

# Diseño de bases de datos relacionales

2023

Bravo Ainara, Bravo Iara



Laboratorio IV

## Contenido

Descripción de la arquitectura, las características y sus funcionalidades	3
Elementos principales de una base de datos relacional	4
Características de una base de datos relacional	5
Ventajas y desventajas de una base de datos relacional	6
Comparación contra los otros tipos de datos, cuándo conviene usarla y cuándo no	7
Bases de datos jerárquicas	9
Bases de datos transaccionales	9
Manejadores o gestores de bases de datos relacionales	10
Instalación del motor de la BD	12

## Descripción de la arquitectura, las características y sus funcionalidades

Una base de datos relacional es un tipo de base de datos que almacena y proporciona acceso a puntos de datos relacionados entre sí. Las bases de datos relacionales se basan en el modelo relacional, una forma intuitiva y directa de representar datos en tablas.

Una base de datos relacional es un conjunto de tablas (o relaciones) formadas por filas (registros) y columnas (campos); así, cada registro (cada fila) tiene una ID única, denominada clave y las columnas de la tabla contienen los atributos de los datos. Cada registro tiene normalmente un valor para cada atributo, lo que simplifica la creación de relaciones entre los puntos de datos.

Las bases de datos relacionales son útiles para cualquier necesidad de información en la que los puntos de datos se relacionan entre sí y también se deben administrar de forma coherente, segura y basada en reglas. Esto es lo que hace que las bases de datos relacionales sean más populares para negocios y empresas. Cuando las empresas quieren extraer conclusiones de sus propios datos, se basan en bases de datos relacionales para generar análisis útiles. Muchos de los informes que generan las empresas para realizar un seguimiento del inventario, las finanzas, las ventas o realizar proyecciones futuras se crean mediante bases de datos relacionales.

La metodología de diseño de bases de datos relacionales se ha consolidado a lo largo de los años satisfaciendo las propiedades de generalidad (independencia de la plataforma hardware/software), calidad del producto (precisión, completitud y eficacia) y facilidad de uso. Consta de las siguientes etapas:

### 1. Diseño conceptual

Su objetivo es definir las entidades y las relaciones entre ellos de manera abstracta, sin enfocarse en un modelo lógico en concreto. La herramienta a usar sería el modelo entidad - relación y el resultado que se obtiene es el diagrama entidad-relación.

### 2. Diseño lógico

Su objetivo es definir el esquema de la base de datos según el modelo que implementa el SGBD. La herramienta a usar sería el modelo lógico que implemente el sistema de gestión de bases de datos objetivo, pero es independiente de los aspectos físicos. Una de las técnicas que se emplea para verificar la calidad del esquema lógico es la normalización. El resultado de esta etapa es un esquema relacional basado en un modelo relacional.

### 3. Diseño físico

Su objetivo es generar el esquema físico de la base de datos en el modelo de datos que implementa el SGBD. Esto abarca la definición sobre el SGBD de las tablas con sus campos, la imposición de todas las restricciones de integridad y la definición de índices. La herramienta a utilizar sería el modelo físico de datos (organización de archivos e índices para el SGBD considerado), y el resultado que se consigue es el esquema físico de la base de datos.

## Elementos principales de una base de datos relacional

En una base de datos relacional, cada fila en una tabla es un registro con una ID única. Las columnas contienen los atributos de los datos y cada registro suele tener un valor para cada atributo.

### 1. Relaciones

La estructura de datos del modelo relacional es la relación. Definiremos a una relación como una tabla con columnas y filas. En el modelo relacional, las relaciones se utilizan para almacenar información sobre los objetos que se representan en la base de datos.

En un SGBD relacional hay dos tipos de relaciones:

- **Relaciones base.** Son aquellas que almacenan datos y su implementación es llamada "tabla".
- **Relaciones derivadas.** Son aquellas que no almacenan datos, pero son calculadas al aplicar operaciones, relaciones y su implementación es llamada "consulta" o "vista". Son convenientes por expresar información de varias relaciones actuando como si fuera una sola tabla. Algunas no son determinadas por los usuarios, sino que son inherentemente definidas por ser la BD relacional.

### 2. Restricciones

- Restricciones semánticas: permite a los usuarios incorporar restricciones personales a los datos
  - Clave primaria: Hace que los atributos marcados como clave primaria no puedan repetir valores.
  - Unicidad: Impide que los valores de los atributos marcados de esa forma, puedan repetirse.
  - Obligatoriedad: Prohíbe que el atributo marcado de esta forma no tenga ningún valor.
  - Integridad referencial: Prohíbe colocar valores en una clave externa que no estén reflejados en la tabla donde ese atributo es clave primaria.

- Regla de validación: Condición que debe de cumplir un dato concreto para que sea actualizado.
- Restricciones inherentes: no son determinadas por los usuarios, sino que son definidas por el hecho de que la base de datos sea relacional.
  - No puede haber dos tuplas iguales
  - El orden de las tuplas/atributos no importa
  - Cada atributo sólo puede tomar un valor en el dominio en el que está inscrito

### 3. Dominios

Una poderosa característica del modelo relacional son los dominios. Estos se definen como un conjunto de valores legales de uno o varios atributos. Mientras que un atributo es el nombre de una columna de una relación. Cada atributo de una base de datos relacional se define sobre un dominio, pudiendo haber varios atributos definidos sobre el mismo dominio. Por lo tanto, un dominio restringe los valores del atributo.

Existen distintos tipos de dominios como pueden ser enteros, cadenas de texto, fecha, no procedurales, etc.

### 4. Claves

- **Clave primaria o única.** Aquella clave candidata que se escoge para identificar sus tuplas de modo único. Ya que una relación no tiene tuplas duplicadas, siempre hay una clave candidata y, por lo tanto, la relación siempre tiene clave primaria.
- **Clave foránea.** Aquel atributo de una relación cuyos valores coinciden con los valores de la clave primaria de alguna otra relación (puede ser la misma). Las claves foráneas representan relaciones entre datos.
- **Clave índice.** Surgen a partir de la necesidad de tener un acceso más rápido a los datos. Generalmente no se consideran parte de la base de datos, pues se trata de un detalle agregado.

### 5. Procedimientos almacenados

Consiste en un código ejecutable que se asocia y almacena con la base de datos. Usualmente recogen y personalizan operaciones en común, como insertar un registro dentro de una tabla, recopilar información estadística, o encapsular cálculos complejos. No forman parte del modelo relacional.

## Características de una base de datos relacional

Las bases de datos relacionales tienen lo que se denomina un modo de coherencia o integridad basado en cuatro criterios: atomicidad, coherencia, aislamiento y durabilidad (ACID).

1. Atomicidad: Esta propiedad indica que, para que una transacción se dé por completada, deben haberse realizado todas sus partes o ninguna de ellas. Si una parte de la transacción falla, el sistema debe ser capaz de hacer que el resto de operaciones fallen, por lo que la base de datos no sufrirá ningún cambio indeseado.
2. Consistencia: se basa en la premisa que afirma que una transacción debe llevar al sistema de un estado válido a otro que también lo sea. Cabe resaltar que la validez de las operaciones está determinada por su seguimiento o no de las reglas establecidas para garantizar la fiabilidad de la base de datos.
3. Aislamiento: se refiere a la manera y el momento en el que los cambios resultantes de una operación se harán visibles para las demás operaciones concurrentes. Es decir, la realización de una operación no debería afectar a las otras, debido a que cada una de las transacciones debe ser ejecutada en aislamiento total, sin importar si se llevan a cabo de manera simultánea.
4. Durabilidad: hace referencia a la propiedad que garantiza que, una vez se haya llevado a cabo una determinada operación (aquellas transacciones que tuvieron un commit), estas tengan la capacidad de persistir y no puedan ser deshechas incluso si el sistema falla o se presentan eventos como errores o caídas o pérdida de alimentación eléctrica, entre otros.

## Ventajas y desventajas de una base de datos relacional

El modelo de datos de la base de datos relacional se implementa y gestiona más fácilmente que otros modelos. Las grandes cantidades de información que las empresas quieren almacenar a largo plazo se organizan sin problemas.

El modelo relacional fija una normativa que tiene el fin de evitar duplicaciones. Si las reglas se aplican, los sistemas relacionales facilitan un almacenamiento de datos libre de redundancias, puesto que solo es necesario editar los datos una única vez.

Las bases de datos relacionales normalizadas permiten almacenar datos sin contradicciones, contribuyendo así a la consistencia de los datos. Asimismo, los sistemas relacionales presentan funciones con las cuales se definen y se controlan automáticamente las condiciones de integridad. Aquellas transacciones que ponen en peligro la consistencia de los datos se bloquean.

El sistema de base de datos relacional se apoya en un procesamiento orientado a conjuntos que subdivide cada entidad en valores mínimos, permitiendo conectar entidades diferentes por medio del contenido.

Para la realización de consultas a bases de datos relacionales se ha consolidado el lenguaje SQL, que ha sido estandarizado por la ISO y la IEC. El propósito de tal estandarización es que las aplicaciones puedan desarrollarse y ejecutarse con independencia del SGBD en que se utilicen.

Por otro lado, no siempre es posible integrar cualquier tipo de dato en el formato fijo de las tablas aun cuando estén interconectadas. Los datos abstractos o no estructurados que surgen en relación con las aplicaciones multimedia y las soluciones de big data no pueden representarse en el modelo relacional.

Al mismo tiempo el principio de base de los sistemas relacionales que consiste en almacenar la información en tablas separadas conduce inevitablemente a su segmentación, debido a que este diseño deriva en complejas consultas que abarcan varias tablas, de modo que el elevado número de segmentos resultantes acostumbra a reflejarse negativamente en el rendimiento.

Por último el modelo relacional plantea elevados requisitos en cuanto a la consistencia de datos que van en detrimento de la velocidad de escritura en las transacciones.

## **Comparación contra los otros tipos de datos, cuándo conviene usarla y cuándo no**

### **Base de datos no relacional**

Si conocemos la información que necesitamos registrar, podremos diseñar la estructura de la base de datos relacional y las tablas que necesitaremos para poder relacionarlas, de manera sencilla y rápida para acceder a los datos que queramos consultar.

Una base de datos no relacional no tiene un identificador que sirva para relacionar un conjunto de datos con otros. Para realizar consultas se emplean lenguajes propios como JSON, CQL o GQL. Normalmente se usa cuando la

información se organiza mediante documentos o cuando no tenemos un esquema exacto de lo que vamos a almacenar.

Después de esta explicación, podemos decir que ambos tipos de bases de datos son útiles y dependerá del tipo de aplicación que deseamos realizar la elección de una u otra base de datos.

Así, si queremos desarrollar una aplicación de tipo contable, de inventario o de información de clientes, es probable que el modelo relacional se adapte mejor. En este tipo de aplicaciones, normalmente habrá más de una tabla que tenga relación con el resto, por lo que una base de datos relacional será más útil y podrá representar mejor nuestra aplicación.

Si por el contrario nuestra aplicación necesita de un sistema en el que los registros que vaya a almacenar no necesitan relacionarse con otros, usaremos una base de datos no relacional. Estas serán muy útiles para guardar grandes volúmenes de datos que sean de un solo tipo de esquema, como por ejemplo puede ser una base de datos en la que solo queramos almacenar libros o películas, ya que almacenaremos cada documento y no lo relacionaremos con ningún otro dato, solamente le daremos una serie de atributos que se podrán consultar pero que no tendrán relación alguna con otra tabla.

## **Base de datos orientada a objetos**

Las bases de datos relacionales se basan en la relación entre la información y diferentes factores. En la práctica, puedes verlo como una tabla, donde cada columna representa un tipo de información asociada con una fila. Una base de datos orientada a objetos se organiza en forma de diferentes objetos, que contienen archivos e información agrupados, así como procedimientos para leerlos y ejecutarlos

En una base de datos relacional, cada información individual se almacena como un archivo separado, independiente de los demás, y no hay soporte para el almacenamiento persistente de objetos con una estructura compleja. La atención se centra en datos más directos. Las bases de datos orientadas a objetos usan identificadores para etiquetar cada objeto, junto con técnicas de indexación para ubicar páginas en el disco. Como resultado, el sistema puede admitir objetos estructurados de forma más compleja.

Una base de datos orientada a objetos, como ya lo hemos mencionado, usa identificadores de objetos, que son como etiquetas. Al realizar una búsqueda de una de estas etiquetas, el sistema trae el conjunto de archivos e información que contiene. Y, dentro de ese objeto, puede haber otros tipos de jerarquías y relaciones internas.

El modelo relacional, a su vez, hace uso de dos claves relacionales. La primaria, como ya se mencionó, es la base, generalmente representada en la primera columna de la tabla, que identifica cada fila. Luego se usa una clave externa, que está relacionada con la primaria. Juntas sirven como una coordenada para localizar la información buscada.

Asimismo, pueden existir claves secundarias o terciarias, que actúan como filtros de búsqueda. De esta forma, es posible utilizar otras claves y separar aquellas que tienen información específica en un área determinada o aquellas que tienen un área vacía.

Una base de datos orientada a objetos, que tiene menos restricciones estructurales, puede contener mucha más información y en mayor variedad. Esto da como resultado un sistema de datos más flexible que se enfrenta a múltiples formatos de archivo, por ejemplo.

Una base de datos relacional, por otro lado, contiene un nivel más bajo de complejidad, lo que da como resultado un volumen de información relativamente menor.

Las bases de datos orientadas a objetos tienen una amplia variedad de permisos y restricciones según el sistema, por lo que es imposible generalizar. Por otro lado, las bases de datos relacionales tienen claves, integridad referencial e integridad de entidad impuestas como parte de su método.

En las bases de datos orientadas a objetos, la manipulación se incorpora directamente al lenguaje de programación utilizado, por ejemplo, C#.

Una base de datos relacional, en cambio, incluye su propio lenguaje para manipular su información, como SQL, QBE o QUEL. Esto reduce un poco la barrera para usar este sistema de manera más eficiente.

El primero es capaz de manejar solo un tipo de información a la vez, mientras que el segundo puede abarcar múltiples tipos de datos simultáneamente .

## **Bases de datos jerárquicas**

La principal diferencia entre las bases de datos jerárquicas y las bases de datos relacionales es que las bases de datos jerárquicas almacenan datos en forma de árbol con nodos parentales y secundarios, mientras que las bases de datos relacionales almacenan datos en tablas con filas y columnas como entidades y atributos. Una base de datos jerárquica contiene datos duplicados, pero las bases de datos relacionales no.

## Bases de datos transaccionales

Las diferencias más notorias son que para las bases de datos transaccionales la redundancia y duplicación de información no es un problema como para las bases de datos relacionales. Otro aspecto importante que podemos analizar es que las bases de datos relacionales se basan en la organización de la información en partes pequeñas que se integran mediante identificadores a diferencia de las bases de datos transaccionales, no tienen un identificador que sirva para relacionar dos o más conjuntos de datos. Además, son más robustas, es decir, tienen mayor capacidad de almacenamiento.

## Manejadores o gestores de bases de datos relacionales

Existe un tipo de software exclusivamente dedicado a tratar bases de datos relacionales, conocido como sistema de gestión de bases de datos relacionales, también llamados manejadores o gestores de las BDR.

### ORACLE

Es un sistema de gestión de base de datos relacional desarrollado por Oracle Corporation. Oracle surge en 1977 bajo el nombre SDL (Software Development Laboratories). Su creador, Larry Ellison, estuvo en el comité que definió SQL. Como software de base de datos, Oracle optimiza la gestión y seguridad de los conjuntos de datos creando esquemas estructurados a los que solo pueden acceder administradores autorizados. Su licenciamiento varía entre los \$57.312 dólares y \$79.892 dólares. Sus principales características son el soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma. Las principales empresas que usan Oracle son Netflix, LinkedIn, eBay, MIT, etc.

### PostgreSQL

Es un sistema de gestión de bases de datos de código abierto y de clase empresarial. Es decir que se puede utilizar, modificar e implementar según las necesidades de cada aplicación. La primera versión formal de Postgresl fue en 1997. Permite diseñar tablas relacionales y campos JSON para no relacionales. Almacena texto, imágenes, sonidos y video. Escala a tantos servidores de bases de datos como el usuario desee. Crea funciones personalizadas usando C,C++, Java o PL/SQL. Es compatible con Linux, Mac Os y Windows. PostgreSQL es más capaz de lidiar con circunstancias inusuales de bases de datos y procesamiento de datos de gran volumen. Las principales empresas que usan PostgreSQL son IKEA, BMV, UBER, Netflix o Instagram. Su licenciamiento es gratuito.

## **MariaDB**

Es un sistema de gestión de bases de datos de código abierto disponible en el mercado que actúa como interfaz SQL para que los datos puedan ser fácilmente accesibles. Su lanzamiento inicial fue en 2009. Grandes proyectos como Wikipedia y Google han migrado hacia ella. Se trata de un servicio de manejo de bases de datos que cuenta con licencia GPL. Creado por el mismo desarrollador de MySQL cuenta con un mayor rendimiento y funcionalidades que MySQL. Su licenciamiento varía en los \$500 dólares.

## **MySQL**

Es el sistema de gestión de bases de datos relacionales más usado en el mundo de código abierto y administrado por Oracle.. Tiene una sencilla curva de aprendizaje. El SGBD MySQL fue desarrollado en 1995. Posee doble licencia: Open Source y comercial con soporte. Compatible con Linux, Mac Os y Windows. Es adecuado tanto para proyectos grandes como pequeños. Utilizada por empresas como Facebook, Twitter, LinkedIn, Yahoo!, Amazon Web Services. Hay diferentes tipos de licenciamiento, gratuito o pago (entre \$2000 dólares a \$10000 dólares).

## **SQL Server**

Es un motor de base de datos relacional desarrollado por Microsoft que incluye mejoras al lenguaje SQL y soluciones para empresas. Su lanzamiento oficial fue en 1989. Funciona en Windows, Linux, Docker y Kubernetes. Integra herramientas de BI para no recurrir a soluciones de terceros. Tiene versiones gratuitas con todo lo necesario para proyectos y también versiones pagas que varían entre los \$230 dólares y \$15123 dólares. Puede ser instalado localmente o ser ejecutado en la nube de Azure. A diferencia de otros SGBD incluye un cliente gráfico para la administración. Facebook, Uber, Netflix, Amazon o Airbnb son algunos de los nombres de grandes empresas que utilizan a SQL Server como su gestor de bases de datos.

## **SQLite**

Es un motor de base de datos SQL transaccional de código abierto, ligero, autónomo, de configuración simple y sin servidor. Apareció en 2000 de la mano de su creador D. Richard Hipp. SQLite hace uso del lenguaje SQL y para el análisis de datos más complejos se puede combinar con scripts en Python. Compatible con Windows, Linux y Mac Os. Al estar integrado en todos los teléfonos se usa para almacenamiento interno de apps. Su código es de dominio público y gratuito. Almacena información persistente de forma sencilla. No admite un gran volumen de información. No obstante muchas empresas lo utilizan para sus aplicaciones de

escritorio como Adobe Photoshop Elements, Mozilla Firefox, Skype, Opera, The New Yorker, etc.

## Instalación del motor de la BD

- <https://youtu.be/FaP03GRAPV8>

# Bases de datos distribuidas

# ¿Qué es una base de datos distribuida?

Una base de datos distribuida es una base de datos que corre y almacena los datos en múltiples computadoras.

Estas bases de datos operan en dos o más servidores interconectados a una red.

A cada una de estas bases de datos se las conoce como instancias o nodos

# Características

- **Independiente de la ubicación.**
  - Los datos físicos almacenados en múltiples sitios y manejados por DBMS independientes
- **Procesamiento de queries distribuidos**
  - Los datos se distribuyen entre las distintas PCs lo que en algunos casos puede ser bastante demandante si no se está sobre una red de alta velocidad
- **Manejo de transacciones distribuidos**
  - Consistencia a través de protocolos de commit, técnicas de control de concurrencia distribuidas y métodos de recuperación distribuidos
- Se representan como una única base de datos lógica interconectadas
- Todas las bases de datos en una colección están conectadas por una red y se comunican entre ellas

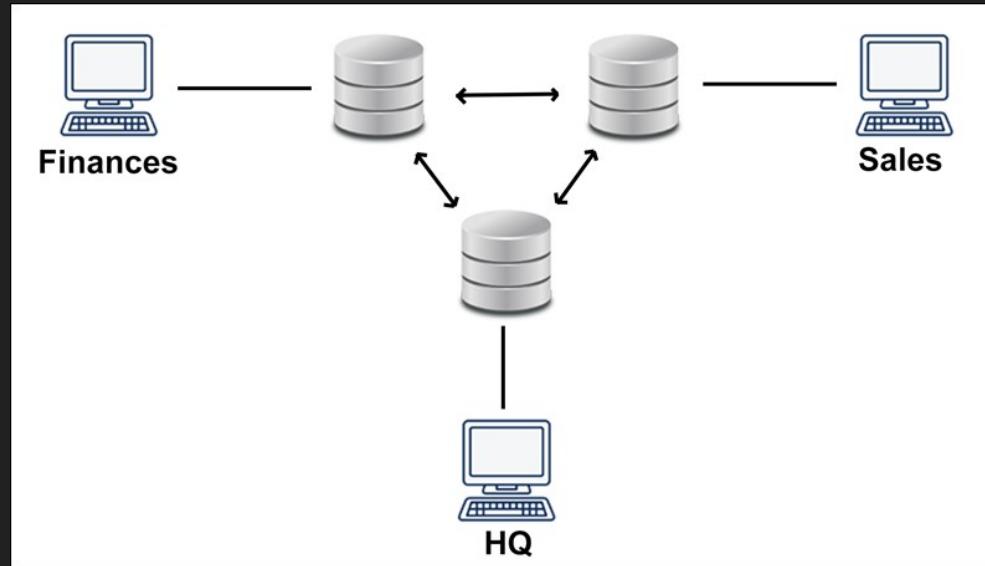
# Tipo - Homogénea

Todos los sitios usan un sistema operativo y SGBD idéntico y la estructura de datos es la misma en todos los sitios.

