDES VOLÚMENES DE DATOS**

**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 simil

• El objetivo es resolver problemas nuevos y mejorar la solución de problemas viejos:

• Sistemas de recomendación (Amazon)

• Escuchar las redes sociales (sentiment analysis)

• Predecir eventos y actuar en consecuencia:

• Mantenimiento predictivo de maquinaria

• Identificación de clientes en riesgo de abandono

• Predicción de epidemias (dónde, cuándo, quién)

• Predicción de crímenes (dónde, cuándo, quién)

• Predicción de consumo energético

• Etc., etc., etc., etc…ar 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

• Para entender las soluciones propuestas, vamos a concentrarnos en la V de volumen (y su amiga inseparable la V de velocidad)

• Trabajar con mayores volúmenes de datos implica aumentar la capacidad de almacenamiento y procesamiento de los sistemas

• Dos tipos posibles de crecimiento (escalabilidad):

• Vertical (scale up)

• Horizontal (scale out)

Incrementar la potencia de la máquina en la que se ejecuta el software • Ya sea nuevos y mejores componentes o cambiar el ordenador completo



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

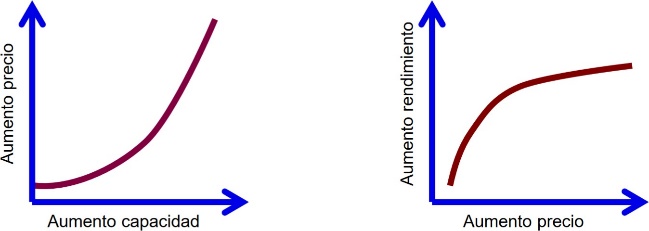
Tipos:

* 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).

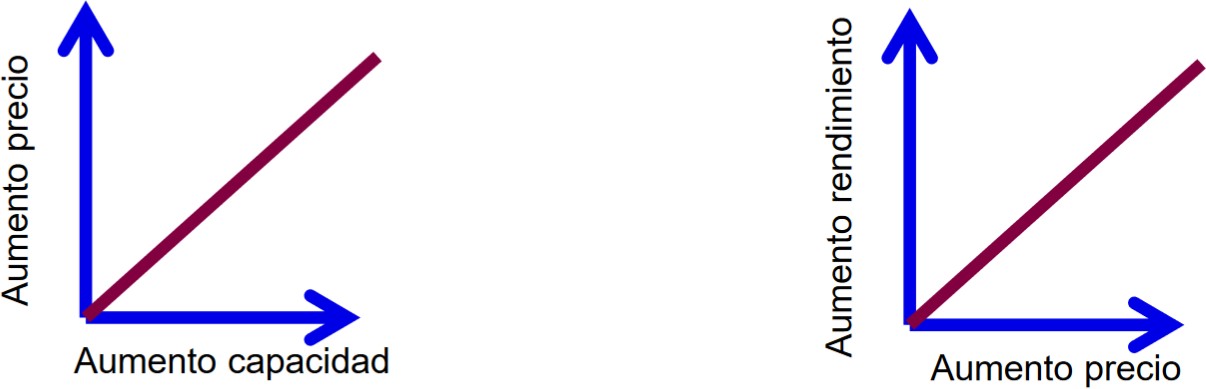
Ventajas:

Res

* + 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.

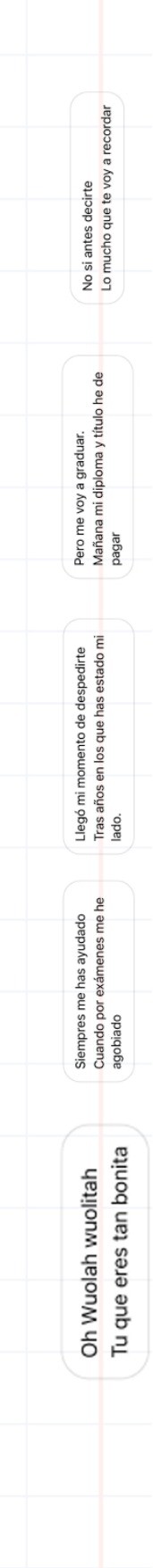
Tres propiedades deseables:

* 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.