Sus propiedades son:

- Todos los sitios usan software similar
- Todos los sitios usan un SGBD idéntico o del mismo vendedor
- Cada sitio conoce del otro y cooperan para procesar los pedidos del usuario
- Se accede a través de una interface como si fuera una sola base de datos

Tipo - Homogénea



## Tipo - Homogénea

- **Autónoma**
  - Cada base de datos es autónoma que funciona por sí sola. Están integradas por una aplicación que maneja todo y utilizan mensajes para compartir los datos que son actualizados.
- **No-Autónoma**
  - Los datos son distribuidos a través de todos los nodos y un SGBD central o maestro coordina la actualización de datos a través de todos los sitios

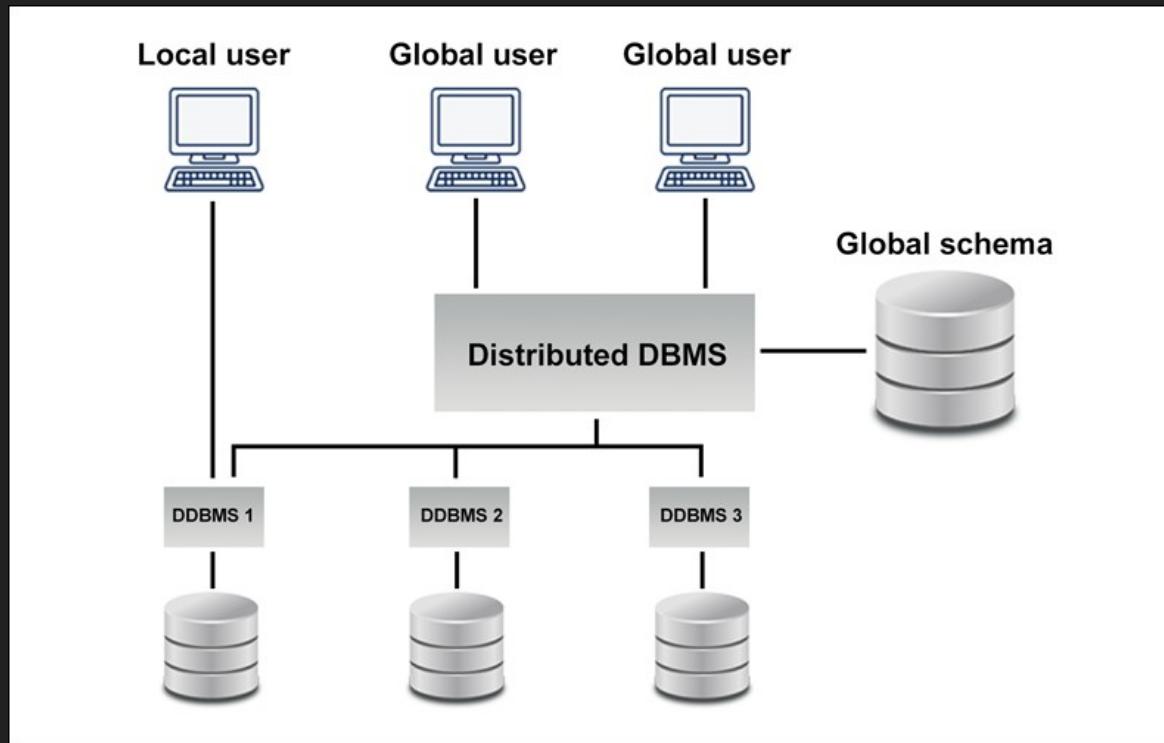
# Tipo - Heterogénea

Todos los sitios tiene diferentes sistemas operativos, SGDB y modelos de datos

Sus propiedades son:

- Todos los sitios usan un software y schema diferente
- Los sitios pueden utilizar distintos SGDB como relacionales, jerárquicos o orientados a objetos
- El procesamiento de queries es complejo debido a que los schemas son distintos
- El procesamiento de transacciones puede ser complejo debido a que los softwares son distintos
- Los sitios pueden no conocerse entre ellos por lo que la cooperación es limitada

# Tipo - Heterogénea



# Tipo - Heterogénea

- **Federated**
  - Los sistemas son naturalmente independientes e integrados para que funcionen como una sola base de datos
- **Un-Federated**
  - Los sistemas de bases de datos utilizan un módulo de coordinación centralizado a través del cual las bases de datos son accedidas.

# Replicación

La replicación me permite almacenar copias de datos en diferentes sitios. Esto me permite tener los datos disponibles en distintos sitios y realizar queries en paralelo.

Las desventaja de esto es que será necesario un constante actualizado y sincronizado de datos para mantener una copia exacta de la base de datos. Esto genera un complicado control de concurrencias.

Podemos tener **replicación completa** donde toda la base de datos se encuentra en todos los sitios o podemos tener **replicación parcial** en donde algunos fragmentos de datos son replicados y otros no

# Estrategias de replicación para datos

	Descripción	Utilidad	Desventaja
Replicación transaccional	Los usuarios reciben una copia inicial completa y van recibiendo actualizaciones a medida que los datos cambian. Funciona con un sistema publicador-suscriptor para garantizar la consistencia.	Se utiliza en entornos servidor-servidor.	Todos deben tener el mismo schema de datos. Constantemente se están enviando datos de un lado a otro.
Replicación Snapshot	Distribuye una copia exacta de la base de datos en un momento dado y no chequea por cambios en los datos.	Generalmente se usan cuando los cambios en los datos no son frecuentes.	Más lento que el anterior dado que tiene que enviar instantáneas de un punto a otro.
Replicación merge	Los datos de 1 o más bases de datos se combinan en una sola base de datos.	Es el tipo de replicación más complejo porque permite al publicador y al suscriptor realizar cambios en la base de manera independiente.	Se usa en entornos clientes-servidor

# Fragmentación

Tablas fragmentadas significa que las tablas están almacenadas en pequeños fragmentos y cada fragmento se almacena en sitios diferentes según sea requerido.

El requisito es que los fragmentos luego puedan ser reconstruidos en una tabla original sin perder datos.

# Fragmentación horizontal

Se divide una tabla horizontalmente asignando a cada fila o a un grupo de fila uno o más fragmentos. Estos fragmentos se asignan a distintos lados de una base de datos distribuida

Eno	Ename	Design	Salary	Dep
101	A	abc	3000	1
102	B	abc	4000	1
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

# Fragmentación horizontal

T1				
Eno	Ename	Design	Salary	Dep
101	A	abc	3000	1
102	B	abc	4000	1

T2				
Eno	Ename	Design	Salary	Dep
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

# Fragmentación vertical

Es el proceso de descomponer las columnas. Esta fragmentación es útil cuando algunos sitios no necesitan todas las columnas. Para recomponer la tabla, es necesario que se tenga la primary key. Dado que este proceso debe recomponerse con un JOIN natural, es necesario agregar una tupla-id para lograrlo

Eno	Ename	Design	Salary	Dep
101	A	abc	3000	1
102	B	abc	4000	1
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

# Fragmentación vertical

Eno	Ename	Design	Tuple_id
101	A	abc	1
102	B	abc	2
103	C	abc	3
104	D	abc	4
105	E	abc	5

# Fragmentación vertical

Salary	Dep	Tuple_id
3000	1	1
4000	1	2
5500	2	3
5000	2	4
2000	2	5

# Fragmentación Mixta

Se realiza la fragmentación en ambos sentidos.

- Se realiza primero una fragmentación horizontal y luego una fragmentación vertical
- Se realiza una fragmentación vertical y luego una fragmentación horizontal

Ename	Design
A	abc
B	abc
C	abc

# Ventajas y desventajas generales

- Ventajas
  - Incrementa la resiliencia y disminuye los riesgos dado que trabaja con réplicas de los mismos datos en múltiples instancias por lo que si se cae un nodo, la aplicación puede seguir trabajando.
  - Distribuir la base de datos puede mejorar la performance ya que todo el trabajo en varias bases de datos evita un cuello de botella por tener que realizar todo en la misma pc.
  - Distribuir la base de datos geográficamente permite reducir la latencia
- Desventajas
  - Incrementa la complejidad operacional
  - Incremento en la curva de aprendizaje

# Ventajas y desventajas de replicación

- Ventajas
  - Mejora del performance ya que los datos pueden leerse de una copia local en vez de una remota.
  - Incrementa la disponibilidad de datos ya que permite tenerlos en caso de que falle la base principal.
  - Mejora la escalabilidad ya que la carga en la base primaria se reduce mediante la lectura de copias.
- Desventajas
  - Incrementa la complejidad ya que el proceso de replicación necesita configurarse y mantenerse
  - Incrementa el riesgo de inconsistencias ya que los datos pueden ser actualizados en distintas réplicas
  - Incrementa el consumo de almacenamiento y uso de red ya que las copias deben ser almacenadas y transmitidas

# Ventajas y desventajas de fragmentación

- Ventajas
  - Aumenta la eficiencia si los datos están cerca del sitio de uso
  - Optimización de queries locales pueden ser suficiente para algunas queries ya que los datos están disponibles localmente
  - Facilita mantener la seguridad y la privacidad ya que los datos se encuentran fragmentados
- Desventajas
  - Velocidad de acceso puede ser muy alta si se necesitan datos de distintos fragmentos

# Comparación

No hay mucho para comparar dado que se está hablando de un tipo de arquitectura de base de datos

# Cuando usar

- Cuando queremos tener una buena disponibilidad (probabilidad de correr continuamente durante un intervalo de tiempo) y fiabilidad (probabilidad de correr en determinado tiempo)
- Cuando los tiempos de respuesta son importantes. Dado que podemos fragmentar los datos y los usuarios pueden realizar solicitudes donde los únicos datos que se necesiten sean los locales

# Bases de datos distribuidas gratis



# Bases de datos pagas - Amazon SimpleDB



## Data Transfer\*\*

Region: South America (Sao Paulo)

Data Transferred	Pricing
<b>Data Transfer IN</b>	
All data transfer in	\$0.00 per GB
<b>Data Transfer OUT ***</b>	
Next 9.99 TB / Month	\$0.15 per GB
Next 40 TB / Month	\$0.138 per GB
Next 100 TB / Month	\$0.126 per GB
Greater than 150 TB / Month	\$0.114 per GB

Data transfer "in" and "out" refers to transfer into and out of Amazon SimpleDB. There is no additional charge for data transferred between Amazon SimpleDB and other Amazon Web Services within the same Region (i.e., \$0.00 per GB). Data transferred across Regions (e.g., between Amazon SimpleDB in the EU (Ireland) Region and Amazon EC2 in the US East (Northern Virginia) Region), will be charged at Internet Data Transfer rates on both sides of the transfer.

## Free Tier\*

You can get started with Amazon SimpleDB for free. New and existing customers receive 25 SimpleDB Machine Hours and 1 GB of Storage for free each month. Many applications should be able to operate perpetually within these free tier limits.

## Machine Utilization

Region: South America (Sao Paulo)

- \$0.00 for first 25 hours
- \$0.19 per machine hour over 25 hours

Amazon SimpleDB measures the machine utilization of each request and charges based on the amount of machine capacity used to complete the particular request (SELECT, GET, PUT, etc.), normalized to the hourly capacity of a circa 2007 1.7 GHz Xeon processor. See below for a more detailed description of how machine utilization charges are calculated.

## Structured Data Storage

Region: South America (Sao Paulo)

- \$0.00 for first GB-month
- \$0.34 per GB-month thereafter

Amazon SimpleDB measures the size of your billable data by adding the raw byte size of the data you upload + 45 bytes of overhead for each item, attribute name and attribute-value pair.

Amazon SimpleDB is designed to store relatively small amounts of data and is optimized for fast data access and flexibility in how that data is expressed. In order to minimize your costs across AWS services, large objects or files should be stored in Amazon S3, while the pointers and the meta-data associated with those files can be stored in Amazon SimpleDB. This will allow you to quickly search for and access your files, while minimizing overall storage costs. [Click here](#) for a detailed explanation of how storage in Amazon SimpleDB and storage in Amazon S3 differ and a more detailed description on calculating your [Storage Costs](#).

# Base de datos pagas - CockroachDB

The image displays three service options for CockroachDB, each with a unique color scheme and features:

- CockroachDB-as-a-Service**
  - CockroachDB Serverless**
    - Fully-managed, autonomous CockroachDB cluster
    - Great for starter projects and evaluation
    - Never overpay and **Start Free!**
    - START INSTANTLY**
  - CockroachDB Dedicated**
    - Fully-managed, reserved CockroachDB cluster
    - Ideal deployment for a cloud database
    - Starting at **\$ .43/hro**
    - GET STARTED**
  - Self Hosted**
    - Self-managed CockroachDB clusters
    - Direct control and ability to run anywhere
    - CONTACT US**

# Links de instalación

## Instalación de Cassandra

<https://linuxhint.com/install-use-apache-cassandra-ubuntu-22-04/>

## Configuración para funcionar como base de datos distribuida

<https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/initialize/initSingleDS.html>

# Historia

Durante los años 60, en plena guerra fría, un investigador llamado Paul Baran desarrolló una idea de comunicación distribuida en donde los mensajes viajan a través de una red de nodos hasta que finalmente llegan a su destino.

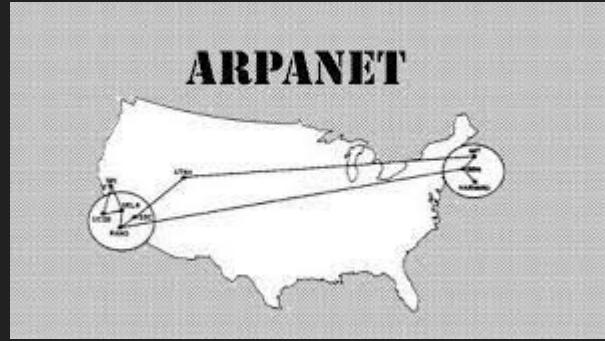
Estos nodos serán computadoras en vez de switches telefónicos que crearian múltiples caminos y fragmentarian el mensaje de tal manera que su transmisión fuera sencilla y eficiente.



# Historia

Con el objetivo de lograr la mayor eficiencia posible en las instalaciones, se creó un proyecto llamado ARPANET que tenía como objetivo conectar todas las instalaciones a una sola red.

A medida que la red fue creciendo, se empezaron a investigar diferentes tipos de redes. En conjunto con Europa, se empezaron a crear protocolos y sistemas de direccionamiento que permitirían la aparición de Internet



# Historia

Los primeros programas distribuidos se conocieron como Creeper y Reaper.

Creeper utilizaba los ciclos de procesador libres para copiarse al próximo sistema y desaparecer del anterior. Más adelante fue modificado para que solo se copiará.

Por este motivo nació Reaper que tenía como objetivo borrar todas las copias de Creeper.

En 1988, el DEC System Research Center comenzó un proyecto donde le enviaba a voluntarios, una aplicación que debían ejecutar en momentos libres y luego reenviar esos resultados. Este proyecto llegó a tener cerca de 100 usuarios en 1990.



## **TRABAJO PRÁCTICO GRUPAL**

## **BASES DE DATOS ORIENTADAS A OBJETOS**

Mateo Holzer  
Matías Meneghetti

13 de Abril de 2023  
Facultad de Ingeniería

## **Descripción de la arquitectura, las características y sus funcionalidades**

Una base de datos orientada a objetos (OODB, por sus siglas en inglés) es un tipo de sistema de gestión de bases de datos que utiliza el paradigma de la programación orientada a objetos (POO) para modelar y almacenar datos. Los datos se modelan como objetos. Los objetos a su vez se dividen en clases, creando así una jerarquía de clases y subclases, en la que las subclases heredan las propiedades las clases superiores y las complementan con sus propios atributos

El sistema de gestión de base de datos orientadas a objetos asigna de forma automática un código de identificación único a cada registro (que será inmutable), permitiendo así recuperar los objetos cuando estos se han guardado y realizar consultas.

La estructura de una base de datos orientada a objetos se basa en encapsular los datos y el código relacionado con cada objeto en una sola unidad. Las interacciones entre los objetos y el resto del sistema se realizan mediante una interfaz.

La interfaz de usuario es la forma en que los usuarios interactúan con la base de datos. Los usuarios pueden crear, modificar y eliminar objetos a través de una interfaz de programación de aplicaciones (API) o una interfaz gráfica de usuario (GUI).

La estructura de estas BDOO se diseña a partir de una serie de diagramas con los que se establecen las clases y sus relaciones, las interacciones entre los objetos y su comportamiento:

Así, a través de un diagrama de clases se presentan las clases con sus respectivas relaciones estructurales y de herencia, que se puede acompañar de un diagrama de objetos cuando no está muy claro cómo serán las instancias de las clases.

Se emplea un diagrama de secuencias para presentar las interacciones entre los objetos organizados en una secuencia temporal y describir cómo colaboran.

A continuación se presentan algunas de las características más importantes de las OODB:

1. Modelado de datos basado en objetos: En una OODB, los datos se modelan y almacenan en forma de objetos, que pueden tener atributos y métodos.

2. Herencia: Las OODB admiten la herencia de clases, lo que permite que las clases hereden atributos y métodos de clases superiores. Esto puede simplificar el diseño de la base de datos y reducir la cantidad de código necesario para implementar ciertas funcionalidades.
3. Capacidad de almacenar objetos complejos: Las OODB pueden almacenar objetos complejos, como imágenes, archivos de audio y video, y otros tipos de datos multimedia. En las bases de datos relacionales, estos datos a menudo se almacenan como referencias a archivos externos.
4. Polimorfismo: admiten polimorfismo, lo que significa que los objetos pueden tener diferentes formas o comportamientos, dependiendo del contexto en el que se usen.
5. Encapsulamiento: admiten el encapsulamiento, lo que significa que los datos y los métodos de un objeto están ocultos del mundo exterior. Esto puede mejorar la seguridad e integridad de los datos almacenados en la base de datos.
6. Triggers y eventos: admiten triggers y eventos, lo que significa que se pueden programar acciones que se desencadenan automáticamente cuando se produce un evento específico, como la inserción o eliminación de datos.
7. Concurrencia: pueden manejar múltiples usuarios que acceden y modifican la base de datos al mismo tiempo, lo que las hace ideales para aplicaciones de alto tráfico y sistemas de tiempo real.

## Ventajas y desventajas

### **Ventajas**

1. - Mayor capacidad de modelado:
2. Un objeto permite encapsular tanto un estado como un comportamiento.
3. Un objeto puede almacenar todas las relaciones que tenga con otros objetos. Los objetos pueden agruparse para formar objetos complejos (herencia).
- 4.
5. - Ampliabilidad: Se pueden construir nuevos tipos de datos a partir de los ya existentes
6. Agrupar propiedades comunes de diversas clases e incluirlas en una superclase, lo que reduce la redundancia.
- 7.
8. - Reusabilidad de clases, lo que repercute en una mayor facilidad de mantenimiento y un menor tiempo de desarrollo.
- 9.

- 10.- Lenguaje de consulta más expresivo.
11. El acceso navegacional desde un objeto al siguiente es la forma más común de acceso a datos en un SGBDOO. Mientras que SQL utiliza el acceso asociativo.
12. El acceso navegacional es más adecuado para gestionar operaciones como los despiece, consultas recursivas, etc.
- 13.
- 14.- Adecuación a las aplicaciones avanzadas de base de datos. Hay muchas áreas en las que los SGBD tradicionales no han tenido excesivo éxito como el CAD, CASE, OIS, sistemas multimedia, etc. en los que las capacidades de modelado de los SGBDOO han hecho que esos sistemas sí resulten efectivos para este tipo de aplicaciones.
- 15.
- 16.- Mayores prestaciones. Los SGBDOO proporcionan mejoras significativas de rendimiento con respecto a los SGBD relacionales.

## Desventajas

- Carencia de un modelo de datos universal. No hay ningún modelo de datos que esté universalmente aceptado para los SGBDOO y la mayoría de los modelos carecen de una base teórica.
- Carencia de experiencia. Todavía no se dispone del nivel de experiencia del que se dispone para los sistemas tradicionales.
- Carencia de estándares. Existe una carencia de estándares generales para los SGBDOO.
- Competencia. Con respecto a los SGBDR, estos productos tienen una experiencia de uso considerable. SQL es un estándar aprobado y ODBC es un estándar de facto. Además, el modelo relacional tiene una sólida base teórica y los productos relacionales disponen de muchas herramientas de soporte que sirven tanto para desarrolladores como para usuarios finales.
- La optimización de consultas compromete la encapsulación. La optimización de consultas requiere una compresión de la implementación de los objetos, para poder acceder a la base de datos de manera eficiente. Sin embargo, esto compromete el concepto de encapsulación.

- El modelo de objetos aún no tiene una teoría matemática coherente que le sirva de base.

## **Cuándo conviene usar la base de datos**

Conviene usar OODB cuando:

1. Datos complejos: Si la aplicación requiere el manejo de datos complejos como objetos, imágenes, sonidos y otros tipos de datos multimedia, entonces OODB puede ser una mejor opción que las bases de datos relacionales.
2. Manipulación directa de objetos de software: Si la aplicación está escrita en un lenguaje de programación orientado a objetos, entonces es posible que sea más fácil y natural manejar los objetos de software directamente usando una OODB.

No conviene usar OODB cuando:

1. Requerimientos simples: Si los requerimientos de la aplicación son simples y no requieren el manejo de datos complejos, una base de datos relacional puede ser más adecuada.
2. Experiencia en OODB: Si el equipo de desarrollo no tiene experiencia en el uso de OODB, puede resultar difícil y costoso aprender y aplicar correctamente las técnicas de OODB.
3. Herencia de clases: Si la aplicación no requiere la herencia de clases, una base de datos relacional puede ser una opción más adecuada, ya que no se está aprovechando una de las principales ventajas de OODB.

## **Comparación con base de datos relacional**

Las bases de datos orientadas a objetos (OODB) y las bases de datos relacionales son dos enfoques diferentes para almacenar y gestionar datos. A continuación se presentan algunas diferencias clave entre ambas:

1. Modelo de datos: Las bases de datos relacionales utilizan un modelo de datos basado en tablas, donde cada tabla tiene filas y columnas que representan los datos almacenados. Las OODB utilizan un modelo de datos basado en objetos, donde los datos se representan como objetos con atributos y métodos.
2. Escalabilidad y rendimiento: Las OODB pueden ofrecer una mayor escalabilidad y rendimiento que las bases de datos relacionales, especialmente en aplicaciones de alta demanda y tiempo real.

3. Consultas: Las bases de datos relacionales utilizan el lenguaje SQL para realizar consultas, mientras que las OODB utilizan lenguajes de programación orientados a objetos, como Java o C++. Esto puede hacer que las consultas en las OODB sean más naturales y fáciles de entender para los programadores.
4. Integridad de los datos: Las bases de datos relacionales tienen un sistema de integridad de datos incorporado que garantiza que los datos sean coherentes y precisos. Las OODB no tienen un sistema de integridad de datos tan completo, lo que puede requerir más trabajo por parte del desarrollador para garantizar la integridad de los datos.
5. Soporte para tipos de datos complejos: Las OODB pueden manejar tipos de datos complejos, como objetos geométricos, imágenes, sonidos y otros tipos de datos multimedia. Las bases de datos relacionales pueden manejar estos tipos de datos, pero a menudo requieren una estructura de tabla compleja o la utilización de varios tipos de datos diferentes.

## Comparación con base de datos NoSQL

Las bases de datos orientadas a objetos (OODB) y las bases de datos NoSQL son dos enfoques diferentes para almacenar y gestionar datos. A continuación se presentan algunas diferencias clave entre ambas:

1. Modelo de datos: Las bases de datos NoSQL, por otro lado, pueden tener varios modelos de datos, como documentos, grafos, clave-valor y columnas.
2. Escalabilidad: Las bases de datos NoSQL están diseñadas para escalar horizontalmente, lo que significa que se pueden agregar más nodos para manejar un mayor volumen de datos y tráfico. Las bases de datos orientadas a objetos no tienen esta capacidad de escalabilidad horizontal nativa.
3. Consultas: Las bases de datos NoSQL utilizan lenguajes específicos para el modelo de datos que se esté utilizando, como MongoDB Query Language para bases de datos de documentos.
4. Integridad de los datos: Las bases de datos orientadas a objetos pueden tener una integridad de los datos más sólida debido a la naturaleza estricta del modelo de programación orientado a objetos. Las bases de datos NoSQL pueden tener menos integridad de los datos, pero pueden compensarlo con la escalabilidad y la flexibilidad.
5. Flexibilidad: Las bases de datos NoSQL son más flexibles que las OODB, ya que pueden manejar una amplia variedad de modelos de datos y pueden adaptarse a las necesidades de la aplicación. Las OODB, por otro lado, están diseñadas específicamente para el modelo de programación orientado a objetos y son menos flexibles en cuanto a la estructura de los datos.

## Comparación con base de datos embebidas

1. Modelo de datos: Las bases de datos embebidas suelen tener un modelo de datos más simple, como tablas y columnas en una estructura de base de datos relacional.
2. Escalabilidad: Las bases de datos embebidas son limitadas en escalabilidad debido a su naturaleza embebida. Están diseñadas para aplicaciones que no requieren grandes cantidades de datos y no necesitan un alto nivel de escalabilidad.
3. Consultas: Las bases de datos embebidas son más adecuadas para aplicaciones que tienen requisitos de consulta más simples y predecibles. Debido a su modelo de datos más simple, estas bases de datos pueden manejar mejor las consultas simples y directas que se realizan en aplicaciones de baja complejidad.
4. Distribución de datos: Las bases de datos embebidas suelen ser monolíticas, lo que significa que todos los datos se almacenan en una sola ubicación física, generalmente dentro de la aplicación que la utiliza. En las OODB los datos se almacenan en múltiples servidores en diferentes ubicaciones geográficas.

## Gestores y motores

- GemStone/S  
Como la mayoría de otras bases, tiene su versión gratuita, por cierta cantidad de días, y luego cuenta con versiones pagas y con sus respectivos beneficios
- ObjectDB  
Base de datos para Java, se puede utilizar de forma gratuita con la prueba de 30 días, y luego tiene dos versiones pagas, una para el uso más personal, con un valor de 500 euros, y una versión más utilizada para empresas, con un valor de 2500 euros
- DB4O  
Es una BDOO para Java y .NET, se puede utilizar, modificar y distribuir libremente, siempre y cuando se respeten los términos de la licencia.
- Velocity DB  
Al igual que objectDB, se puede utilizar la prueba gratuita por 30 días, y luego cuenta con 3 planes distintos, el más básico tiene un valor de 400 dólares por mes, el intermedio 600 y el más completo, 3000
- ZODB  
Base de datos orientada a objetos para Python, con licencia de software libre.

## **Historia y antigüedad**

A inicios de la década de los años 60 se gesta el origen de este tipo de bases de datos. Ocurre en Noruega, donde el doctor Nygaard, especialista en la elaboración de sistemas informáticos que llevaban a cabo simulaciones de sistemas de naturaleza física, puso a punto esta nueva tendencia. Acostumbrado a buscar la representación informática del rendimiento de elementos físicos como el motor de un coche, Nygaard tenía ciertas necesidades que hasta ese momento nadie podía satisfacer. En su época, todavía los sesenta, había ciertos obstáculos que impedían que su trabajo pudiera llegar a buen puerto. Se encontraba con que el software del momento era excesivamente complicado y debía recurrir a modificaciones en cada una de sus operaciones. El proceso de trabajo superaba los límites de la comodidad, dado que por cada iniciativa que llevaba a cabo tenía que contemplar varias alteraciones y posibilidades que cubrían las alternativas a sus planteamientos. Todo ello hacía que el trabajo fuera demasiado absorbente y que se saliera de los límites de lo tolerable.

Esta situación hizo que el doctor Nygaard y su equipo planteasen alternativas. Así es como trabajaron en la elaboración de un software que se diseñase de forma paralela al objeto físico en cuestión. Esto suponía cambiar la forma de trabajar, pero con el objetivo de alcanzar un resultado mucho más adecuado y beneficioso. El equipo del doctor planteó un requisito clave: que el programa fuera un reflejo del objeto. Si el objeto físico en cuestión estaba formado por 50 componentes principales, el software también debería tener 50 módulos. Esto garantizaría que por cada pieza del objeto habría una contraposición en el software, todo ello visible y analizable. Y si el objeto en cuestión funcionaba bien gracias a la comunicación de sus 50 piezas, el software también lo haría a través de la comunicación de los 50 módulos y de sus componentes.

¿Qué ganó Nygaard? Se encontró con que su solución le resolvía todos los problemas. Disponía de un mantenimiento absolutamente dominado, sin sorpresas, pero al mismo tiempo también veía que ahora sus programas se podían dividir de forma lógica. De esta forma, con los posteriores análisis y pruebas el doctor podría cambiar las piezas de forma independiente y cómoda, así como hacerlo con facilidad. Así es como comenzó a realizar pruebas más fiables, específicas y de resultados precisos.

Y entonces se descubrió su potencial. Uno de los principales problemas a los que se enfrentaba Nygaard en su época se encontraba en que debía realizar procesos de trabajo largos para una tarea posterior que no simbolizaba tanta dificultad. No era tirar horas de trabajo a la basura, pero sí sentía que su tiempo se estaba desperdiando dentro del contexto. Por eso planteó soluciones y descubrió que finalmente la orientación a objetos con la que había trabajado le resolvía este aspecto. El motivo se encontró en que pudo introducir la reusabilidad en la creación

del software. A partir de este momento cada programa no representaba un inicio y final para el software, sino que en el proceso de creación muchos elementos podían recuperarse y mantenerse en activo para futuras creaciones.

Pero con este descubrimiento Nygaard necesitaba un lenguaje que le sirviera de apoyo. Así fue como nació Simula-67, lenguaje que hoy día aún se usa y que está considerado como uno de los inventos más importantes en el mundo de la informática aunque no sea algo conocido en términos populares.

Con el paso del tiempo la industria continuó el trabajo en las bases de datos orientada a objetos. Una década después el equipo de Alan Kay y Xerox tomó de referencia el trabajo de Simula-67 para crear otro lenguaje similar: Smalltalk. En los años 80 este lenguaje fue utilizado en combinación con algunas de las ideas que habían sido más características de Simula-67 para dar forma a otro lenguaje que tiene gran importancia hoy día: C++, que se ocupó de que este tipo de trabajo alcanzará sus máximos niveles de rendimiento.

### **Principales empresas donde funciona:**

- IBM
- Microsoft
- Oracle
- Informix
- SAP
- Progress software

### **Proveedores que facilitan el producto:**

- Objectivity
- InterSystems
- Versant
- Starcounter
- ObjectBox

# Trabajo Práctico

Materia: Laboratorio IV

Tema: Bases de datos distribuidas

Profesora: Marcela Russo

Alumno:

- Nicolás Ezequiel Salvia

Año: 2023

# Base de datos distribuidas

Descripción de la arquitectura, las características y sus funcionalidades.

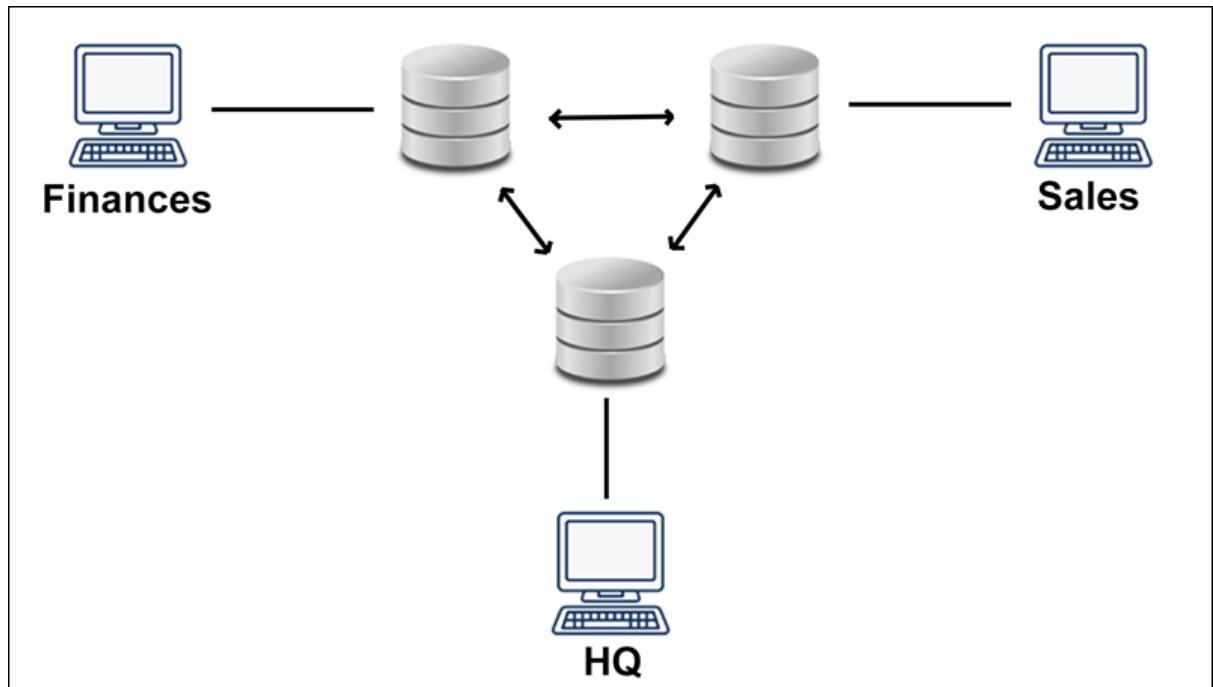
Una base de datos distribuida es una base de datos que corre y almacena los datos en múltiples computadoras. Estas bases de datos operan en dos o más servidores interconectados a una red. A cada una de estas bases de datos se las conoce como instancias o nodos.

## Características

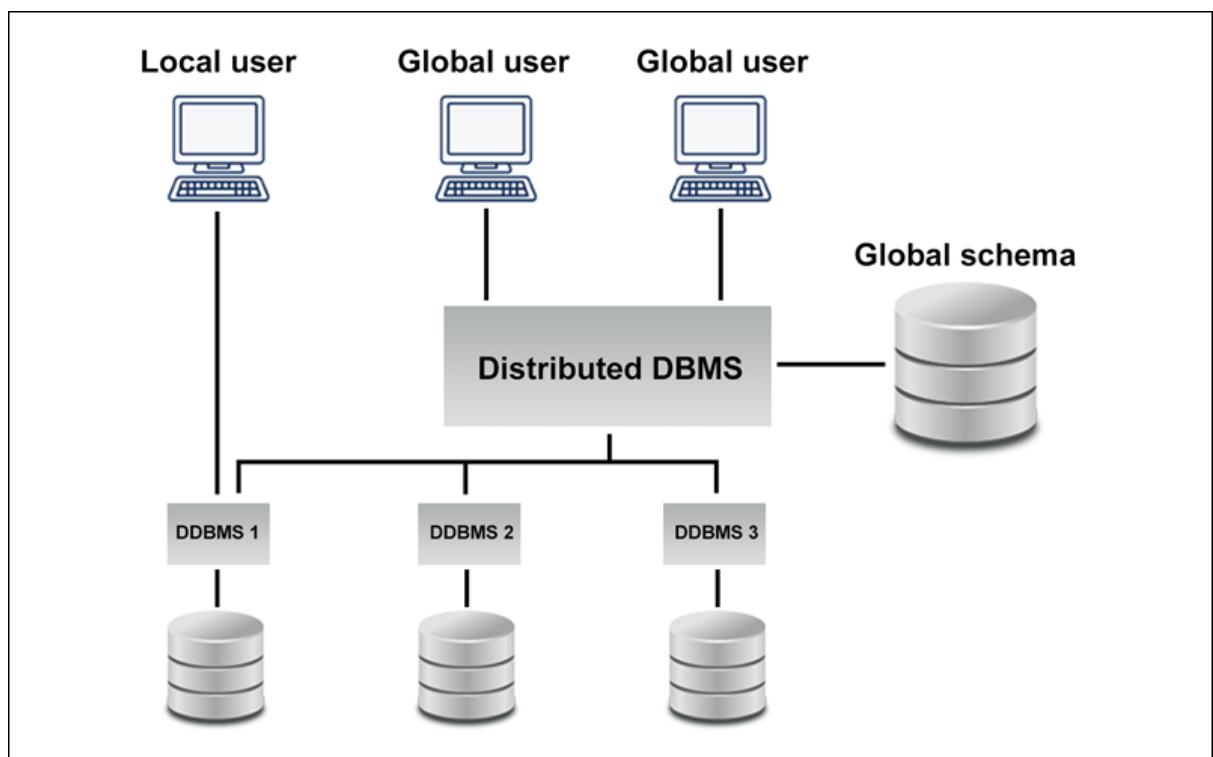
- **Independiente de la ubicación.** Datos físicos almacenados en múltiples sitios y manejados por DDBMS independientes
- **Procesamientos de queries distribuidos.**
- **Manejo de transacciones distribuidos.** Provee consistencia a través de protocolos de commit, técnicas de control de concurrencia distribuidas y métodos de recuperación distribuidos
- Las bases de datos se representan como una única base de datos lógica interconectadas
- Todas las bases de datos en una colección están linkeadas por una red y se comunican entre ellas.
- **Transaction processing.** Programa que incluye una colección de operaciones de una o más base de datos. Consiste en un proceso atómico donde se ejecuta todo o nada.

## Tipos

- **Homogénea:** Red de idénticas bases de datos en múltiples sitios. Todas tienen el mismo sistema operativo, DDBMS y estructura de datos haciéndola más fácil de manejar.



- **Heterogénea:** Usan diferentes schemas, sistema operativo, DDBMS y diferentes modelos de datos.



## Almacenado en base de datos distribuidos

### Replicación

El sistema almacena copias de datos en diferentes sitios. La ventaja es que aumenta la disponibilidad de datos en diferentes sitios y permite realizar queries en paralelo. Sin embargo, esto implica que están constantemente actualizando y sincronizando las unas con otras para mantener una copia exacta de la base de datos. Esto genera un complicado control de concurrencias.

Podemos tener **replicación completa** en donde toda la base de datos se encuentra en todos los sitios o podemos tener **replicación parcial** en donde algunos fragmentos de datos son replicados y otros no.

Tipos de replicación para datos			
	Descripción	Utilidad	Desventaja
<b>Replicación transaccional</b>	Los usuarios reciben una copia inicial completa y van recibiendo actualizaciones a medida que los datos cambian. Funciona con un sistema publicador-suscriptor para garantizar la consistencia.	Se utiliza en entornos servidor-servidor.	
<b>Replicación Snapshot</b>	Distribuye una copia exacta de la base de datos en un momento dado y no chequea por cambios en los datos.	Generalmente se usan cuando los cambios en los datos no son frecuentes.	Más lento que el anterior dado que tiene que enviar instantáneas de un punto a otro.
<b>Replicación merge</b>	Los datos de 1 o más bases de datos se combinan en una sola base de datos.	Es el tipo de replicación más complejo porque permite al publicador y al suscriptor realizar cambios en la base de manera independiente.	Se usa en entornos clientes-servidor

## Fragmentación

Las tablas están fragmentadas. Esto quiere decir que están separados en pequeñas partes y cada fragmento se almacena en sitios diferentes según sea requerido. El requisito es que los fragmentos luego puedan ser reconstruidos en una relación original sin perder datos. La ventaja que tienen es que previene la inconsistencia de datos. Existen dos tipos de fragmentaciones:

### Horizontal

Es el proceso de dividir una tabla horizontalmente asignando a cada fila o a un grupo de filas de relaciones, uno o más fragmentos. Estos fragmentos se asignan a distintos lados de una base de datos distribuida.

Ejemplo:

Consideremos una tabla empleado

<b>Eno</b>	<b>Ename</b>	<b>Design</b>	<b>Salary</b>	<b>Dep</b>
101	A	abc	3000	1
102	B	abc	4000	1
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

Podemos dividir la tabla de la siguiente manera:

<b>T1</b>				
<b>Eno</b>	<b>Ename</b>	<b>Design</b>	<b>Salary</b>	<b>Dep</b>
101	A	abc	3000	1
102	B	abc	4000	1

<b>T2</b>				
<b>Eno</b>	<b>Ename</b>	<b>Design</b>	<b>Salary</b>	<b>Dep</b>
103	C	abc	5500	2
104	D	abc	5000	2
105	E	abc	2000	2

Donde para reconstruirla hacemos  $T = T1 \cup T2$

Vertical

Es el proceso de descomponer las columnas por columnas. Esta fragmentación es útil cuando algunos sitios no necesitan todas las columnas. Para recomponer la tabla, es necesario que se tenga la primary key. Este proceso debe poder lograrse mediante un JOIN natural por lo que es necesario agregar una tupla-id para lograrlo.

Ejemplo:

Utilizando la misma tabla del ejemplo anterior, podemos dividir la tabla en

<b>Eno</b>	<b>Ename</b>	<b>Design</b>	<b>Tuple_id</b>
101	A	abc	1
102	B	abc	2
103	C	abc	3
104	D	abc	4
105	E	abc	5

<b>Salary</b>	<b>Dep</b>	<b>Tuple_id</b>
3000	1	1
4000	2	2
5500	3	3
5000	1	4
2000	4	5

## Fragmentación Mixta

Se realiza la fragmentación en ambos sentidos. Esto se puede realizar de dos maneras

1. Se realiza una fragmentación horizontal y luego una fragmentación vertical.
2. Se realiza mediante una fragmentación vertical y luego una fragmentación horizontal.

Ejemplo:

Ename	Design
A	abc
B	abc
C	abc

## Configuraciones de replicación

- **Maestro-Esclavo:** Una base de datos es configurada como maestro y las otras como esclavos. El maestro recibe todas las operaciones de escritura y los esclavos una copia de los datos
- **Replicación Multi-Maestro:** Todos los servidores son maestros, en todos se les escribe y todos los cambios en un maestro serán replicados en los otros maestros.
- **Replicación peer-to-peer:** Todos los servidores actúan como maestro-esclavo y los datos son replicados de manera peer-to-peer
- **Replicación de fuente única:** Una sola fuente de base de datos es replicada en múltiples bases de datos a las que se les apunta.
- **Multiactiva:** Utilizado por CockroachDB. Se diferencia del sistema **activo-activo** ya que este solo aplica la escritura cuando se logra un consenso entre las replicas, es decir, la mayoría de las réplicas pudieron escribir el cambio exitosamente. Si la mayoría de las replicas están offline, la base de datos no estará disponible.

## Ventajas y desventajas.

Ventajas y desventajas generales	
Ventajas	Desventajas
<b>Incrementa la resiliencia y disminuye los riesgos.</b> Dado que trabajan con réplicas de los mismos datos en múltiples instancias, si uno de los nodos cae, la aplicación puede seguir trabajando.	<b>Incremento de la complejidad operacional.</b>
<b>Distribuir la base de datos puede mejorar la performance.</b> Permite distribuir todo el trabajo en varias bases de datos evitando un cuello de botella por tener que realizar todas las operaciones de lectura y escritura en la misma pc.	<b>Incremento en la curva de aprendizaje</b>
<b>Distribuir la base de datos geográficamente permite reducir la latencia.</b>	

Ventajas y desventajas de replicación	
Ventajas	Desventajas
Mejora del performance ya que los datos pueden leerse de una copia local en vez de una remota	Incrementa la complejidad ya que el proceso de replicación necesita configurarse y mantenerse
Incrementa la disponibilidad de datos ya que permite tenerla en caso de que falle la principal	Incrementa el riesgo de inconsistencias ya que los datos pueden ser actualizados en distintas replicas
Mejora la escalabilidad ya que la carga en la base primaria se reduce mediante la lectura de copias	Incrementa el consumo de almacenamiento y uso de red ya que las copias deben ser almacenadas y transmitidas

Ventajas y desventajas de fragmentación	
Ventajas	Desventajas
Aumenta la eficiencia si los datos están cerca del sitio de uso	Velocidad de acceso puede ser muy alta si se necesitan datos de distintos fragmentos
Optimización de queries locales puede ser suficiente para algunas queries ya que los datos están disponibles localmente	Puede ser muy costoso utilizar fragmentación recursiva
Facilita mantener la seguridad y la privacidad ya que los datos se encuentran fragmentados	

Comparación contra los otros tipos de datos, cuándo conviene usarla y cuándo no.

No hay mucha para comparar dado que estamos hablando de un tipo de arquitectura de base de datos más que de un motor de base de datos.

Cuando usar una base de datos distribuidas:

- Cuando queremos tener una buena disponibilidad (probabilidad de correr continuamente durante un intervalo de tiempo) y fiabilidad (probabilidad de correr en determinado tiempo)
- Cuando los tiempos de respuesta son importantes. Con esta base de datos podemos fragmentar los datos y los usuarios pueden realizar solicitudes donde los únicos datos que se necesiten sean los locales.

Cuando no usar una base de datos distribuidas:

- Cuando no se cumple ninguna de las anteriores

Modo de licenciamiento, de ser posible comentar acerca de los precios.

• <b>Apache Ignite</b>	Apache Software Foundation
• <b>Apache Cassandra</b>	Apache Software Foundation
• <b>Apache HBase</b>	Apache Software Foundation
• <b>Couchbase Server</b>	Couchbase Inc.
• <b>Amazon SimpleDB</b>	Amazon
• <b>FoundationDB</b>	Apple
• <b>CockroachDB</b>	Cockroach Labs

Mostrar la instalación del motor de la BD siempre que sea posible.

<https://linuxhint.com/install-use-apache-cassandra-ubuntu-22-04/>

<https://docs.datastax.com/en/cassandra-oss/3.0/cassandra/initialize/initSingleDS.html>

Años de antigüedad de su creación, principales empresas en donde funciona, proveedor que facilita el producto.

Durante los años 60, en plena guerra fría, un investigador llamado Paul Baran desarrolló una idea de comunicación distribuida en donde los mensajes viajaban a través de una red de nodo hasta que finalmente llegaran a su destino. Estos nodos serían computadoras en vez de switches telefónicos. Estos nodos crearían múltiples caminos y fragmentarían el mensaje de tal manera que su transmisión fuera sencilla y eficiente.

Con el objetivo de lograr la mayor eficiencia posible de las instalaciones, se creó un proyecto llamado ARPANET que tenía como objetivo conectarlas todas a una red.

A medida que la red fue creciendo se empezaron a investigar diferentes tipos de redes. En conjunto con Europa, se empezó a crear protocolos y sistemas de direccionamiento entre otras cosas que permitieron la aparición de Internet.

Los primeros programas distribuidos se conocieron como Creeper and Reaper. Creeper utilizaba los ciclos de procesador que no se utilizaban para copiarse al próximo sistema y borrarlo del anterior. Creeper fue modificado para que no se borrará del sistema anterior y así surgió Reaper que se encargaba de borrar todas las copias de Creeper.

En 1988, el DEC(Digital Equipment Corporation) System Research Center comenzó un proyecto donde le enviaba a los voluntarios una aplicación que debían ejecutar el momentos libres y luego reenviar esos resultados. Este proyecto llegó a tener cerca de 100 usuarios en 1990.

Más adelante, el DEC y otros grupos comenzaron a utilizar Internet con el objetivo de distribuir material para ser calculado. Eran similares a rompecabezas que consiste en factorizar números, buscar números primos y crackeo de encriptaciones. Estos rompecabezas tenían incentivos monetarios ya que requieren de mucho tiempo para poder resolver estos problemas.

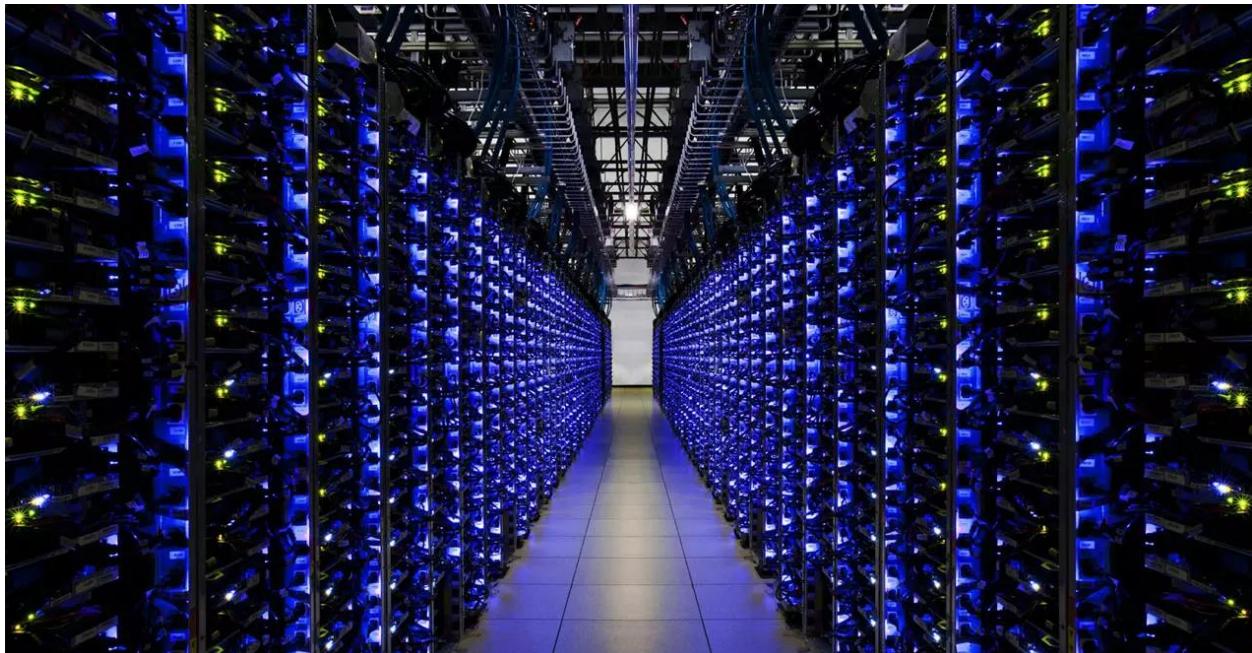
Fuentes:

- <https://www.cockroachlabs.com/blog/what-is-a-distributed-database/>
- <https://phoenixnap.com/kb/distributed-database>
- <https://www.geeksforgeeks.org/fragmentation-in-distributed-dbms/>
- <https://www.geeksforgeeks.org/data-replication-in-dbms/>
- <https://www.techtarget.com/searchoracle/definition/distributed-database>

# Optimización de Bases de Datos

Por Sebastian Machado y Daira Orlandini

---



---

La optimización de bases de datos es esencial para garantizar un rendimiento eficiente en la gestión de datos. En este documento se explorarán conceptos, técnicas y mejores prácticas para lograr un rendimiento óptimo en bases de datos y maximizar su potencial.