**NUEVA REALIDAD, NUEVA PROBLEMÁTICA**

• Problema de caída de servidores cuando tengo tantos

• Las BBDD se pueden distribuir, pero a costa de sus propiedades ACID (atomicidad, consistencia, aislamiento y durabilidad)

## 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

Res

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).

¿Qué tiene de “mágico”?

• Hace fácil lo difícil (procesamiento paralelo)

• Ejemplo: clasificar y contar libros de una biblioteca

• Los mappers recorren las salas, clasifican los libros y los cuentan: generan la cuenta de libros de cada categoría que han encontrado

• Los reducers cogen las listas anteriores y cada uno suma los libros de una categoría. Al final tengo total de libros por cada categoría (y total de libros si sumo eso)

• Puedo reducir a la mitad el tiempo de los mappers si duplico su cantidad (cada uno tiene que hacer la mitad)

**EL ECOSISTEMA HADOOP**

• No es sólo Hadoop, sino una serie de sistemas de apoyo y complemento

• Sqoop: transferencia de ficheros entre HDFS y bases de datos

• Flume: transferencia de datos generados de forma continua (p.ej., logs) a ficheros HDFS • Oozie: permite definir y ejecutar flujos de trabajo sobre Hadoop

• Hive: Motor SQL sobre ficheros HDFS. Traduce de forma transparente al usuario consultas SQL a programas map-reduce. Esta traducción ralentiza el proceso

• Cludera Impala: funcionalidad equivalente a Hive pero puede llegar a ser 100 veces más rápido: algoritmos distribuidos propios (no map-reduce) que trabajan en memoria principal

• Pig: permite utilizar lenguaje más sencillo que MapReduce en Java (Pig Latin) para procesar datos

• Hue: interfaz gráfica para componentes principales de Hadoop

• Configurar y mantener actualizados todos los componentes Hadoop puede ser una tarea compleja

• Como alternativa, se ofrecen instalaciones de pago, con todo listo para ser instalado y usado: CDH de Cloudera, HDP de HortonWorks, MapR…

**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

**APACHE SPARK**

• Alternativa principal a Hadoop

• Versión 1.0 en junio de 2014

• Modelo de programación más sencillo y más potente (incluye pero no se limita a MapReduce)

• Conjunto de herramientas mejor integradas entre sí

• Rendimientos entre 10 y 100 veces mejor que el MapReduce de Hadoop

• Preferencia por procesamiento en memoria

**3.7 BASES DE DATOS NOSQL**

**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.
* Las BBDD relacionales imponen esquemas y estructuras rígidas que no siempre son adecuadas para un tratamiento rápido y flexible de la información.
* Cuando hablamos de BBDD NoSQL estamos hablando en realidad de BBDD no relacionales, que escapan del modelo habitual de filas y columnas para almacenar la información modelada en forma de entidades y relaciones.

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:



Reservados todos los derechos. No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

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:

* Se basan en el 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.

MONGO DB

• MongoDB (del inglés humongous, “enorme”) es un sistema de BBDD orientado a documentos. Es gratis y de código abierto. Muy utilizada en la industria.

• Permite trabajar con documentos de manera distribuida. ´

• Los documentos pueden tener estructuras diferentes (schemaless).

• Este aspecto, ventajoso, puede volverse un inconveniente (que en las BBDD Relacionales está mejor controlado por su rigidez). A veces son necesarias reglas explícitas.

• No existe validación ni integridad referencial. Deben ser implementadas.

• Otros aspectos a destacar:

• Características ACID comprometidas

• Más bien se habla de CAP (Consistencia, Disponibilidad y Tolerancia al particionado).

• Se garantiza la atomicidad sólo a nivel de documento.

• Permite el escalado horizontal de manera sencilla.

• Sistema de sharding para distribuir información en diferentes clusters.

• Buen rendimiento en lectura.

• Peor cuando el número de operaciones de escritura es muy alto.

• Funciona bien para grandes cantidades de datos almacenados.

• No es adecuado para transacciones complejas.