## ¿Qué es la optimización?

La optimización de bases de datos se refiere a la mejora del rendimiento de una base de datos, con el objetivo de aumentar la velocidad de las consultas, conseguir la mayor eficiencia posible en el almacenamiento de datos y conseguir la menor carga del sistema. En general, se busca que la base de datos tenga la capacidad de manejar grandes volúmenes de datos y usuarios concurrentes (aquellos que tienen la posibilidad de actuar en simultáneo con otros).

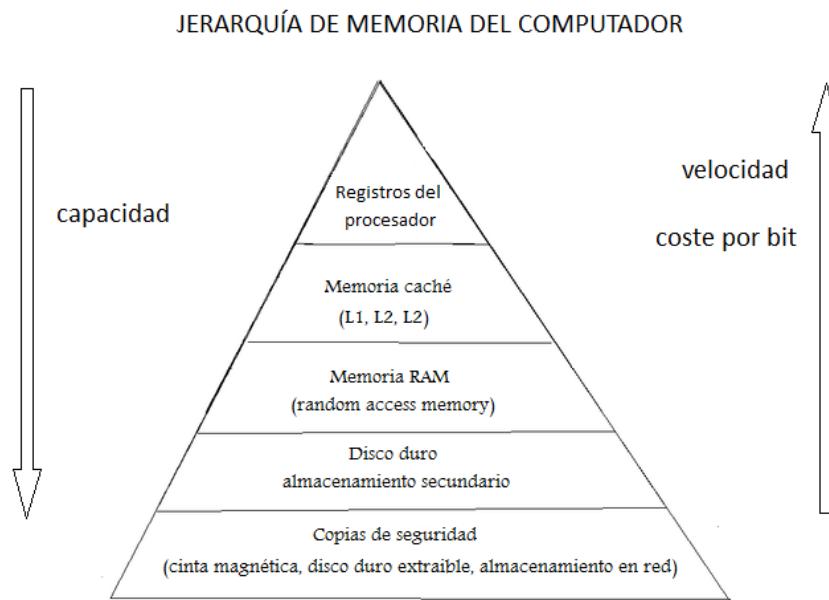
## ¿Cómo optimizar una base de datos?

### 1. El diseño de la base

Se deben tomar las decisiones correctas en cuanto a una serie de aspectos según el proyecto que se vaya a encarar. Esto implica, el tipo de base de datos, las tecnologías a utilizar, el diseño



propriamente dicho de la base, las restricciones y el hardware que dará soporte, entre otras. Recordemos que hay memorias más efectivas que otras.



Dependiendo del proyecto, podría elegir distintos tipos de base de datos.

Algunos ejemplos son:

- Una base de datos relacional para un proyecto con una estructura de datos bien definida y estable, como un sistema de gestión o de inventario.
- Una base de datos no relacional para un proyecto con grandes cantidades de datos no estructurados y variables, como redes sociales o sistemas de sensores.
- Una base de datos orientada a objetos para una aplicación en la cual no solo necesite guardar datos, sino también métodos.

## 2. Gestión de Índices

Un índice SQL se utiliza para recuperar datos de una base de datos rápidamente. Indexar una tabla o vista es, sin duda, una de las mejores formas de mejorar el rendimiento de consultas y aplicaciones.

Es una **tabla de búsqueda rápida** para encontrar registros que los usuarios necesitan buscar con frecuencia. Un índice es pequeño, rápido y está optimizado para búsquedas rápidas. Es muy útil para conectar las tablas relacionales y buscar tablas grandes.

Los índices SQL son principalmente una herramienta de rendimiento, por lo que realmente se aplican si una base de datos crece. SQL Server admite varios tipos de índices, pero uno de los tipos más comunes es el índice agrupado. Este tipo de índice se crea automáticamente con una **clave principal**.

Para aclarar el punto, el siguiente ejemplo crea una tabla que tiene una clave principal en la columna "EmployeeId":

```
CREATE TABLE dbo.EmployeePhoto  
  
    (EmployeeId      INT NOT NULL PRIMARY KEY,  
  
     Photo          VARBINARY(MAX) NULL,  
  
     MyRowGuidColumn UNIQUEIDENTIFIER NOT NULL  
                           ROWGUIDCOL UNIQUE  
                           DEFAULT NEWID()) ;
```

En la tabla "EmployeePhoto", la clave principal al final de la definición de columna "EmployeeId". Esto crea un índice SQL que está especialmente optimizado para usarse mucho. Cuando se ejecuta la consulta, SQL Server creará automáticamente un índice agrupado en la columna especificada y podemos verificar esto desde el **Explorador de objetos** si navegamos a la tabla recién creada y luego a la carpeta **Índices**:

The screenshot shows the Object Explorer on the left with 'dbo.EmployeePhoto' selected, displaying its columns (EmployeeId, Photo, MyRowGuidColumn), keys (Clustered Primary Key), constraints (UQ\_Employee\_1842D5A9392045CF), triggers, and indexes. The SQL Query window on the right contains the following code:

```

CREATE TABLE dbo.EmployeePhoto
(
    EmployeeId INT NOT NULL PRIMARY KEY,
    Photo VARBINARY(MAX) NULL,
    MyRowGuidColumn UNIQUEIDENTIFIER NOT NULL ROWGUIDCOL UNIQUE
        DEFAULT NEWID()
);

```

The 'Messages' pane at the bottom right shows 'Commands completed successfully.'

Se pueden crear índices adicionales utilizando la palabra clave Index en la definición de la tabla. Esto puede ser útil cuando hay más de una columna en la tabla que se buscará con frecuencia. El siguiente ejemplo crea índices dentro de la instrucción Create table:

```

CREATE TABLE Bookstore2
(
    ISBN_NO      VARCHAR(15) NOT NULL PRIMARY KEY,
    SHORT_DESC   VARCHAR(100),
    AUTHOR       VARCHAR(40),
    PUBLISHER   VARCHAR(40),
    PRICE        FLOAT,
    INDEX SHORT_DESC_IND(SHORT_DESC, PUBLISHER)
);

```

Estrategias para indexar:

- Evite indexar tablas/columnas muy utilizadas: cuantos más índices haya en una tabla, mayor será el efecto en el rendimiento de las instrucciones Insertar, Actualizar, Eliminar y Merge porque todos los índices deben modificarse adecuadamente. Esto significa que SQL Server tendrá que dividir páginas, mover datos y tendrá que hacer de las eso para todos los índices afectados por esas declaraciones DML.
- Use claves de índice estrechas siempre que sea posible: mantenga los índices estrechos, es decir, con la menor cantidad de columnas posible. Las claves numéricas

exactas son las claves de índice SQL más eficientes (p. ej., números enteros). Estas claves requieren menos espacio en disco y gastos generales de mantenimiento.

- Use índices agrupados en columnas únicas : considere las columnas que son únicas o contienen muchos valores distintos y evítelas para las columnas que sufren cambios frecuentes
- Índices no agrupados en columnas que se buscan o unen con frecuencia : asegúrese de que los índices no agrupados se coloquen en foreign keys y columnas que se usan con frecuencia en las condiciones de búsqueda, como la cláusula Where que devuelve coincidencias exactas.
- Cubra los índices de SQL para obtener grandes ganancias de rendimiento : las mejoras se logran cuando el índice contiene todas las columnas de la consulta.

### 3. Optimización de consultas

Al consultar una base de datos, la optimización es clave. Una consulta ineficiente agotará los recursos de la base de datos y provocará un rendimiento lento o no disponibilidad. Es vital la optimización de consultas para lograr un impacto mínimo en el rendimiento de la base de datos.

Puede ocurrir por dos vías:

→ **Manual:**

Es aquella que realiza el programador, y será ampliada en una sección posterior.

En caso de que sea una base de datos relacional, es importante el conocimiento de álgebra relacional, del su diseño y la relación entre las tablas de la base.

En caso de que no lo sea, es importante conocer los algoritmos de búsqueda que se pueden utilizar y elegir los más óptimos.

Dos consultas distintas pueden traer los mismos resultados, pero tener eficiencias distintas, y utilizar una cantidad mayor o menor de recursos, además de demandar distintas cantidades de tiempo de procesamiento.

→ **Parte de la base de datos:**

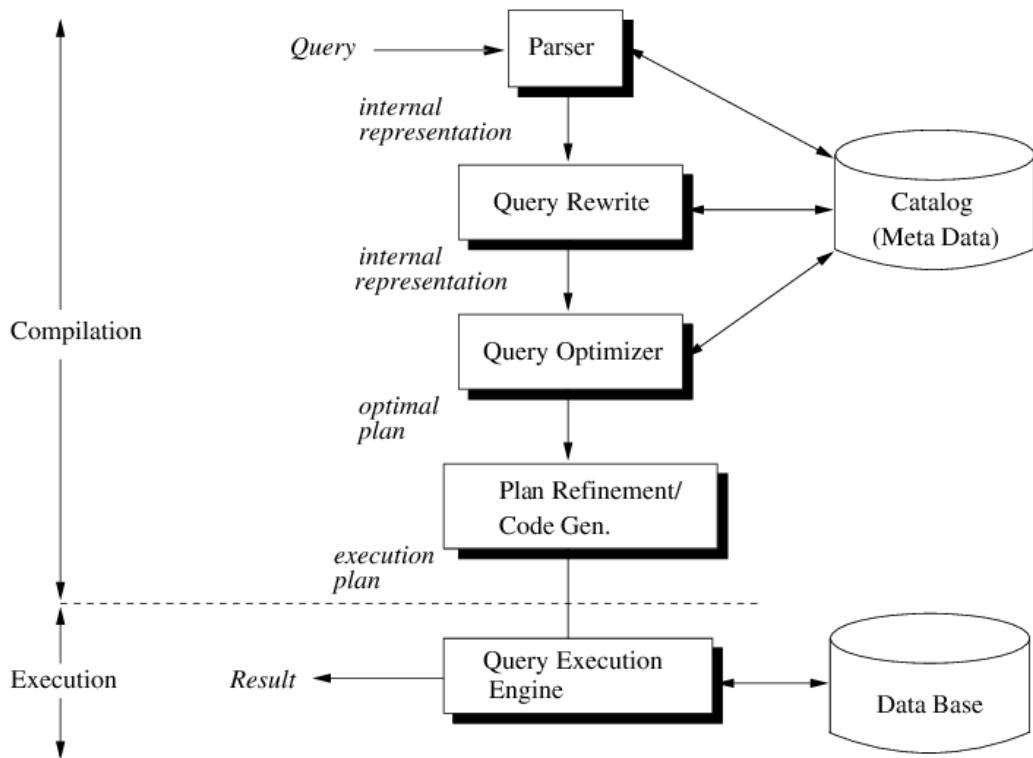
Puede haber estructuras intrínsecas de los sistemas de gestión de bases de datos, como el procesador de consultas, que tengan un impacto directo en el rendimiento.

El procesador de consultas es el encargado de traducir la consulta al plan de consultas más eficiente.

Un plan de consultas es la secuencia de operaciones a realizar sobre los datos.

Este tiene 3 partes:

- 1) **Query phaser:** Analiza la consulta y construye una estructura de árbol a partir del texto de la consulta.
- 2) **Query processor:** Procesa la consulta y realiza su comprobación semántica (verifica la existencia de lo solicitado) para poder descartar la consulta en caso de ser negativa.
- 3) **Query optimizer:** Transforma el plan inicial en la mejor secuencia de operaciones disponible.



### Seleccionar campos necesarios en lugar de usar SELECT \*

Consultar únicamente los campos que requerimos, es decir, si nuestra consulta tiene como objetivo obtener los datos de envío de un cliente.

Consulta ineficiente:

```
SELECT *
FROM Customers
```

La forma correcta de realizar la consulta sería:

```
SELECT FirstName, LastName, Address, City, State, Zip
FROM Customer
```

Es una consulta más limpia y solo trae los datos que requerimos.

### Evitar SELECT DISTINCT

Si bien sirve para eliminar registros duplicados, internamente agrupando todos los campos de la consulta, SELECT DISTINCT funciona agrupando todos los campos de la consulta para crear resultados distintos y esto requiere un gran poder de procesamiento.

Ineficiente:

```
SELECT DISTINCT FirstName, LastName, State
FROM Customers
```

En bases de datos grandes, una gran cantidad de Juan Perez harán que esta consulta se ejecute lentamente.

Eficiente:

```
SELECT FirstName, LastName, Address, City, State, Zip
FROM Customers
```

Añadiendo más campos, los campos no duplicados serán devueltos sin necesidad de un DISTINCT, por lo que la base no tendrá que agrupar ningún campo.

```
SELECT Customers.CustomerID, Customers.Name,  
Sales.LastSaleDate  
FROM Customers, Sales  
WHERE Customers.CustomerID = Sales.CustomerID
```

En una unión cartesiana(cross join), se crean todas las combinaciones posibles de las variables. En este ejemplo, si tuviéramos 1,000 clientes con 1,000 ventas totales, la consulta generaría primero 1,000,000 de resultados, luego filtraría los 1,000 registros donde CustomerID está correctamente unido. Este es un uso ineficiente de los recursos de la base de datos, ya que la base de datos ha realizado 100 veces más trabajo del requerido. Las uniones cartesianas son especialmente problemáticas en bases de datos a gran escala, porque una unión cartesiana de dos tablas grandes podría generar miles de millones o billones de resultados.

En lugar de usar una unión cartesiana, usar un INNER JOIN:

```
SELECT Customers.CustomerID, Customers.Name,  
Sales.LastSaleDate  
FROM Customers  
INNER JOIN Sales  
ON Customers.CustomerID = Sales.CustomerID
```

Algunos sistemas DBMS pueden reconocer las uniones WHERE y ejecutarlas automáticamente como INNER JOIN en su lugar. En esos sistemas DBMS, no habrá diferencia en el rendimiento entre un WHERE join y un INNER JOIN.

**Usar WHERE en lugar de HAVING para definir los filtros**

---

HAVING se calcula después de las declaraciones WHERE. Si lo que se requiere es filtrar una consulta según las condiciones, una declaración WHERE es más eficiente.

Por ejemplo, supongamos que se realizaron 200 ventas en el año 2016 y queremos consultar la cantidad de ventas por cliente en 2016.

```
SELECT Customers.CustomerID, Customers.Name,
Count(Sales.SalesID)
FROM Customers
INNER JOIN Sales
ON Customers.CustomerID = Sales.CustomerID
GROUP BY Customers.CustomerID, Customers.Name
HAVING Sales.LastSaleDate BETWEEN #1/1/2016# AND
#12/31/2016#
```

Esta consulta extraería 1000 registros de ventas de la tabla Ventas, luego filtraría los 200 registros generados en el año 2016 y finalmente contaría los registros en el conjunto de datos.

```
SELECT Customers.CustomerID, Customers.Name,
Count(Sales.SalesID)
FROM Customers
INNER JOIN Sales
ON Customers.CustomerID = Sales.CustomerID
WHERE Sales.LastSaleDate BETWEEN #1/1/2016# AND
#12/31/2016#
GROUP BY Customers.CustomerID, Customers.Name
```

---

Esta consulta extraería los 200 registros del año 2016 y luego contaría los registros en el conjunto de datos.

HAVING solo debe usarse cuando se filtra en un campo agregado (COUNT). Podríamos filtrar por clientes con más de 5 ventas utilizando una declaración HAVING.

```
SELECT Customers.CustomerID, Customers.Name,  
Count(Sales.SalesID)  
FROM Customers  
INNER JOIN Sales  
ON Customers.CustomerID = Sales.CustomerID  
WHERE Sales.LastSaleDate BETWEEN #1/1/2016# AND  
#12/31/2016#  
GROUP BY Customers.CustomerID, Customers.Name  
HAVING Count(Sales.SalesID) > 5
```

### Limitar la cantidad de datos recuperados

Utiliza la paginación o límites de resultados en las consultas para reducir la cantidad de datos que se recuperan y se transmiten.

```
SELECT name  
FROM Player  
WHERE point>100;
```

---

```
SELECT name
FROM Player
WHERE point>100
LIMIT 10;
```

Usar WILDCARDS solo al final de la frase.

Al buscar datos de texto sin formato, como ciudades o nombres, los wildcard crean la búsqueda más amplia posible. La búsqueda más amplia es también la más ineficiente.

Cuando se usa un wildcard inicial, especialmente en combinación con un wildcard al final, la base de datos tiene la tarea de buscar en todos los registros una coincidencia en cualquier lugar dentro del campo seleccionado.

Esta consulta para extraer ciudades que comienzan con 'Char':

```
SELECT City FROM Customers
WHERE City LIKE '%Char%'
```

Esta query va retornar registros como **Charleston**, **Charlotte**, and **Charlton**. However, it will also pull unexpected results, such as Cape **Charles**, Crab **Orchard**, and **Richardson**.

En lugar, consultar:

```
SELECT City FROM Customers
WHERE City LIKE 'Char%'
```

Traerá resultados como **Charleston**, **Charlotte**, and **Charlton**.

## Correr consultas complejas durante un horario no pico

Para minimizar el impacto de sus consultas analíticas en la base de datos la programación de la consulta para que se ejecute en un horario de menor actividad. La consulta debe ejecutarse cuando los usuarios simultáneos están en su número más bajo, que suele ser en medio de la noche (3 a 5 a. m.).

- Selección de tablas grandes (>1,000,000 registros)
- Uniones cartesianas o CROSS JOIN
- Declaraciones en bucle
- SELECT DISTINCT
- Subconsultas anidadas
- Búsquedas con wildcards
- Consultas de varios esquemas

## Evitar subconsultas innecesarias

Las subconsultas pueden ser costosas en términos de rendimiento. Evita anidar demasiadas subconsultas o utilizarlas de manera innecesaria. En su lugar, utiliza joins o técnicas de combinación de datos para obtener los resultados necesarios de manera más eficiente.

## 4. Mantenimiento

El mantenimiento para la mejora del rendimiento de una base de datos implica una serie de actividades destinadas a optimizar su funcionamiento con el fin de lograr un mejor rendimiento en términos de velocidad de procesamiento, eficiencia y utilización de recursos.

Algunas de las actividades comunes que se pueden realizar para mejorar el rendimiento de una base de datos incluyen:

- **Reorganización de índices:** Los índices son estructuras utilizadas para acelerar la búsqueda y recuperación de datos en una base de datos. Con el tiempo, los índices pueden volverse fragmentados y desorganizados, lo que puede afectar negativamente el rendimiento de las consultas. La reorganización de índices implica reorganizar físicamente los índices para eliminar la fragmentación y mejorar la eficiencia de las operaciones de búsqueda y recuperación de datos.
- **Actualización de estadísticas:** Las estadísticas son utilizadas por el sistema de gestión de bases de datos (SGBD) para tomar decisiones sobre cómo ejecutar consultas y optimizar el rendimiento. Es importante mantener actualizadas las estadísticas de la base de datos para asegurarse de que el SGBD tenga la información más reciente sobre la distribución de datos y pueda tomar decisiones de optimización adecuadas.
- **Ajuste de la configuración del SGBD:** Los sistemas de gestión de bases de datos suelen tener configuraciones que afectan su rendimiento. Revisar y ajustar la configuración del SGBD, como el tamaño de memoria asignada, la cantidad de conexiones permitidas, la configuración de caché y otros parámetros de rendimiento, puede tener un impacto en el rendimiento general de la base de datos.
- **Compactación de tablas:** Las tablas pueden volverse fragmentadas y ocupar más espacio del necesario en disco, lo que puede afectar negativamente el rendimiento de la base de datos. La compactación de tablas implica reorganizar físicamente los datos en las tablas para reducir la fragmentación y el espacio ocupado en disco, lo que puede mejorar la eficiencia en la búsqueda y recuperación de datos.
- **Gestión de transacciones:** Las transacciones son operaciones que se realizan en la base de datos y pueden afectar el rendimiento si no se gestionan adecuadamente. Es importante asegurarse de que las transacciones se manejen de manera eficiente, con un compromiso adecuado entre la duración de las transacciones y la cantidad de recursos utilizados, para evitar cuellos de botella y optimizar el rendimiento.

- **Actualización del software del SGBD:** Mantener actualizado el software del SGBD con las últimas actualizaciones y parches puede mejorar el rendimiento y la seguridad de la base de datos. Las actualizaciones del software suelen incluir mejoras de rendimiento y correcciones de errores, por lo que es importante mantener la base de datos actualizada con las últimas versiones

## 5. Escalabilidad

La escalabilidad de una base de datos se refiere a su capacidad para manejar eficientemente un crecimiento en la cantidad de datos y la carga de trabajo sin perder rendimiento o eficiencia. Una base de datos escalable es capaz de adaptarse y crecer con las necesidades de almacenamiento y procesamiento de datos a medida que estas aumentan con el tiempo.

Existen dos tipos principales de escalabilidad en una base de datos:

### **Escalabilidad vertical:**

Implica agregar más recursos a un servidor o máquina para mejorar el rendimiento de la base de datos. Esto puede incluir agregar más CPU, memoria RAM o almacenamiento en disco al servidor que aloja la base de datos. La escalabilidad vertical suele ser más sencilla de implementar, pero tiene un límite físico en cuanto a la cantidad de recursos que se pueden agregar a un solo servidor.

### **Escalabilidad horizontal:**

Implica agregar más servidores a un sistema distribuido para manejar la carga de trabajo. Esto implica dividir los datos y las tareas de procesamiento entre varios servidores que trabajan en paralelo para mejorar el rendimiento y la capacidad de manejo de la base de datos.

---

Se configuran en forma de una red de servidores denominada clúster. Esta tiene la finalidad de dividir eficientemente la demanda de trabajo entre todos los nodos que conforman la red de servidores.

Es importante tener en cuenta que lograr una verdadera escalabilidad en una base de datos puede requerir una planificación y diseño adecuados desde el inicio, como el uso de técnicas de diseño de bases de datos distribuidas, particionamiento de datos, uso de clústeres y técnicas de replicación, entre otros. Además, el monitoreo y la optimización constantes son esenciales para asegurar un rendimiento óptimo en una base de datos escalable a medida que crece y se enfrenta a mayores demandas de carga de trabajo.



# Bases de Datos NoSQL (Not Only SQL)

Lucas Castronuovo y Sebastián Roca

# ¿Qué son las BD NoSQL?



NoSQL es utilizado para describir un subconjunto de bases de datos que difiere en distintos modos de bases de datos tradicionales. Son útiles cuando se necesita acceder y analizar gran cantidad de datos no estructurados o datos que se almacenen de forma remota en varios servidores virtuales o en la nube. Resuelve problemas de almacenamiento masivo y alto desempeño. No usa SQL como principal lenguaje. De ahí viene su nombre NotOnly SQL.

## Breve Historia

El término NoSQL fue acuñado en 1998 por Carlo Trozzi, Not Only SQL.

Erick Evans retomó el término en 2009, la diferencia es que Erick Evans sugiere referirse a esta familia de base de datos como big data.

# Definiciones



*"NoSQL abarca una amplia variedad de **diferentes tecnologías** de bases de datos que se desarrollaron en respuesta a las **demandas** presentadas en la construcción de aplicaciones **modernas**"*

## MongoDB

*"Un ambiente de base de datos NoSQL es, simplemente, un **sistema** de base de datos **no***

**Relacional y ampliamente distribuido** que permite una organización **rápida** y ad hoc y análisis de tipos de datos muy **dispareos**. Las bases de datos NoSQL a veces se denominan bases de datos en la nube, bases de datos no relacionales, bases de datos Big Data y una miríada de otros términos y se desarrollaron en respuesta al gran volumen de datos generados, almacenados y analizados por usuarios modernos (datos generados por el usuario) y sus aplicaciones (datos generados por máquina)".

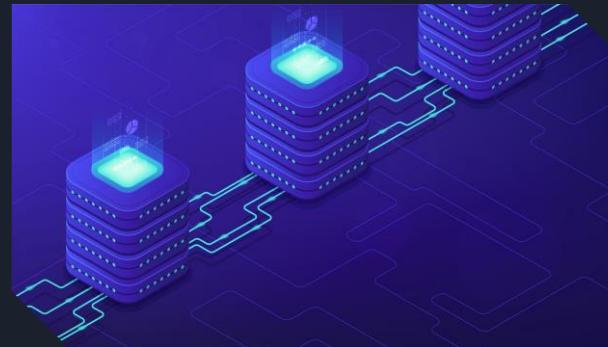
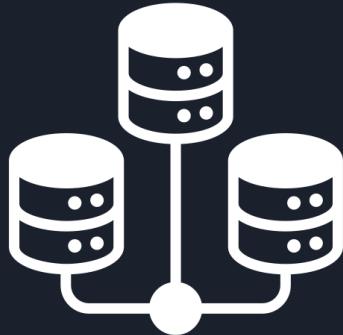
Datastax

*"La intención original ha sido desarrollar las modernas bases de datos de la **web**. El movimiento comenzó a principios de 2009 y está creciendo rápidamente. A menudo más características se aplican, tales como: libertad de esquemas, fácil soporte de replicación, API simple, consistencia eventual (no ACID), una gran cantidad de datos y mucho más. Así que el término engañoso "nosql" (la comunidad ahora lo traduce principalmente como "no sólo sql") debe ser visto como un alias a algo parecido a la definición de arriba".*

NoSql-database.org

# Arquitecturas NoSQL

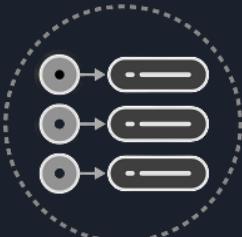
- Generalidades
  - Consistencia débil o transacciones restringidas
  - Arquitectura Distribuida
  - Estructura de datos sencillas



# Arquitecturas NoSQL

## BD de Clave-Valor

- Estructura de datos sencilla
- Formada por tablas donde se almacenan filas o elementos



## BD de Documentos

- Se almacena en forma de documentos
- Conjunto de pares clave-valor
- Se puede contener otros documentos



## BD de Columnas

### Extendidas

- BD ordenada en un mapa multidimensional
- Definida por
  - Rowkey
  - Columnkey
  - Timestamp

The diagram illustrates the difference between row-oriented and column-oriented databases. It shows two tables side-by-side. The left table is labeled 'Base de datos orientada a filas' (row-oriented database) and has columns 'Nombre', 'Dirección', and 'Ciudad'. The right table is labeled 'Base de datos orientada a columnas' (column-oriented database) and has columns 'Nombre', 'Dirección', and 'Ciudad' with specific values for María, José, and Luisa.

Nombre	Dirección	Ciudad
Maria	Calle Mallorca	Madrid
José	Avenida Castellana	Sevilla
Luisa	Calle Aragón	Barcelona

Nombre	Maria	José	Luisa
Dirección	Calle Mallorca	Avenida Castellana	Calle Aragón
Ciudad	Madrid	Sevilla	Barcelona

## BD de Grafos

- Basada en grafos
- Usado en motores de recomendación y aplicaciones geoespaciales



# Big Data y su relación con NoSQL



Big Data se refiere a datos que son masivamente generados. Un tipo de tecnología que ha surgido para tratar de poner solución a muchos de los problemas de los que adolecen los sistemas de almacenamiento tradicionales cuando intentan manejar este tipo de datos masivos. Son las tecnologías que se conocen como NoSQL.

## Definición

Big Data es la ingente cantidad de información, en su mayor parte desestructurada, que hoy en día generamos toda la sociedad como consecuencia de nuestra actividad tanto en Internet como fuera de ella.

# Definiciones



*“Big data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn't fit the strictures of your database architectures. To gain value from this data, you must choose an alternative way to process it.”*

O'Really 2012

En el blog oficial de Microsoft Enterprise se puede leer:

*“Big data is the term increasingly used to describe the process of applying serious computing power — the latest in machine learning and artificial intelligence — to seriously massive and often highly complex sets of information”*

HowieT 2013

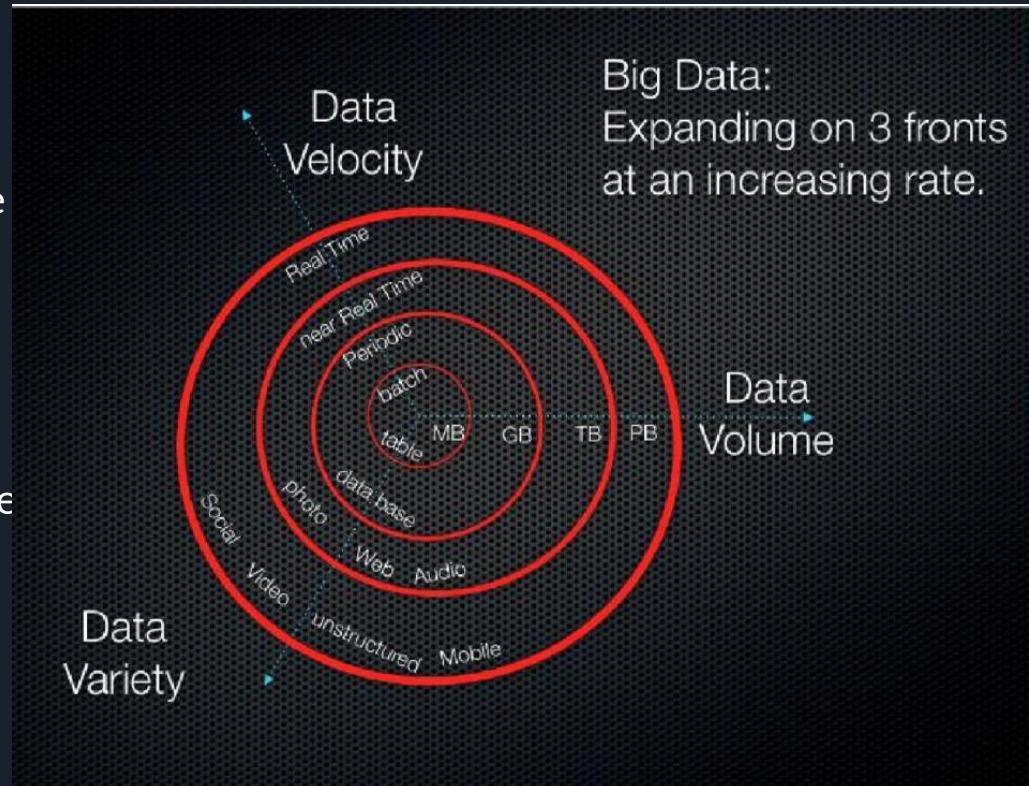
Adrian Mery, de la revista Teradata expone la siguiente definición: *“Big data exceeds the reach of commonly used hardware environments and software tools to capture, manage and process it within a tolerable elapsed time for its user population”*

Merv 2011

# 3V's : Volumen, Velocidad y Variedad

Volumen, Velocidad y Variedad son los principales factores que intentan mejorarse con las B.D. NoSQL.

En los últimos 10 años, el crecimiento del Volumen de Datos No Estructurados que se generan a gran velocidad llevó a que se desarrollen nuevos paradigmas de almacenamiento.



# Funcionalidad

- Las bases de datos NoSQL son requeridas cuando se cumplen estos puntos:
  - Mayor Velocidad en que los datos se generan
  - Mayor Volumen de Datos
  - Necesidad de almacenar datos de formatos Variados
  - Ser ACID no es mandatorio
- Las bases de datos NoSQL se utilizan para:
  - Desarrollo de Redes Sociales
  - Desarrollo Web o Móvil
  - Big Data
- Las bases de datos NoSQL no se utilizan para
  - Sistemas consistentes, donde la posibilidad de error debe ser baja
  - Uso empresarial
  - Rama de negocios



# Funcionalidad

## BD de Clave-Valor

- Almacenamiento en diccionarios

- Cada registro se define como un conjunto de una clave que referencia a un valor

- 

## BD de Documentos

- Conformado por programas que almacenan, recuperan y gestionan datos de documento
- Diseño basado en una noción abstracta de un documento

## BD de Columnas Extendidas

- Almacenamiento de datos en columnas en vez de filas
- Los datos se leen y se escriben de manera eficiente en el disco duro
- El almacenamiento de datos se da por orden de registro

## BD de Grafos

- Se almacena la información como nodos de un grafo y sus respectivas relaciones con otros nodos
- Las relaciones pueden tener propiedades, definidas por pares clave-valor

# Generalidades

## Ventajas

La escalabilidad y su carácter descentralizado.

Soportan estructuras distribuidas.

Abiertas y flexibles.

Cambios de los esquemas sin tener que parar bases de datos.

Escalabilidad horizontal

Se pueden ejecutar en máquinas con pocos recursos.

Optimización de consultas en base de datos para grandes cantidades de datos.

## Desventajas

No contemplan la atomicidad de las instrucciones y la integridad de los datos.

Soportan lo que se llama consistencia eventual.

Problemas de compatibilidad entre instrucciones SQL.

Falta de estandarización.

No hay soporte multiplataforma.

# Ventajas y desventajas específicas

## BD de Clave-Valor

- Ventajas: Modelo dinámico, Estructuras genérica, Altamente flexible, permite agregar atributos en una etapa posterior al diseño.
- Desventaja: No asegura la integridad del tipo de dato.

## BD de Documentos

- Ventajas: La posibilidad de usar estructuras anidadas, soportar arreglos y entender diferentes tipos de datos.
- Desventaja: No asegura la integridad del tipo de dato.

## BD de Columnas Extendidas

- Ventajas: Flexibilidad, puede agregar columnas de diferentes filas.
- Desventaja: No asegura la integridad del tipo de dato.

## BD de Grafos

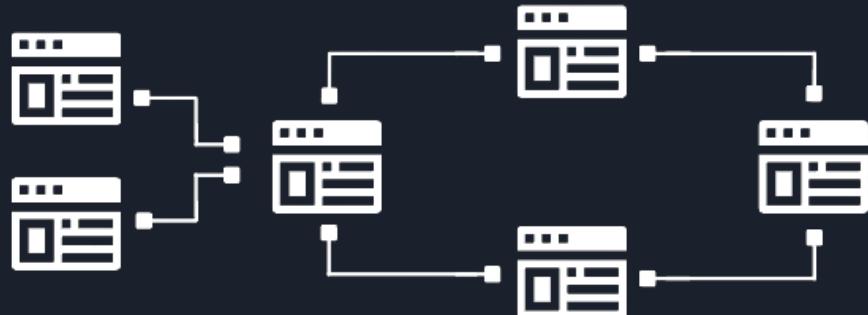
- Ventajas: las estructuras internas son eficientes para almacenar y buscar en forma de estructuras de grafos.

# Comparación con otras Bases de Datos

## Base de datos NoSQL y Base de datos Relacional

Una Base de Datos NoSQL se diferencia de una Base de datos Relacional teniendo en cuenta que:

- No permite joins.
- No garantizan ACID
- Indexable
- Colecciones de tamaño fijo
- Teorema CAP



# Comparación con otras Bases de Datos

## Base de datos NoSQL y Base de datos Orientada a Objetos

- Las bases de datos NoSQL, al principio, solían ser bases de datos orientadas a objetos
- La Popularidad de las bases de datos NoSQL influenció a los proveedores de las bases de datos orientada a objetos



# Productos de Bases de Datos NoSQL

## BD de Clave-Valor

- Redis
- Amazon DynamoDB
- Riak

## BD de Documentos

- CouchDB
- Couchbase
- Marklogic
- MongoDB

## BD de Columnas

### Extendidas

- Hypertable
- Accumulo
- Apache Cassandra
- Apache HBASE
- Amazon SimpleDB

## BD de Grafos

- Neo4j
- InfiniteGraph



HYPERTABLE<sup>INC</sup>



# ¿Como se usan las BD NoSQL?

Dado la variedad de base de datos NoSQL, decidimos elegir Dynamodb como un ejemplo particular.

Tutorial de instalación y uso [https://www.youtube.com/watch?v=liTlg\\_aweBk](https://www.youtube.com/watch?v=liTlg_aweBk)



# **Capa de persistencia**

**(2023-1C) Laboratorio IV**

## **Integrantes**

Luciano Pastocchi  
Martín Pallero

**Cuatrimestre / Año:**  
**I/2023**

## Introducción

La capa de persistencia es un componente esencial en el diseño de bases de datos y sistemas de información. Su principal función es gestionar el almacenamiento, recuperación y manipulación de los datos en un sistema. En este documento, analizaremos en profundidad el concepto de capa de persistencia, sus principales características y cómo se implementa en diferentes sistemas de bases de datos.

## Concepto de capa de persistencia

La capa de persistencia es una abstracción que permite separar la lógica de negocio de la gestión de almacenamiento y recuperación de datos. De esta manera, los desarrolladores pueden centrarse en la implementación de la lógica de negocio sin preocuparse por los detalles técnicos de cómo se almacenan y recuperan los datos. La capa de persistencia se encarga de traducir las operaciones realizadas en la lógica de negocio a operaciones específicas del sistema de almacenamiento subyacente, como consultas SQL en bases de datos relacionales o comandos específicos en bases de datos NoSQL.

## Origen

La capa de persistencia en bases de datos es una técnica que ha existido desde los primeros días de la informática y el almacenamiento de datos electrónicos. Sin embargo, la capa de persistencia como la entendemos hoy en día, con su capacidad de mapear objetos de programación a filas y columnas de una base de datos relacional, se popularizó a principios de la década de 2000.

## Principales características de la capa de persistencia

### Abstracción

La capa de persistencia proporciona una interfaz de alto nivel que oculta los detalles de implementación del sistema de almacenamiento. Esto facilita la sustitución del sistema de almacenamiento subyacente sin afectar a la lógica de negocio, ya que las operaciones realizadas en la capa de persistencia son independientes del sistema de almacenamiento concreto.

## Uniformidad

La capa de persistencia permite unificar el acceso a diferentes sistemas de almacenamiento, proporcionando una interfaz común para realizar operaciones de lectura, escritura, actualización y eliminación de datos. Esto facilita la integración de distintos sistemas de bases de datos en una única aplicación.

## Flexibilidad

Gracias a la abstracción y la uniformidad que ofrece la capa de persistencia, es posible modificar la forma en que se almacenan y gestionan los datos sin afectar a la lógica de negocio. Esto permite adaptar el sistema de almacenamiento a las necesidades cambiantes de la aplicación, como la incorporación de nuevas funcionalidades o el escalado para soportar un mayor volumen de datos.

## Ventajas

### Durabilidad de los datos

Una de las principales ventajas de una capa de persistencia es que los datos almacenados en la base de datos están protegidos contra pérdidas debido a errores de hardware o software. Esto garantiza la durabilidad de los datos y permite una recuperación rápida y eficiente en caso de fallos.

### Integridad de los datos

La capa de persistencia garantiza que los datos almacenados en la base de datos sean coherentes y precisos. Esto es especialmente importante en aplicaciones críticas donde la precisión de los datos es vital.

### Escalabilidad

Las capas de persistencia permiten la escalabilidad de una aplicación, ya que permiten almacenar grandes cantidades de datos en la base de datos. Además, las capas de persistencia pueden ser escaladas horizontalmente mediante la adición de más nodos o servidores de base de datos.

### Acceso concurrente

Las capas de persistencia permiten el acceso concurrente a los datos almacenados en la base de datos, lo que significa que varios usuarios pueden acceder y manipular los datos

al mismo tiempo. Esto es importante en aplicaciones con muchos usuarios que acceden a los mismos datos al mismo tiempo.

## **Desventajas**

### **Costo**

Una de las principales desventajas de utilizar una capa de persistencia en una base de datos es el costo. La implementación y el mantenimiento de una capa de persistencia pueden ser costosos, especialmente en aplicaciones de alta disponibilidad y alto rendimiento.

### **Complejidad**

La capa de persistencia puede ser muy compleja y puede requerir habilidades especializadas para implementar y mantener. Esto puede aumentar el costo y la complejidad del desarrollo de una aplicación.

### **Rendimiento**

El uso de una capa de persistencia puede afectar el rendimiento de una aplicación. Los tiempos de acceso a los datos pueden ser más lentos debido a la necesidad de escribir y leer datos de la base de datos. Además, la sobrecarga de las transacciones de la base de datos puede afectar el rendimiento de la aplicación.

### **Vulnerabilidades de seguridad**

La capa de persistencia puede ser vulnerable a ataques de seguridad, como la inyección de SQL. Es importante implementar medidas de seguridad adecuadas para proteger la base de datos y los datos almacenados en ella.

## **Implementación de la capa de persistencia en diferentes sistemas de bases de datos**

### **Bases de datos relacionales**

En sistemas de bases de datos relacionales, como MySQL o PostgreSQL, la capa de persistencia se implementa mediante el uso de un Object Relational Mapping (ORM). Un ORM es una herramienta que mapea las estructuras de datos en la lógica de negocio a

tablas y relaciones en una base de datos relacional. Los ORMs permiten realizar operaciones de CRUD (Create, Read, Update, Delete) utilizando objetos y métodos de la lógica de negocio, en lugar de consultas SQL directamente.

### Bases de datos NoSQL

En sistemas de bases de datos NoSQL, como MongoDB o Cassandra, la capa de persistencia se implementa utilizando un Object Document Mapper (ODM) o similar. Al igual que los ORMs, estas herramientas permiten realizar operaciones de CRUD utilizando objetos y métodos de la lógica de negocio, en lugar de comandos específicos del sistema de almacenamiento NoSQL.

### Licenciamiento

El licenciamiento de una capa de persistencia en bases de datos dependerá del proveedor de la tecnología en cuestión y de las características específicas de la licencia que se esté adquiriendo.

En general, los proveedores de bases de datos suelen ofrecer diferentes opciones de licenciamiento, que pueden variar en función del tamaño de la empresa, del número de usuarios, de las funcionalidades incluidas, etc.

Algunos proveedores de bases de datos ofrecen licencias por usuario, mientras que otros ofrecen licencias por servidor. Además, algunos proveedores ofrecen licencias por uso, es decir, en función del número de transacciones o del volumen de datos almacenados.

En cuanto a la capa de persistencia en sí misma, es posible que algunos proveedores la incluyan en la licencia de la base de datos de manera automática, mientras que otros la ofrezcan como un producto independiente con su propia licencia.

Es importante revisar cuidadosamente las opciones de licenciamiento ofrecidas por el proveedor de la base de datos y asegurarse de que se entienden claramente los términos y condiciones de la licencia antes de adquirirla.

### Principales empresas en donde funciona y proveedores

#### Amazon

Utiliza una variedad de tecnologías de persistencia de datos, incluyendo DynamoDB, RDS y Aurora.

### Microsoft

Utiliza Entity Framework como su capa de persistencia de datos para .NET Framework.

### Google

Utiliza Cloud Datastore y Cloud SQL como capas de persistencia para sus aplicaciones en la nube.

### IBM

Utiliza IBM Data Server Driver para la capa de persistencia en sus aplicaciones de base de datos.

### Oracle

Utiliza Oracle Database y Oracle Application Development Framework como su capa de persistencia para sus aplicaciones empresariales.

### Salesforce

Utiliza Salesforce Object Query Language (SOQL) y Salesforce Object Search Language (SOSL) como su capa de persistencia en la plataforma Salesforce.

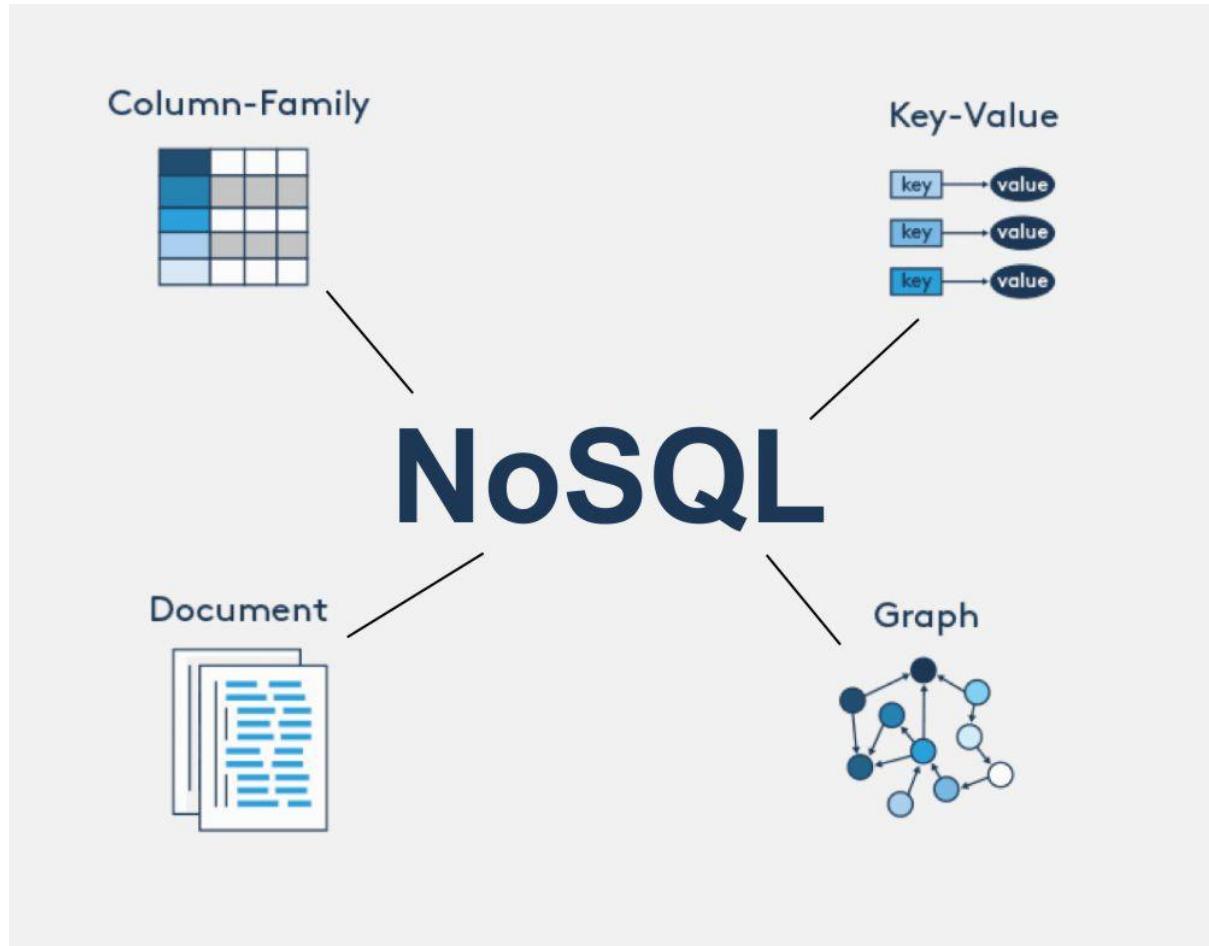
### Conclusión

La capa de persistencia juega un papel crucial en el diseño y desarrollo de aplicaciones y sistemas de información que utilizan bases de datos. Al ofrecer una abstracción que desacopla la lógica de negocio de la administración del almacenamiento y acceso a los datos, la capa de persistencia permite a los desarrolladores enfocarse en la creación de funcionalidades y en adaptarse a las demandas cambiantes de la aplicación. Además, debido a su flexibilidad y uniformidad, la capa de persistencia facilita la integración de distintos sistemas de bases de datos y el cambio del sistema de almacenamiento subyacente cuando sea necesario.

En resumen, la capa de persistencia es esencial para asegurar la escalabilidad, sostenibilidad y versatilidad de los sistemas de bases de datos actuales. Comprender su funcionamiento y cómo se implementa en diferentes sistemas de bases de datos es fundamental para cualquier experto en bases de datos y desarrollador de aplicaciones.