• Definiciones previas:

• Documento: unidad básica de almacenamiento. La información se guarda en formato BSON (Binary JSON). Se permiten documentos embebidos en otros.

• Colección: grupos de documentos.

• Base de Datos: contenedores físicos para almacenar colecciones.

Tabla

Descripción generada automáticamente con confianza media• Cluster: almacena varias bases de datos. Al tratarse de una BBDD distribuida, permite una gran escalabilidad horizontal.

• Comandos básicos CRUD del cliente MongoDB (https://docs.mongodb.com):

• Mostrar bases de datos:

• show dbs

• Creación y utilización de bases de datos:

• use nombre\_base\_de\_datos

• Borrado de bases de datos en uso:

• db.dropdatabase

• Mostrar colecciones:

• show collections

• Crear colección en una base de datos en uso:

• db.createCollection(nombre, opciones)

• Se crea automáticamente cuando se inserta un documento.

• Insertar un documento en una colección:

• db.nombre\_coleccion.insert(documento)

• db.nombre\_coleccion.insertOne(documento)

• db.nombre\_coleccion.insertMany(documentos)

• Eliminar colección:

• db.nombre\_coleccion.drop()

• Búsqueda:

• db.nombre\_coleccion.find()

• Búsqueda con salida formateada:

• db.nombre\_coleccion.find().pretty()

• Comandos básicos CRUD del cliente MongoDB (<https://docs.mongodb.com>):

• Actualizar datos de documentos:

• db.nombre\_coleccion.update(criterio\_seleccion, datos)

• Actualizar documento completo:

• db.nombre\_coleccion.save({\_id:ObjectId(), datos})

• Eliminar documentos de una colección:

• db.nombre\_coleccion.remove(criterio\_seleccion)

• Ordenar documentos en una colección:

• db.nombre\_coleccion.find().sort({KEY:i})

• Donde i puede ser 1 (ascendente) o -1 (descendente)

• Se pueden indicar multiples campos a indexar

• Creación de índices:

• db.nombre\_coleccion.ensureIndex({KEY:i})

• Donde i puede ser 1 (ascendente) o -1 (descendente)

• Agregación en la búsqueda:

• db.nombre\_colección.aggregate(operacion\_de\_agregacion)

• Tipos de datos más utilizados en MongoDB (<https://docs.mongodb.com>):

• String • Integer • Boolean • Double • Min/ Max • Arrays • Timestamp

• Object • Null • Symbol • Date • Object ID • Binary data • Code • Regular expression

Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente con confianza media

Algunos operadores: • $eq: igual que • $gt: mayor que • $gte: mayor o igual que • $in: en • $lt: menor que • $lte: menor o igual que • $ne: no igual • $nin: no en • $and: y • $or: o • $not: no • $nor: nor • $exists: existe • $type: tipo • $all: todos

Texto

Descripción generada automáticamente con confianza media

• Agregaciones

• Permiten consultas de datos más complejas.

• Se utiliza aggregate, y la estructura interna de composición es la de un “pipe”, donde los distintos filtros se van ejecutando secuencialmente:

• $project: proyección de campos, existentes y nuevos (creados ad-hoc para operar)

• $match: filtra documentos usando condiciones a través de campos existentes y nuevos

• $group: agrupa

• $sort: ordena.

• $lookup: permite realizar un join de documentos de distintas colecciones

• Algunas funciones de agregación

• $sum: suma • $avg: media • $min: mínimo • $max: máximo • $first: primero • $last: último

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Texto

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Características de las basadas en grafos: Neo4J

Si el nuevo paradigma son aplicaciones basadas en data lakes, hay oportunidad para BBDD especializadas

Las nuevas aplicaciones sociales dan origen a muchos datos en forma de red (networks) que son naturalmente representados como grafos

* 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.

• No se representa esquema, aunque muchas veces para optimización usamos etiquetas e índices ◊ ellos constituyen el esquema para Neo4J. ´

• Pero un nodo puede tener múltiples labels.

• Cypher incluye comandos DDL para manipular el esquema

Imagen que contiene Gráfico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente



# Texto Descripción generada automáticamente

Texto

Descripción generada automáticamente