# Base de datos NoSQL (Not Only SQL)

por: Lucas Castronuovo y Sebastián Roca



## ¿Qué son las bases de datos NoSQL?

### Definición

El término NoSQL fue acuñado en 1998 por Carlo Trozzi y retomado en 2009 por Erick Evans, la diferencia es que Erick Evans sugiere referirse a esta familia de base de datos como big data.

Se generan a partir del uso de sistemas web y de respuesta inmediata, esto relacionado con que se consume un volumen de datos mayor así como la utilización de datos desde distintos usuarios y servicios.

Estas bases crecieron conjunto de las empresas líderes de servicios web como son: Google, Amazon, Facebook y Twitter, ya que estas tenían que enfrentarse a desafíos con el tratamiento de datos, a las cuales las tradicionales bases de datos no solucionaban

NoSql es utilizado para describir un subconjunto de bases de datos que difiere en distintos modos de bases de datos tradicionales. Son útiles cuando se necesita acceder y analizar gran cantidad de datos no estructurados o datos que se almacenen de forma remota en varios servidores virtuales o en la nube. Resuelve problemas de almacenamiento masivo y alto desempeño. No usa SQL como principal lenguaje. De ahí viene su nombre NotOnly SQL.

### Definición según

*"NoSQL abarca una amplia variedad de diferentes tecnologías de bases de datos que se desarrollaron en respuesta a las demandas presentadas en la construcción de aplicaciones modernas"*

MongoDB

*"Un ambiente de base de datos NoSQL es, simplemente, un sistema de base de datos no Relacional y ampliamente distribuido que permite una organización rápida y ad hoc y análisis de tipos de datos muy dispares. Las bases de datos NoSQL a veces se denominan bases de datos en la nube, bases de datos no relacionales, bases de datos Big Data y una miríada de otros términos y se desarrollaron en respuesta al gran volumen de datos generados, almacenados y analizados por usuarios modernos (datos generados por el usuario) y sus aplicaciones (datos generados por máquina)".*

Datastax

*"La intención original ha sido desarrollar las modernas bases de datos de la web. El movimiento comenzó a principios de 2009 y está creciendo rápidamente. A menudo más características se aplican, tales como: libertad de esquemas, fácil soporte de replicación, API simple, consistencia eventual (no ACID), una gran cantidad de datos y mucho más. Así que el término engañoso "nosql" (la comunidad ahora lo traduce principalmente como "no sólo sql") debe ser visto como un alias a algo parecido a la definición de arriba".*

## Big data

En los últimos años, la cantidad de datos digitales que se generan en el mundo se ha multiplicado. Las redes sociales y el cada vez más fácil acceso a Internet del que disponemos las personas hacen que el volumen de tráfico y de datos que se generan sea cada vez mayor.

Con el surgimiento de las bases de datos relacionales las empresas encontraron el aliado perfecto para cubrir sus necesidades de almacenamiento, disponibilidad, copiado de seguridad y gestión de sus datos.

Pero debido a las tendencias actuales de uso de Internet, este tipo de sistemas han comenzado a experimentar dificultades técnicas, en algunos casos bloqueantes, que impiden el buen funcionamiento de los sistemas de algunas de las empresas más importantes de Internet. Este tipo de datos que son masivamente generados reciben un nombre: BigData, y un tipo de tecnología ha surgido para tratar de poner solución a muchos de los problemas de los que adolecen los sistemas de almacenamiento tradicionales cuando intentan manejar este tipo de datos masivos. Esta tecnología se conoce como NoSql.

### **Definición**

Big Data es la ingente cantidad de información, en su mayor parte desestructurada, que hoy en día generamos toda la sociedad como consecuencia de nuestra actividad tanto en Internet como fuera de ella.

### **Definición según**

*"Big data is data that exceeds the processing capacity of conventional database systems. The data is too big, moves too fast, or doesn't fit the strictures of your database architectures. To gain value from this data, you must choose an alternative way to process it."*

O'Really 2012

En el blog oficial de Microsoft Enterprise se puede leer:

*"Big data is the term increasingly used to describe the process of applying serious computing power — the latest in machine learning and artificial intelligence — to seriously massive and often highly complex sets of information"*

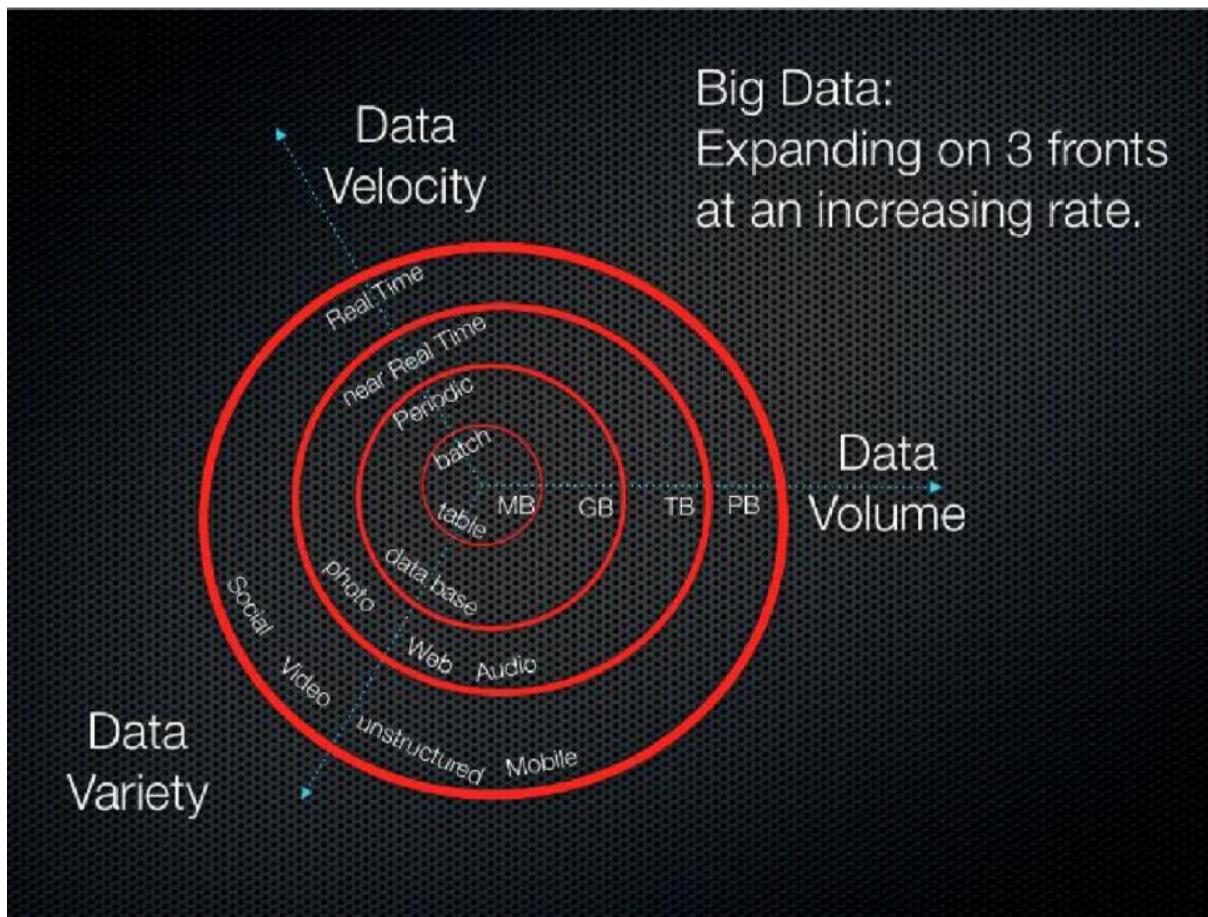
HowieT 2013

Adrian Mery, de la revista Teradata expone la siguiente definición: *"Big data exceeds the reach of commonly used hardware environments and software tools to capture, manage and process it within a tolerable elapsed time for its user population"*

Merv 2011

En los últimos 10 años, el crecimiento del Volumen de Datos No Estructurados que se generan a gran Velocidad llevó a que se desarrollen nuevos paradigmas de almacenamiento. Se pueden destacar las 3V's:

- Volumen
- Velocidad
- Variedad



## Arquitecturas NoSQL

### **Generalidades**

Ofrecen garantías de consistencia débil, o transacciones restringidas a elementos de datos simples.

Emplean una arquitectura distribuida en donde los datos se guardan de modo redundante en distintos servidores. Esto es importante por que esto permite que las bases NoSQL puedan escalar horizontalmente y tener una copia completa en cada nodo.

Suelen ofrecer estructuras de datos sencillas como arrays asociativos o almacenes de pares clave-valor.

### **Bases de datos de Clave-Valor**

Posee una estructura de datos sencilla. Está formada por tablas donde se almacenan filas o elementos. Estos últimos no tienen porque tener los mismos atributos ni la misma cantidad.

### **Bases de datos de Documentos**

Almacena elementos en forma de documentos, codificados en formato JSON. Cada elemento es un conjunto de pares clave valor. Cada documento puede contener otros documentos.

### **Bases de Datos de Columnas Extendidas**

Es una base de datos distribuida, persistente y ordenada en un mapa multidimensional. Una celda de esta base de datos viene definida por una clave primaria de fila (rowkey), por una clave de columna (columnkey) y una marca de tiempo (timestamp).

### **Bases de datos de Grafos**

Es una base de datos basada en grafos, que son un conjunto de nodos y relaciones, usos comunes incluyen motores de recomendación y aplicaciones geoespaciales.

## **Funcionalidad**

### **Las bases de datos NoSQL son requeridas cuando se cumplen estos puntos:**

- Mayor Velocidad en que los datos se generan
- Mayor Volumen de Datos
- Necesidad de almacenar datos de formatos Variados
- Necesidad de Escalar en forma ilimitado
- Necesidad de Tiempos de Respuesta inmediatos
- Aplicaciones Web/Mobile 7x24 con alta disponibilidad
- Ser ACID no es mandatorio
- BASES DE DATOS diseñadas para SISTEMAS DISTRIBUIDOS

### **Comúnmente, las bases de datos NoSQL se utilizan para:**

- Desarrollo de Redes Sociales
- Desarrollo Web o Móvil
- Big Data
- En aquellos sistemas donde las estructuras de los datos son variables
- Análisis de grandes cantidades de datos en modo lectura
- Captura y procesado de eventos

### **Comúnmente, las bases de datos NoSQL no se utilizan para:**

- En aquellos sistemas donde los datos deben ser consistentes sin dar posibilidad de error
- Uso empresarial
- Rama de negocios

**Bases de Datos de Clave-Valor:** El almacenamiento de datos se da desde un método sencillo de clave-valor. Estos son almacenados en diccionarios. Los diccionarios contienen

una colección de objetos referenciados con una clave que identifica los datos asociados de manera única y que se utiliza para encontrar los datos en la base de datos. Esto determina que cada registro se define como un conjunto de una clave que referencia un valor.

**Bases de Datos de Documentos:** Está conformado por programas que almacenan, recuperan y gestionan datos de documentos, por lo que su diseño se determina desde una noción abstracta de documento. Un documento es una forma de organizar y almacenar la data como un set de pares de campo-valor, siendo el campo un identificador único para un datapoint y el valor es la data relacionada a un determinado identificador.

**Bases de Datos de Columnas Extendidas:** se encarga de almacenar los datos en columnas en vez de filas. Esto busca lograr que los datos puedan escribirse y leerse de manera eficiente en el almacenamiento del disco duro, disminuyendo el tiempo en que se puedan obtener las consultas requeridas. Por lo tanto, todos los valores de una columna “..están físicamente juntos..”. El almacenamiento de datos se da por orden de registro. Al fin y al cabo, permitirá que se accedan a elementos de datos individuales en columnas que forman un grupo. Las bases de datos columnares están formadas por un Keyspace que estructura la base de datos columnar, formando estas familias de columnas. En cada fila se encuentra una Row Key que es la clave única e identificador de la fila, la Columna donde dentro de ella estará el nombre, un valor y una marca de tiempo (Es el timestamp, indica la fecha y hora en la cual se insertan los datos).

**Bases de Datos de Grafos:** Se almacena la información como nodos de un grafo y sus respectivas relaciones con otros nodos, permitiendo así aplicar la teoría de grafos para recorrer la base de datos. Las relaciones pueden tener propiedades, que se definen por pares clave-valor, y añaden información adicional para poder aplicar algoritmos y realizar consultas. Cada nodo consta de un grado que indica el número de aristas que tiene, a su vez un grafo puede ser dirigido o no dirigido, dependiendo de si las aristas tienen nodos origen y nodos destino.

## Comparación con otras bases de datos



### Relacional

Una Base NoSQL es una base orientada a Documentos, es decir no es relacional. No permiten Joins. Pero al mismo tiempo se puede generar uno. No intentan garantizar ACID (ATOMICIDAD, CONSISTENCIA, AISLAMIENTO, DURABILIDAD). Pero soporta Escalabilidad Horizontal, es de una escalabilidad muy sencilla. Utilizando ampliamente los usos de memoria principales de los equipos para soportar los altos volúmenes de información.

Por otro lado, es Indexable: O sea, por ejemplo, nos va a permitir utilizar índices Geoespaciales como para hacer búsquedas por cercanía. Permiten colecciones de tamaño fijo, que van a facilitarnos con una velocidad más alta en acceso y modificación de datos. También se puede destacar el Teorema de CAP, donde se resalta que un sistema computacional distribuido no puede ofrecer Consistencia, Disponibilidad (Availability) y Tolerancia a la partición (Partition) simultáneamente. Las bases de datos Relacionales se dedican a respetar las propiedades ACID, mientras que las bases de datos NoSQL rescinden ACID para dar mayor prioridad a la Disponibilidad y Tolerancia a las particiones.

### Orientada a Objetos

Las bases de datos NoSQL, al principio, solían ser bases de datos orientadas a objetos. Cuando las bases de datos NoSQL fueron más populares, los encargados de proveer bases de datos orientadas a objetos vieron la oportunidad de formar parte de este movimiento que se generaba.

## Ventajas y Desventajas

### **Generalidades**

Dado que no hay un único tipo de base de datos NoSQL, dependiendo de cual hablemos podemos encontrarnos con distintas ventajas y desventajas, pero en líneas generales estas cuentan con las siguientes ventajas:

- La escalabilidad y su carácter descentralizado. Soportan estructuras distribuidas.
- Suelen ser bases de datos mucho más abiertas y flexibles. Permiten adaptarse a necesidades de proyectos mucho más fácilmente que los modelos de Entidad Relación.
- Se pueden hacer cambios de los esquemas sin tener que parar bases de datos.
- Escalabilidad horizontal: son capaces de crecer en número de máquinas, en lugar de tener que residir en grandes máquinas.
- Se pueden ejecutar en máquinas con pocos recursos.
- Optimización de consultas en base de datos para grandes cantidades de datos.

Respecto a las desventajas de las bases de datos NoSQL en general, encontramos las siguientes:

- No todas las bases de datos NoSQL contemplan la atomicidad de las instrucciones y la integridad de los datos. Soportan lo que se llama consistencia eventual.
- Problemas de compatibilidad entre instrucciones SQL. Las nuevas bases de datos utilizan sus propias características en el lenguaje de consulta y no son 100% compatibles con el SQL de las bases de datos relacionales. El soporte a problemas con las queries de trabajo en una base de datos NoSQL es más complicado.
- Falta de estandarización. Hay muchas bases de datos NoSQL y aún no hay un estándar como si lo hay en las bases de datos relacionales.
- Soporte multiplataforma. Aún quedan muchas mejoras en algunos sistemas para que soporten sistemas operativos que no sean Linux.(Esto es algo que se deberá tomar en cuenta al elegir el motor)

### **Bases de datos de Clave-Valor**

Este tipo se caracteriza por almacenar los datos en pares clave-valor, lo que le da ventajas tales como: Modelo dinámico, Estructuras genérica, Altamente flexible, permite agregar atributos en una etapa posterior al diseño. También como resultado de esto se crea la desventaja de no asegurar la integridad del tipo de dato.

### **Bases de datos de Documentos**

Este tipo se caracteriza por almacenar documentos en la parte value del almacenamiento key-value, especificado en json. Las ventajas de este tipo son la posibilidad de usar estructuras anidadas, soportar arreglos y entender diferentes tipos de datos. Al igual que las claves-valor, tienen el problema de no asegurar la integridad del tipo de dato.

### **Bases de Datos de Columnas Extendidas**

Este tipo se caracteriza por como es la unidad básica de almacenamiento, una simple columna. Esto le da flexibilidad para agregar columnas de diferentes filas. Aunque al igual que las anteriores no asegura la integridad del tipo de dato.

### **Bases de datos de Grafos**

Este tipo se caracteriza por varias cosas como tener nodos relacionados entre sí. Cada uno tiene propiedades y los arcos tienen tipos y pueden tener múltiples propiedades. Por esto las estructuras internas son eficientes para almacenar y buscar en forma de estructuras de grafos.

## **Productos de Bases de Datos NoSQL**

Base de dato - Empresas que lo usan - lanzamiento

#### **En Base de Datos de Documentos:**

- CouchDB (Apache Software Foundation) - Ubuntu, BBC, Credit Suisse, Meebo - abril 2005
- Couchbase (Couchbase, Inc) - Zynga, Cisco, Adobe, Mc Graw Hill, Mozilla foundation, Honda, AOL, LG - agosto 2010
- MarkLogic (Marklogic) - BBC, Boeing, Bancos líderes mundiales
- MongoDB (MongoDB Inc) - Foursquare, LinkedIn, Cisco, eBay, Bosch - 2007

#### **En Base de Datos de Grafos:**

- Neo4j (Neo Technology) - Walmart, eBay, UBS, Nomura, Cisco, HP, Telenor - 2007
- InfiniteGraph (Objectivity, Inc.) - Objectivity inc. - mayo 2021

#### **En Base de Datos de Clave-Valor:**

- Redis (Salvatore Sanfilippo) - Twitter, GitHub, Pinterest, Snapchat, StackOverflow, Trello - abril 2009
- Amazon DynamoDB (Amazon) - Samsung, Toyota, Capital One, Dropbox, Zoom, Disney - 2012
- Riak (Basho Technologies) - AT&T, Comcast, Github, Best Buy, UK National Health Service, Riot Games - Agosto 2009

#### **En Base de Datos Columnas Extendidas:**

- Hypertable(Zvents inc.) - Baidu, GlusterFS, CloudStore - 2008
- Accumulo(NSA) - NSA(National Security Agency), DoD(Department of Defense) - 2012
- Apache Cassandra(Apache software foundation) - Twitter, Google, Amazon, Facebook, Instagram, Netflix, Apple - 2008
- Apache HBASE(Apache software foundation) - Facebook, Twitter, Yahoo, Adobe - 2006
- Amazon SimpleDB(Amazon) - Amazon, Google, Azure - diciembre 2007

## Como se usa una base de datos NoSQL?

Dado la variedad de base de datos NoSQL, decidimos elegir Dynamodb como un ejemplo particular.

### Tutorial de instalación y uso

Link: [https://www.youtube.com/watch?v=liTlg\\_aweBk](https://www.youtube.com/watch?v=liTlg_aweBk)

## Configuración

```
module.exports = {
  ...
  table_products: 'production_products',
  aws_local_config: {
    region: 'local',
    endpoint: 'http://localhost:8000'
  },
  aws_remote_config: {
  }
};
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash + - ×

added 14 packages from 66 contributors in 10.132s

New minor version of npm available! 6.3.0 → 6.10.0  
Changelog: <https://github.com/npm/cli/releases/tag/v6.10.0>  
Run `npm install -g npm` to update!

## Tabla

The screenshot shows the VS Code interface with the following details:

- Explorer View:** Shows the project structure with files like `package.json`, `config.js`, and `products.json`.
- Code Editor:** Displays the `products.json` file content, which defines a table schema with a primary key "id".
- Terminal:** Shows the command `npm install` being run, resulting in 14 packages being added from 66 contributors in 10.132s.
- Notification Bar:** A yellow box displays a message about a new npm version available (6.3.0 → 6.10.0) with a changelog link and a command to update.

The terminal window shows the following session:

```
...cal_lib -jar DynamoDBLocal.jar -sharedDb ...namodb_local_latest — aws configure
{
  "ItemCount": 0,
  "TableArn": "arn:aws:dynamodb:ddblocal:000000000000:table/production_products",
  "BillingModeSummary": {
    "BillingMode": "PROVISIONED",
    "LastUpdateToPayPerRequestDateTime": 0.0
  }
}
MacBook-Pro-de-Julio:dynamodb_local_latest juliopaez$ aws dynamodb list-tables --endpoint-url http://localhost:8000
{
  "TableNames": [
    "production_ad",
    "production_advertisement",
    "production_categories",
    "production_cities",
    "production_countries",
    "production_products",
    "production_states"
  ]
}
[MacBook-Pro-de-Julio:dynamodb_local_latest juliopaez$ aws configure
AWS Access Key ID [*****foo]: f]
```

## Funciones

```

var AWS = require("aws-sdk");
const config = require("./config/config.js");
AWS.config.update(config.aws_local_config);
const docClient = new AWS.DynamoDB.DocumentClient();
let saveProducts = function(){
    var object = {
        "id": "1",
        "name": "tennis Reebok",
        "amount": "50",
        "cost": "100 USD",
        "created_at": new Date().toString()
    };
    const params = {
        TableName: config.table_products,
        Item: object
    };
    docClient.put(params, function(err, data){
        if(err){
            console.log("Este es el error: ", err);
        }else{
            console.log("Item successfully added");
        }
    });
}

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: bash

added 14 packages from 66 contributors in 10.132s

New minor version of npm available! 6.3.0 → 6.10.0  
Changelog: <https://github.com/npm/cli/releases/tag/v6.10.0>  
Run `npm install -g npm` to update!

## Cálculo del precio

Link: <https://calculator.aws/#/addService/DynamoDB>

Configurar Amazon DynamoDB [Información](#)

Descripción

Elija un tipo de ubicación [Información](#) Elija una región

Región: Este de EE. UU. (Ohio)

Elegir las características de DynamoDB  
Elija las características de DynamoDB para las cuales desea calcular los precios.

DynamoDB on-demand capacity     DynamoDB provisioned capacity     DynamoDB Accelerator (DAX) clusters  
 DynamoDB Streams     DynamoDB Backup and restore     DynamoDB change data capture  
 DynamoDB Data export to Amazon S3     DynamoDB Data Import from Amazon S3

DynamoDB provisioned capacity

Clase de tabla [Información](#)

DynamoDB ofrece dos clases de tablas diseñadas para contribuir a la optimización de los costos. La clase de tabla DynamoDB Standard es la predeterminada y se recomienda para la gran mayoría de las cargas de trabajo. La clase de tabla DynamoDB Standard-Infrequent Access (DynamoDB Standard-IA) está optimizada para las tablas que almacenan datos a los que se accede con poca frecuencia, donde el almacenamiento es el principal costo. Cada clase de tabla ofrece diferentes precios para el almacenamiento de datos, así como para las solicitudes de lectura y escritura.

Costo inicial total: 180.00 USD    Costo total mensual: 26.14 USD    Mostrar detalles ▾

Guardar y ver resumen    Guardar y agregar servicio

## Bibliografía

[Material de la materia Laboratorio II \(NoSql-dto2.pdf\)](#)

[https://es.wikipedia.org/wiki/Amazon\\_SimpleDB](https://es.wikipedia.org/wiki/Amazon_SimpleDB)

<https://es.wikipedia.org/wiki/CouchDB>

<https://couchdb.apache.org>

[https://es.wikipedia.org/wiki/Servidor\\_de\\_CouchBase](https://es.wikipedia.org/wiki/Servidor_de_CouchBase)

<https://www.couchbase.com>

[http://erecursos.uacj.mx/bitstream/handle/20.500.11961/3053/Proyecto%20de%20titula\\_ci%C3%B3n%20base%20de%20datos%20no%20relacional%20couchbase%20y%20su%20impacto%20en%20las%20nuevas%20exigencias%20de%20manejo%20y%20control%20de%20datos%20en%20linea%20\\_282\\_29%20%282%29.Pdf?sequence=1&isAllowed=y#:~:text=Entre%20los%20clientes%20de%20Couchbase,%2C%20entre%20otros%20%5B1%5D.](http://erecursos.uacj.mx/bitstream/handle/20.500.11961/3053/Proyecto%20de%20titula_ci%C3%B3n%20base%20de%20datos%20no%20relacional%20couchbase%20y%20su%20impacto%20en%20las%20nuevas%20exigencias%20de%20manejo%20y%20control%20de%20datos%20en%20linea%20_282_29%20%282%29.Pdf?sequence=1&isAllowed=y#:~:text=Entre%20los%20clientes%20de%20Couchbase,%2C%20entre%20otros%20%5B1%5D.)

<https://www.tecnologias-informacion.com/nosql-empresas.html#:~:text=Los%20clientes%20de%20MarkLogic%20incluyen,y%20numerosos%20bancos%20I%C3%AAdderes%20mundiales.>

<https://en.wikipedia.org/wiki/MarkLogic>

<https://www.marklogic.com>

<https://mappinggis.com/2014/07/mongodb-y-gis/#:~:text=La%20lista%20de%20organizaciones%20que,Otras%20son%20eBay%2C%20Expedia.>

[https://go.neo4j.com/rs/710-RRC-335/images/Neo4j\\_CS\\_DieBayerische\\_ES.pdf](https://go.neo4j.com/rs/710-RRC-335/images/Neo4j_CS_DieBayerische_ES.pdf)

<https://es.wikipedia.org/wiki/Neo4j>

<https://github.com/neo4j/neo4j>

<https://neo4j.com>

<https://infinitegraph.com/about-us/>

<https://en.wikipedia.org/wiki/InfiniteGraph>

<http://infinitegraph.com>

<https://adrianalonsodev.medium.com/utilizando-redis-como-sistema-de-cache-en-symfony-437e449b8c0b#:~:text=Para%20que%20puedas%20entender%20el,%2C%20Snapshot%2C%20StackOverflow%20o%20Trello.>

<https://es.wikipedia.org/wiki/Redis>

<https://redis.io>

<https://platzi.com/tutoriales/1426-db-aws/5045-amazon-dynamodb-servicio-de-base-de-datos-nosql-rapido-y-flexible-para-cualquier-escala/#:~:text=Muchos%20de%20los%20negocios%20del,sus%20cargas%20de%20trabajo%20fundamentales.>

<https://calculator.aws/#/addService/DynamoDB>

<https://es.wikipedia.org/wiki/NoSQL>

<https://web.archive.org/web/20110217092700/http://www.nosql.es/blog/>

<https://hostingdata.co.uk/nosql-database/>

<https://www.mongodb.com/scale/types-of-nosql-database-management-systems>

<https://www.allthingsdistributed.com/2012/01/amazon-dynamodb.html>

[https://es.wikipedia.org/wiki/Amazon\\_SimpleDB](https://es.wikipedia.org/wiki/Amazon_SimpleDB)

[https://es.wikipedia.org/wiki/Apache\\_HBase](https://es.wikipedia.org/wiki/Apache_HBase)

<https://hbase.apache.org>

<https://github.com/apache/hbase>

[https://es.wikipedia.org/wiki/Apache\\_Cassandra](https://es.wikipedia.org/wiki/Apache_Cassandra)

[https://cassandra.apache.org/\\_/index.html](https://cassandra.apache.org/_/index.html)

<https://de.slideshare.net/grro/cassandra-by-example-the-path-of-read-and-write-requests>



UNIVERSIDAD DE PALERMO

Laboratorio IV



Facultad de Ingeniería

## Trabajo Práctico Grupal

### Informe - Base de Datos Embebida

**Año / Cuatrimestre**

2023 - 1C

**Curso / Código / Modalidad**

Laboratorio IV – 020149 – Presencial

**Docentes**

Marcela Russo

**Alumnos**

Andrés Isaac Biso - Legajo 0125044

Matías Hernán Coria - Legajo 0127111

Fecha de Presentación: 20/04/2023

Calificación: \_\_\_\_\_

## Índice

### Table of Contents

Índice.....	2
Consignas.....	3
Introducción.....	4
Historia.....	4
Acerca de SQLite.....	4
Descripción de la arquitectura.....	5
Interfáz (Interface).....	6
Tokenizador (tokenizer).....	6
Analizador Sintáctico (Parser).....	6
Generador de código (Code Generator).....	7
Motor Bytecode (Bytecode Engine).....	7
Árbol B (B-Tree).....	7
Caché de página (Page Cache).....	7
Interfaz del sistema operativo (OS Interface).....	8
Utilidades (Utilities).....	8

Código de Prueba (Test Code).....	8
Características y Funcionalidades.....	8
Usos sugeridos para SQLite.....	9
Ventajas de SQLite.....	10
Desventajas de SQLite.....	11
Comparación contra otros tipos de bases de datos.....	12
Situaciones en las que un RDBMS cliente/servidor puede funcionar mejor.....	12
Lista de verificación para elegir el motor de base de datos adecuado.....	13
Cuando conviene usarla y cuándo no.....	14
Modo de licenciamiento - Proveedor que facilita el producto.....	15
Instalación del motor de la BD.....	17
Pasos.....	17
Años de antigüedad de su creación.....	20
Principales empresas en donde funciona.....	20
Preguntas.....	21
Pregunta 1.....	21
Pregunta 2.....	21
Pregunta 3.....	22
Pregunta 4.....	22
Pregunta 5.....	22
Pregunta 6.....	22
Pregunta 7.....	23

Pregunta 8.....	23
Pregunta 9.....	23
Pregunta 10.....	23
Repositorio de Código.....	24
Links.....	24

## Consignas

La presentación consta del armado de un documento que debe ser entregado una semana antes de la instancia oral, en la que se realizará la exposición del TP (la fecha será estipulada al momento de la asignación del TP).

El trabajo práctico puede realizarse de forma grupal, pero se evaluará a cada participante de forma individual.

Contenido básico de la presentación sobre el tipo de base de datos o tema asignados:

- Descripción de la arquitectura, las características y sus funcionalidades.
- Ventajas y desventajas.
- Comparación contra otros tipos de bases de datos, cuándo conviene usarla y cuándo no.
- Modo de licenciamiento, de ser posible comentar acerca de los precios.
- Mostrar la instalación del motor de la BD siempre que sea posible.
- Años de antigüedad de su creación, principales empresas en donde funciona, proveedor que facilita el producto.

## Introducción

Un sistema de base de datos embebido es un sistema de administración de base de datos (DBMS) que está estrechamente integrado con un software de aplicación; está incrustado en la aplicación.

Son sistemas de bases de datos con diferentes interfaces de programación de aplicaciones (SQL y API nativas propietarias).

## Historia

Los sistemas de bases de datos embebidos han existido desde finales de los años 70. Podemos mencionar: Btrieve, C-tree, Empress, db\_VISTA y dBASE. En el pasado, las bases de datos integradas se usaban para aplicaciones de línea de negocios de computación departamental, principalmente en PC y pequeños sistemas Unix. Estos sistemas eran silos: no compartían ni necesitaban compartir sus datos. En la medida en que se compartieron los datos, era en forma de informes generados por el sistema. Por ejemplo, en un proveedor de sistemas de bases de datos, desarrollaron un sistema para rastrear el movimiento de motores de misiles nucleares (motores de cohetes), que se utilizó para cumplir con el tratado START. Era un sistema basado en PC que, podría generar un informe sobre cualquier motor específico: dónde está, dónde había estado, si se había gastado en una prueba, etc.

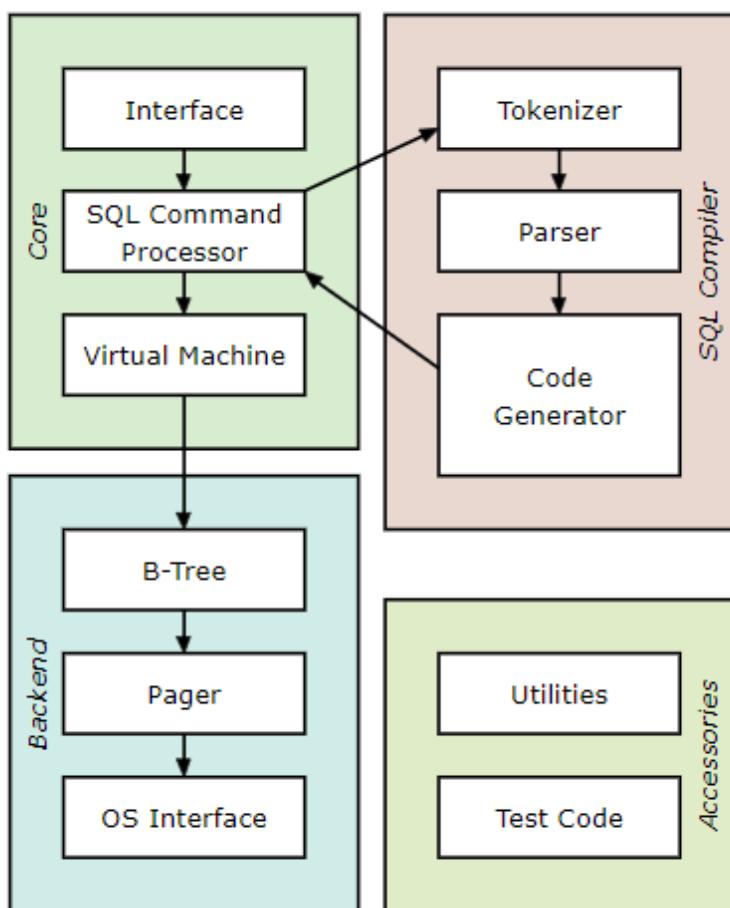
## Acerca de SQLite

La base de datos embebida elegida para el desarrollo de este informe es SQLite ya que es la base de datos embebida más popular del mercado.

SQLite es una biblioteca y un motor de base de datos SQL embebido. A diferencia de la mayoría de las otras bases de datos SQL, SQLite no tiene un proceso de servidor separado. SQLite lee y escribe directamente en archivos de disco ordinarios. Una base

de datos SQL completa con varias tablas, índices, disparadores y vistas está contenida en un solo archivo de disco. El formato de archivo de la base de datos es multiplataforma.

## Descripción de la arquitectura



SQLite funciona compilando texto SQL en bytecode y luego ejecutando ese bytecode usando una máquina virtual.

El mismo cuenta con las siguientes interfaces:

- El sqlite3\_prepare\_v2() y las interfaces relacionadas actúan como un compilador para convertir texto SQL en bytecode.
- El objeto sqlite3\_stmt es un contenedor para un único programa en bytecode que implementa una sola instrucción SQL.
- La interfaz sqlite3\_step() pasa un programa en bytecode a la máquina virtual y ejecuta el programa hasta que este finaliza o forma una cola de resultados para devolver, se produce un error fatal o se interrumpe.

La arquitectura se compone de cuatro módulos: Core, Backend, SQL Compiler y Accessories.

### Interfáz (Interface)

Gran parte de la interfaz en lenguaje C se encuentra en los archivos fuente main.c , legacy.c y vdbeapi.c aunque algunas rutinas están dispersas en otros archivos donde pueden tener acceso a estructuras de datos con alcance de archivo.

Para evitar colisiones de nombres, todos los símbolos externos en la biblioteca SQLite comienzan con el prefijo sqlite3 . Aquellos símbolos que están destinados para uso externo (en otras palabras, aquellos símbolos que forman la API para SQLite) agregan un guión bajo y, por lo tanto, comienzan con sqlite3.

### Tokenizador (tokenizer)

Cuando se va a evaluar una cadena que contiene instrucciones SQL, primero se envía al tokenizador. El tokenizador divide el texto SQL en tokens y entrega esos tokens uno por uno al analizador. El tokenizador está codificado a mano en el archivo tokenize.c.

Tenga en cuenta que en este diseño, el tokenizador llama al analizador sintáctico (parser).

### [Analizador Sintáctico \(Parser\)](#)

El analizador asigna significado a los tokens en función de su contexto. El analizador para SQLite se genera utilizando el generador de analizador Lemon (lemon parser generator).

### [Generador de código \(Code Generator\)](#)

Después de que el analizador ensambla los tokens en un árbol de análisis (parse tree), el generador de código se ejecuta para analizar el árbol de análisis y generar un bytecode que realiza el trabajo de la instrucción SQL. El objeto de declaración preparada (prepared statement object) es un contenedor para este bytecode.

### [Motor Bytecode \(Bytecode Engine\)](#)

El programa de bytecode creado por el generador de código es ejecutado por una máquina virtual.

La máquina virtual en sí está completamente contenida en un único archivo fuente vdbe.c

SQLite implementa funciones SQL utilizando callbacks a rutinas de lenguaje C. Incluso las funciones SQL integradas se implementan de esta manera.

### [Árbol B \(B-Tree\)](#)

Una base de datos SQLite se mantiene en el disco utilizando una implementación de árbol B que se encuentra en el archivo fuente btree.c . Se utilizan árboles B independientes para cada tabla y cada índice de la base de datos. Todos los árboles B se almacenan en el mismo archivo de disco.

## Caché de página (Page Cache)

El módulo B-Tree solicita información del disco en páginas de tamaño fijo. El tamaño de página predeterminado es 4096 bytes, pero puede ser cualquier potencia de dos entre 512 y 65536 bytes. El caché de página es responsable de leer, escribir y almacenar en caché estas páginas. La memoria caché de la página también proporciona la abstracción de rollback y commit atómico y se encarga de bloquear (locking) el archivo de la base de datos. El controlador (driver) de árbol B solicita páginas particulares de la page cache y notifica a la page cache cuando desea modificar páginas o confirmar o revertir cambios. El page cache maneja todos los detalles desordenados para asegurarse de que las solicitudes se manejen de manera rápida, segura y eficiente.

## Interfaz del sistema operativo (OS Interface)

Para proporcionar portabilidad entre sistemas operativos, SQLite usa un objeto abstracto llamado VFS . Cada VFS proporciona métodos para abrir, leer, escribir y cerrar archivos en el disco, y para otras tareas específicas del sistema operativo, como encontrar la hora actual u obtener aleatoriedad para inicializar el generador de números pseudoaleatorios incorporado. SQLite actualmente proporciona VFS para Unix (en el archivo os\_unix.c ) y Windows (en el archivo os\_win.c ).

## Utilidades (Utilities)

La asignación de memoria, las rutinas de comparación de cadenas sin mayúsculas y minúsculas, las rutinas portables de conversión de texto a número y otras utilidades se encuentran en util.c.

## Código de Prueba (Test Code)

Los archivos en el directorio "src/" cuyos nombres comienzan con "test" son únicamente para testing y no están incluidos en el build estándar de la biblioteca.

## Características y Funcionalidades

- Las transacciones son atómicas, consistentes, aisladas y duraderas (ACID) incluso después de fallas del sistema y fallas de energía.
- Configuración cero: no se necesita configuración ni administración.
- Implementación completa de SQL con capacidades avanzadas como: índices parciales, índices en expresiones, JSON, expresiones comunes de tablas y funciones de ventana.
  - Similar a una función de agregación, una función de ventana realiza un cálculo (ej: SUM()) en un conjunto de filas. Sin embargo, una función de ventana no hace que las filas se agrupen en una sola fila de salida.
- Una base de datos completa se almacena en un único archivo de disco multiplataforma. Ideal para usar como formato de archivo de aplicación.
- Admite bases de datos del tamaño de un terabyte y cadenas y blobs del tamaño de un gigabyte.
- Huella de código pequeña: menos de 750 KiB completamente configurado o mucho menos con funciones opcionales omitidas.
- API simple y fácil de usar.
- Rápido: en algunos casos, SQLite es más rápido que la E/S directa del sistema de archivos
- Escrito en ANSI-C. TCL Bindings incluidas. Bindings para docenas de otros idiomas disponibles por separado.
- Código fuente bien comentado con una cobertura de pruebas del 100%.

- Disponible como un solo archivo de código fuente ANSI-C que es fácil de compilar y, por lo tanto, fácil de agregar a un proyecto más grande.
- Autónomo: sin dependencias externas.
- Multiplataforma: Android, \*BSD, iOS, Linux, Mac, Solaris, VxWorks y Windows (Win32, WinCE, WinRT) son compatibles desde el primer momento. Fácil de portar a otros sistemas.
- Los archivos fuente son de dominio público. Se puede utilizar para cualquier propósito.
- Viene con un cliente de interfaz de línea de comandos (CLI) independiente que se puede usar para administrar bases de datos SQLite.

### Usos sugeridos para SQLite

**Base de datos para Internet de las cosas (IoT):** SQLite es una opción popular para el motor de base de datos en teléfonos celulares, PDA, reproductores de MP3, decodificadores y otros dispositivos electrónicos. SQLite tiene una huella de código pequeña, hace un uso eficiente de la memoria, el espacio en disco y el ancho de banda del disco, es altamente confiable y no requiere mantenimiento por parte de un administrador de base de datos.

**Formato de archivo de la aplicación:** En lugar de utilizar fopen() para escribir XML, JSON, CSV o algún formato propietario en archivos de disco utilizados por su aplicación, utilice una base de datos SQLite. Evitará tener que escribir y solucionar problemas de un analizador, sus datos serán más fácilmente accesibles y multiplataforma, y sus actualizaciones serán transaccionales.

**Base de datos de sitios web:** Debido a que no requiere configuración y almacena información en archivos de disco ordinarios, SQLite es una opción popular como base de datos para respaldar sitios web pequeños y medianos.

**Sustituto de un RDBMS empresarial:** SQLite se utiliza a menudo como sustituto de un RDBMS empresarial con fines de demostración o de prueba. SQLite es rápido y no requiere configuración, lo que elimina muchas molestias de las pruebas y hace que las demostraciones sean alegres y fáciles de iniciar.

### Ventajas de SQLite

Las ventajas de utilizar SQLite son:

**Es fácil de usar:** SQLite es muy sencillo de utilizar, ya que no utiliza una comunicación cliente-servidor para las consultas, ya que se comunica con un archivo que es la base de datos y que puede ser autogenerado por la propia aplicación.

**Ideal para el desarrollo de apps móviles:** Sus características lo convierten en una alternativa ideal para el desarrollo de aplicaciones para celulares. Se puede utilizar fácilmente para gestionar bases de datos en app que usen motores como Java, o en proyectos desarrollados con Flutter.

Como la base es un archivo, si se apaga el celular o no hay conexión a internet, el almacenamiento de datos no se ve afectado.

**Utiliza SQL:** Las consultas a la base de datos se realizan en SQL, reduciendo la complejidad del código de la app. SQLite es una versión reducida de SQL que sigue utilizando este estándar, aunque con pequeñas modificaciones, a la hora de realizar consultas a las bases de datos.

**Ocupa poco espacio:** El almacenamiento de una base de datos SQLite se realiza en un solo archivo y tiene una huella de código pequeña (ocupa poco espacio). En comparación con MySQL, SQLite es una alternativa mucho más ligera, por lo que

puede ser utilizada como software integrado en dispositivos como celulares, Smart TV, cámaras...

### Desventajas de SQLite

Las desventajas de utilizar SQLite son:

**No es fácilmente escalable:** No se adapta bien a grandes bases de datos, por lo que si una app comienza a crecer se complica su gestión utilizando SQLite.

**Problemas de seguridad:** Al no contar con funciones de seguridad y administración de usuarios puede presentar problemas en cuanto a seguridad.

**Monousuario para escrituras:** No permite que un usuario modifique datos, si otro se encuentra conectado y realizando acciones sobre la base de datos.

**Limitación de almacenamiento:** El tamaño de la base de datos se encuentra normalmente restringido a un tamaño chico en comparación con otras bases de datos (no es ideal para grandes bases de datos).

### Comparación contra otros tipos de bases de datos

SQLite no es directamente comparable con los motores de base de datos SQL de cliente/servidor como MySQL, Oracle, PostgreSQL o SQL Server, ya que SQLite está tratando de resolver un problema diferente.

Los motores de base de datos SQL cliente/servidor se esfuerzan por implementar un repositorio compartido de datos empresariales. Destacan la escalabilidad, la concurrencia, la centralización y el control. SQLite se esfuerza por proporcionar almacenamiento de datos local para aplicaciones y dispositivos individuales. SQLite enfatiza la economía, la eficiencia, la confiabilidad, la independencia y la simplicidad.

## Situaciones en las que un RDBMS cliente/servidor puede funcionar mejor

**Aplicaciones cliente/servidor:** Si hay muchos programas cliente que envían SQL a la misma base de datos a través de una red, es mejor utilizar un motor de base de datos cliente/servidor en lugar de SQLite. SQLite funcionará en un sistema de archivos de red, pero debido a la latencia asociada con la mayoría de los sistemas de archivos de red, el rendimiento no será muy bueno. Además, la lógica de bloqueo de archivos tiene errores en muchas implementaciones de sistemas de archivos de red (tanto en Unix como en Windows). Si el bloqueo de archivos no funciona correctamente, es posible que dos o más clientes intenten modificar la misma parte de la misma base de datos al mismo tiempo, lo que provocaría daños. Debido a que este problema se debe a errores en la implementación del sistema de archivos subyacente, SQLite no puede hacer nada para evitarlo.

Una buena regla general es evitar el uso de SQLite en situaciones en las que se accede a la misma base de datos directamente (sin un servidor de aplicaciones intermedio) y simultáneamente desde muchas computadoras en una red.

**Sitios web de alto volumen:** SQLite normalmente funcionará bien como base de datos de un sitio web. Pero si el sitio web requiere mucha escritura o está tan ocupado que requiere varios servidores, hay que considerar el uso de un motor de base de datos cliente/servidor de clase empresarial en lugar de SQLite.

**Conjuntos de datos muy grandes:** Una base de datos SQLite tiene un tamaño limitado. Incluso si pudiera manejar bases de datos más grandes, SQLite almacena la base de datos completa en un solo archivo de disco y muchos sistemas de archivos

limitan el tamaño máximo de los archivos a algo menos que esto. Entonces, si está contemplando bases de datos de esta magnitud, haría bien en considerar el uso de un motor de base de datos de cliente/servidor que distribuye su contenido en varios archivos de disco y quizás en varios volúmenes.

**Alta concurrencia:** SQLite admite una cantidad ilimitada de lectores simultáneos, pero solo permitirá un escritor en cualquier momento. Para muchas situaciones, esto no es un problema. Los escritores hacen cola. Cada aplicación hace su trabajo de base de datos rápidamente y continúa, y ningún bloqueo dura más de unos pocos milisegundos. Pero hay algunas aplicaciones que requieren más simultaneidad, y es posible que esas aplicaciones deban buscar una solución diferente.

[Lista de verificación para elegir el motor de base de datos adecuado](#)

### **¿Los datos están separados de la aplicación por una red?**

Los motores de bases de datos relacionales actúan como filtros de datos que reducen el ancho de banda. Por lo tanto, es mejor mantener el motor de base de datos y los datos en el mismo dispositivo físico para que el enlace de motor a disco de ancho de banda alto no tenga que atravesar la red, solo el enlace de aplicación a motor de ancho de banda más bajo.

### **¿Muchos escritores concurrentes?**

Si muchos subprocesos y/o procesos necesitan escribir la base de datos en el mismo instante (y no pueden hacer cola y tomar turnos), entonces es mejor seleccionar un motor de base de datos que admita esa capacidad, lo que siempre significa un motor de base de datos cliente/servidor.

## ¿Grandes datos?

Si sus datos crecerán a un tamaño que no le resulte cómodo o que no pueda caber en un solo archivo de disco, entonces debe seleccionar una solución que no sea SQLite. SQLite admite bases de datos de hasta 281 terabytes de tamaño, suponiendo que pueda encontrar una unidad de disco y un sistema de archivos que admita archivos de 281 terabytes. Aun así, cuando parece que el tamaño del contenido podría llegar al rango de terabytes, sería bueno considerar una base de datos cliente/servidor centralizada.

En caso de que la respuesta a estas tres preguntas sea No, entonces la utilización de una base de datos embebida como SQLite es una muy buena opción.

## Cuando conviene usarla y cuándo no

Las bases de datos embebidas no corren un servicio independiente a la aplicación. Es decir son dependientes de la aplicación, por ejemplo, la podemos colocar como un fichero en la carpeta de la aplicación logrando así que corra dentro de un proceso, siendo únicamente accesible por la aplicación.

Se usará una db embebida cuando sea necesario contar con portabilidad, cuando hayan recursos de memoria reducidos o rapidez.

## Modo de licenciamiento - Proveedor que facilita el producto

Hay varias organizaciones que ofrecen base de datos embebidas:

- Advantage Database Server de Sybase Inc
- Berkeley DB de Oracle Corporation
- CSQL de csqlcache.com

- Extensible Storage Engine de Microsoft
- Firebird Embedded
- HSQLDB de HSQLDB.ORG
- HarperDB
- Informix Dynamic Server (IDS) de IBM
- InfinityDB de Boiler Bay Inc
- InnoDB de Oracle Corporation
- InterBase de Embarcadero Technologies
- RDM Database Manager de Raima
- solidDB
- SQLite
- SQL Server Compact de Microsoft Corporation
- Sophia Embeddable

Pasamos a detallar algunas de estas soluciones:

### **SQLite**

El motor de base de datos elegido es "Open-Source" y "not Open-Contributed".

Esto quiere decir que es posible realizar tantas copias como se requieran pero no se encuentra abierto a que usuarios externos a Hwaci y que no hayan presentado una declaración jurada dedicando su contribución al dominio público incluyan parches.

Por lo que no es requerida una licencia. Sin embargo, algunas organizaciones necesitarán una prueba legal del derecho a utilizar SQLite.

Estos casos pueden ser por:

- Reclamos de indemnización frente a reclamos por derecho de autor.
- Se utiliza SQLite en una jurisdicción que no reconoce el dominio público.

- Se desea tener un documento legal tangible como evidencia de que se posee el derecho legal de usar y distribuir SQLite.
- El departamento legal de la compañía requiere que se encuentre licenciado

Precio: USD 6000

## Berkeley DB

DB clave/valor embebida

				9,800	2,156.00
	Per Wireless Handset	Software Update License & Support	Processor License	Software Update License & Support	
Berkeley DB - Transactional Data Store	6	1.32	5,800	1,276.00	
Berkeley DB - Concurrent Data Store	6	1.32	1,800	396.00	
Berkeley DB - Data Store	6	1.32	900	198.00	
	Named User Plus	Software Update License & Support	Processor License	Software Update License & Support	
Berkeley DB - Transactional Data Store	-	-	5,800	1,276.00	
Berkeley DB - Concurrent Data Store	-	-	1,800	396.00	
Berkeley DB - Data Store	-	-	900	198.00	
Berkeley DB Java Edition - High Availability	-	-	9,800	2,156.00	
Berkeley DB Java Edition - Transactional Data Store	-	-	5,800	1,276.00	
Berkeley DB Java Edition - Concurrent Data Store	-	-	1,800	396.00	
Berkeley DB XML - High Availability	-	-	13,800	3,036.00	
Berkeley DB XML - Transactional Data Store	-	-	8,100	1,782.00	
Berkeley DB XML - Concurrent Data Store	-	-	2,600	572.00	
Berkeley DB XML - Data Store	-	-	1,800	396.00	

## Firebird:

Los módulos de Firebird se publican bajo la licencia pública de desarrollador. InterBase fue lanzado por Inprise bajo la licencia Pública de InterBase.

Ambas son variantes de la Licencia Pública de Mozilla V.1.1 (MPL).

La documentación del Proyecto Firebird se publica bajo la Licencia de Documentación Pública a menos que se indique lo contrario.

## Instalación del motor de la BD

Como se estuvo detallando anteriormente estos tipos de DB no poseen una instalación en sí. Funcionan desde un archivo. SQLite crea un archivo y dentro de este se guardan los datos.

### Pasos

1. Ingresar a: <https://sqlite.org/download.html>
2. Descargar el file: sqlite-tools-win32-x86-3410200.zip

#### SQLite Download Page

##### Source Code

<a href="#">sqlite-amalgamation-3410200.zip</a>	C source code as an amalgamation, version 3.41.2. (2.50 MB)	(SHA3-256: c51ca72411b8453c64e0980be23bc995330bdc3ec1513e00fb022ed0f02463)
<a href="#">sqlite-autoconf-3410200.tar.gz</a>	C source code as an amalgamation. Also includes a "configure" script and <a href="#">TEA</a> makefiles for the <a href="#">TCL Interface</a> . (2.98 MB)	(SHA3-256: 1ebb5539ddfe9a0fb9eab765af0bf02010fc6bafe985b54781a38c00020a)

##### Documentation

<a href="#">sqlite-doc-3410200.zip</a>	Documentation as a bundle of static HTML files. (10.14 MB)	(SHA3-256: 8b3daa86d41cbc407e0992e11193d81094f01771c2dbeae93695f3dbaf19cbfe)
--	---	--

##### Precompiled Binaries for Android

<a href="#">sqlite-android-3410200.aar</a>	A precompiled Android library containing the core SQLite together with appropriate Java bindings, ready to drop into any Android Studio project. (3.26 MB)	(SHA3-256: 6cd0ca943868978a945d4521bdcc4e21fee8e3a7650e577f052550a37bc62db6e)
--	---	---

##### Precompiled Binaries for Linux

<a href="#">sqlite-tools-linux-x86-3410200.zip</a>	A bundle of command-line tools for managing SQLite database files, including the <a href="#">command-line shell</a> program, the <a href="#">sqlcipher</a> program, and the <a href="#">sqlite3_analyzer</a> program. (2.16 MB)	(SHA3-256: c41d2d97f77bcd06c5bd523ce9a0d1ea9f4e22ee649f7a677a1c43f3fd65ac3272c)
--	--	---

##### Precompiled Binaries for Mac OS X (x86)

<a href="#">sqlite-tools-osx-x86-3410200.zip</a>	A bundle of command-line tools for managing SQLite database files, including the <a href="#">command-line shell</a> program, the <a href="#">sqlcipher</a> program, and the <a href="#">sqlite3_analyzer</a> program. (1.54 MB)	(SHA3-256: 0cf1eaeaddff7bcd06c5bd523ce9a15da192e5fc316cb29)
--	--	---

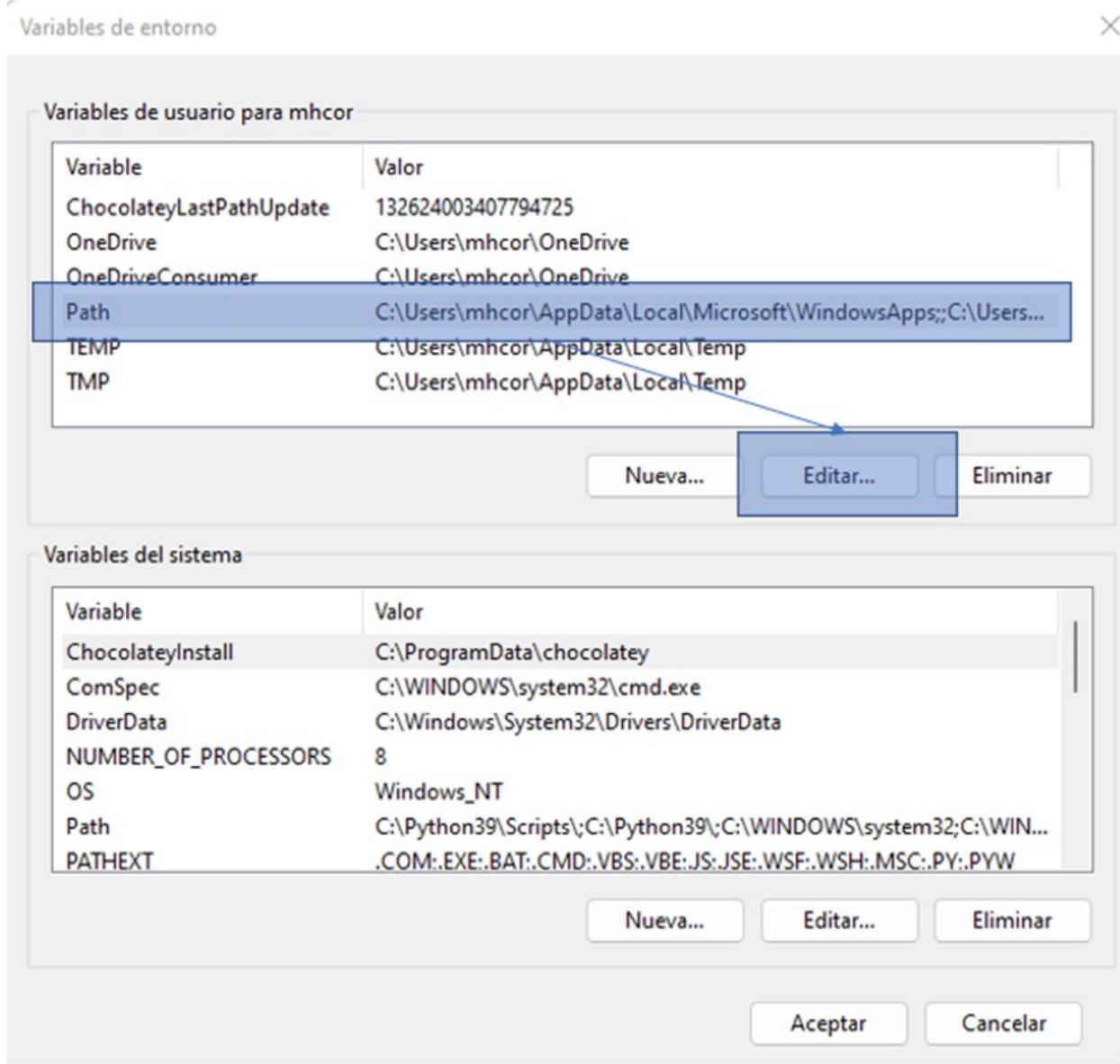
##### Precompiled Binaries for Windows

<a href="#">sqlite-dll-win32-x86-3410200.zip</a>	32-bit DLL (x86) for SQLite version 3.41.2. (563.08 KB)	(SHA3-256: 2a3b02395eef7cb172465d9be3990f8ac10fa92b9af22031b67fa42c2325e1)
<a href="#">sqlite-dll-win64-x64-3410200.zip</a>	64-bit DLL (x64) for SQLite version 3.41.2. (1.15 MB)	(SHA3-256: a85aa4c03f919414704a99050245fb6d5d0b00ae726df259e71c07055e44854cf)
<a href="#">sqlite-tools-win32-x86-3410200.zip</a>	A bundle of command-line tools for managing SQLite database files, including the <a href="#">command-line shell</a> program, the <a href="#">sqlcipher.exe</a> program, and the <a href="#">sqlite3_analyzer.exe</a> program. (1.91 MB)	(SHA3-256: 0ceeb7fb378707d6d6b0737edf2ba02253a3044b1009400f86273719d98f1f)

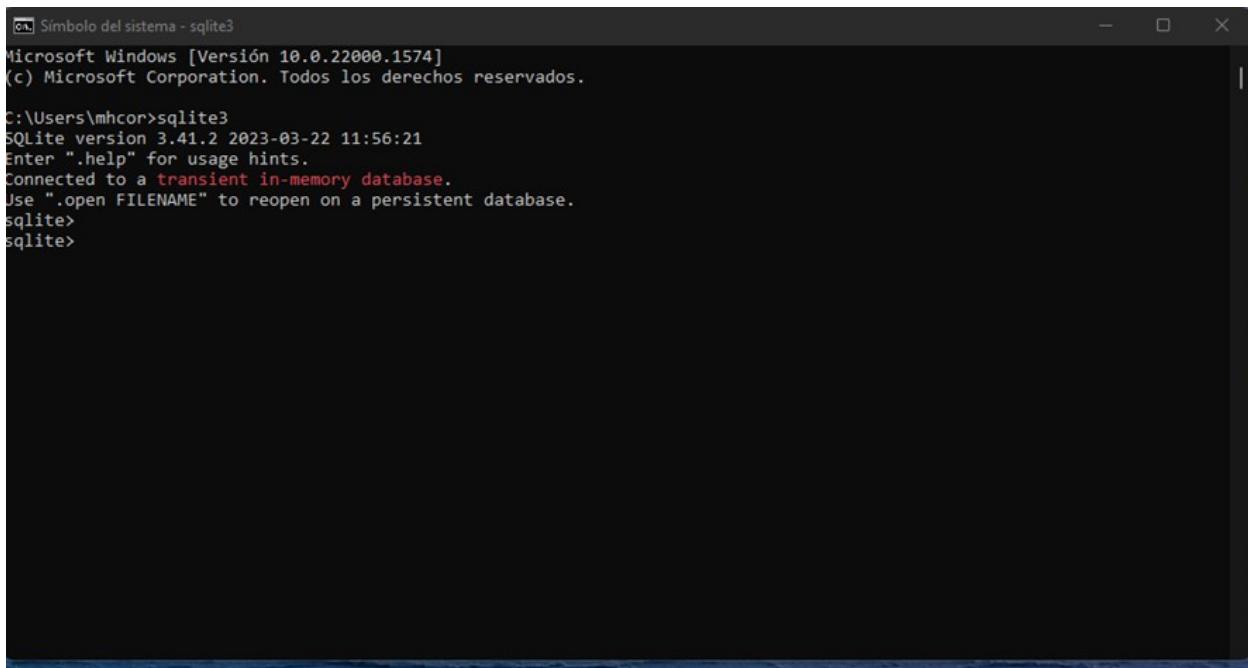
3. Descomprimir el .zip dentro de la raíz del disco del host donde se desea instalar.

Nombre	Fecha de modificación	Tipo	Tamaño
sqldiff.exe	22/3/2023 10:46	Aplicación	574 KB
sqlite3.exe	22/3/2023 10:46	Aplicación	1,107 KB
sqlite3_analyzer.exe	22/3/2023 10:46	Aplicación	2,054 KB

4. Para poder iniciar la DB desde cualquier directorio se debe agregar la variable de entorno. Para esto copiar el path donde se copiaron los archivos. En este caso: C:\DB\sqlite



5. Abrir una ventana de línea de comando y ejecutar “sqlite3”



The screenshot shows a Windows terminal window titled "Símbolo del sistema - sqlite3". The window displays the following text:

```
Microsoft Windows [Versión 10.0.22000.1574]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\mhcor>sqlite3
SQLite version 3.41.2 2023-03-22 11:56:21
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
sqlite>
```

Años de antigüedad de su creación

**SQLite:** Lanzamiento: 17 de agosto de 2000

**Barkeley DB:** Release inicial 1994

**Firebird:** Lanzamiento inicial 2000

Principales empresas en donde funciona

- **Adobe Photoshop Elements** utiliza SQLite como motor de base de datos la última versión del producto (la 6.0)
- **Clementine** usa SQLite para guardar su colección de datos por defecto.
- **Kexi** usa SQLite como un motor de base de datos interno por defecto.
- **Mozilla Firefox** usa SQLite para almacenar, entre otros, las cookies, los favoritos, el historial y las direcciones de red válidas.
- Varias aplicaciones de **Apple** utilizan SQLite, incluyendo Apple Mail y el gestor de RSS que se distribuye con Mac OS X. El software Aperture de Apple guarda

la información de las imágenes en una base de datos SQLite, utilizando la API Core Data.

- El navegador web **Opera** usa SQLite para la gestión de bases de datos WebSQL.
- **Skype** es otra aplicación de gran despliegue que utiliza SQLite.
- **XBMC Media Center** hoy llamado Kodi (antes conocido como "XBox Media Center") es un reproductor de medios de audio, video, fotos, etc de código libre (open source) multi-plataforma a la vez que un centro de entretenimiento. Usa SQLite para administrar las librerías de música, video y fotografías, listas de reproducción y bookmarks entre otras utilidades menores.
- Las empresas **Airbus, Amazon, AOL, BMC, Cisco System, eBay** utilizan Barkeley DB. Otras como **Bas-X** (empresa de telecomunicaciones) o **Watermark Technologies** (empresa que ofrece servicios financieros) utilizan Firebird.

## Preguntas

### Pregunta 1

Las bases de datos embebidas aparecieron en el mercado a principios del año 2000.

¿Verdadero o Falso?

- A. Verdadero
- B. Falso

Opción Correcta: B

### Pregunta 2

Un sistema de base de datos embebido...

- A. Es un sistema de administración de base de datos (DBMS) que está estrechamente integrado con un software de aplicación.
- B. Está incrustado en la aplicación.
- C. Todas las anteriores.

Opción Correcta: C

### Pregunta 3

Una base de datos SQL completa con varias tablas, índices, disparadores y vistas está contenida en un solo archivo de disco. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso

Opción Correcta: A

### Pregunta 4

El módulo core de la arquitectura de SQLite está compuesto por:

- A. Interface, Tokenizer, Parser
- B. Interface, Sql Command Processor, Virtual Machine
- C. B-Tree, Utilities, Test Code

Opción Correcta: B

### Pregunta 5

¿En cuál de las siguientes situaciones conviene usar SQLite?

- A. Base de datos para internet de las cosas
- B. Base de datos de Sitios Web pequeños
- C. Sustituto de RDBMS para pruebas
- D. Todas las anteriores.

Opción Correcta: D

### Pregunta 6

¿Cuál de las siguientes opciones NO es una desventaja de SQLite?

- A. Ocupa poco espacio
- B. No es fácilmente escalable
- C. Problemas de Seguridad

Opción Correcta: A

### Pregunta 7

Si tenemos una situación donde los datos están separados de la aplicación por una red. Es mejor hacer uso de:

- A. RDBMS Cliente/Servidor
- B. Base de datos embebida

Opción Correcta: A

### Pregunta 8

¿Cuáles de las siguientes organizaciones ofrecen base de datos embebidas?

- A. Advantage Database Server de Sybase Inc
- B. Berkeley DB de Oracle Corporation
- C. SQLite
- D. Firebird Embedded
- E. Todas las anteriores

Opción Correcta: E

### Pregunta 9

Mozilla Firefox usa SQLite para almacenar, entre otros, las cookies, los favoritos, el historial y las direcciones de red válidas. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso

Opción Correcta: A

### Pregunta 10

SQLite no hace uso del lenguaje SQL. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso

Opción Correcta: B

## Repositorio de Código

- [https://github.com/andresbiso/laboratorio\\_IV\\_bd\\_embebida](https://github.com/andresbiso/laboratorio_IV_bd_embebida)

## Links

- <https://www.sqlite.org/arch.html>
- <https://www.sqlite.org/copyright.html>
- <https://www.sqlite.org/prosupport.html>
- <https://sqlite.org/about.html>
- <https://www.sqlite.org/features.html>
- <https://www.sqlite.org/whentouse.html>
- <https://dzone.com/articles/how-sqlite-database-works#:~:text=SQLite%20database%20architecture%20split%20into,to%20access%20the%20file%20system.>
- <https://hostingplus.ar/blog/sqlite-ventajas-y-desventajas/>
- <https://www.quora.com/What-are-the-pros-and-cons-of-using-an-SQLite-database>
- <https://www.javatpoint.com/sqlite-advantages-and-disadvantages>
- [https://www.researchgate.net/publication/334067445\\_BASES\\_DE\\_DATOS\\_INCRUSTADAS\\_EMPOTRADAS\\_O\\_EMBEBIDAS](https://www.researchgate.net/publication/334067445_BASES_DE_DATOS_INCRUSTADAS_EMPOTRADAS_O_EMBEBIDAS)
- <https://www.getapp.es/software/125791/sqlite#:~:text=P.,Prueba%20gratis%3A%20No%20disponible>
- <https://www.oracle.com/assets/technology-price-list-070617.pdf>
- <https://circuitcellar.com/insights/the-future-of-embedded-databases/>

- <https://thenewstack.io/the-origin-story-of-sqlite-the-worlds-most-widely-used-database-software/>
- <https://stackoverflow.com/questions/24575080/what-is-the-difference-between-embedded-database-and-ordinary-database-like-mysq>

# Base de Datos Embebida

Laboratorio IV

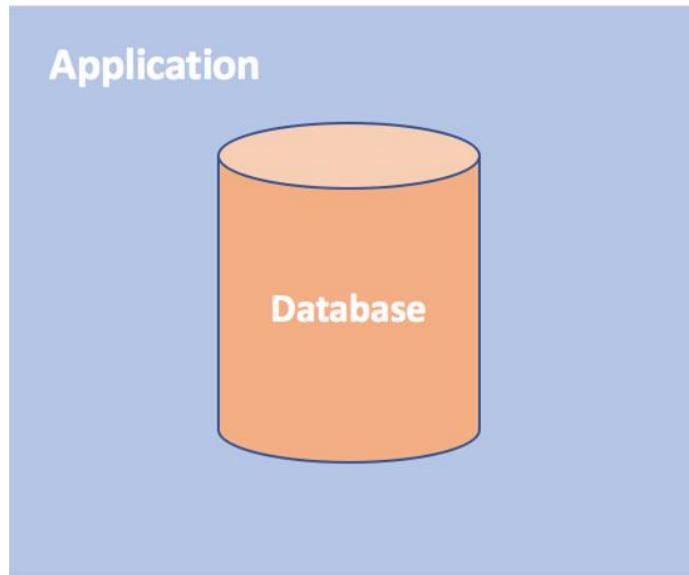


# Integrantes

Andrés Isaac Biso - Legajo 0125044

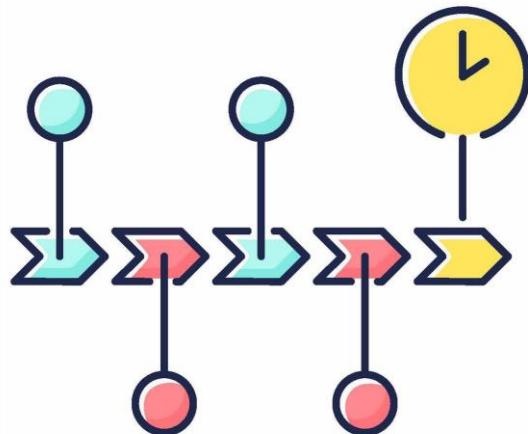
Matias Hernan Coria - Legajo 0127111

# Introducción



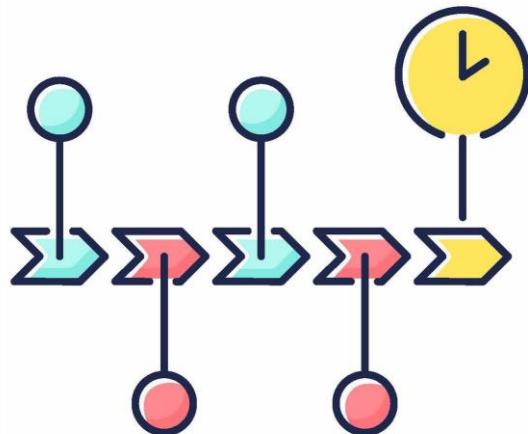
- Un sistema de base de datos embebido es un sistema de administración de base de datos (DBMS) que está estrechamente integrado con un software de aplicación; está incrustado en la aplicación.
- Son sistemas de bases de datos con diferentes interfaces de programación de aplicaciones (SQL y API nativas propietarias).

# Historia



- Los sistemas de bases de datos embebidos han existido desde finales de los años 70.
- Podemos mencionar: Btrieve, C-tree, Empress, db\_VISTA y dBASE.
- En el pasado, las bases de datos integradas se usaban para aplicaciones de línea de negocios de computación departamental, principalmente en PC y pequeños sistemas Unix. Estos sistemas eran silos: no compartían ni necesitaban compartir sus datos.

# Historia



En la medida en que se compartieron los datos, era en forma de informes generados por el sistema. Por ejemplo, en un proveedor de sistemas de bases de datos, desarrollaron un sistema para rastrear el movimiento de motores de misiles nucleares (motores de cohetes), que se utilizó para cumplir con el tratado START. Era un sistema basado en PC que, podría generar un informe sobre cualquier motor específico: dónde está, dónde había estado, si se había gastado en una prueba, etc.

\*Tratado START: [https://en.wikipedia.org/wiki/START\\_I](https://en.wikipedia.org/wiki/START_I)

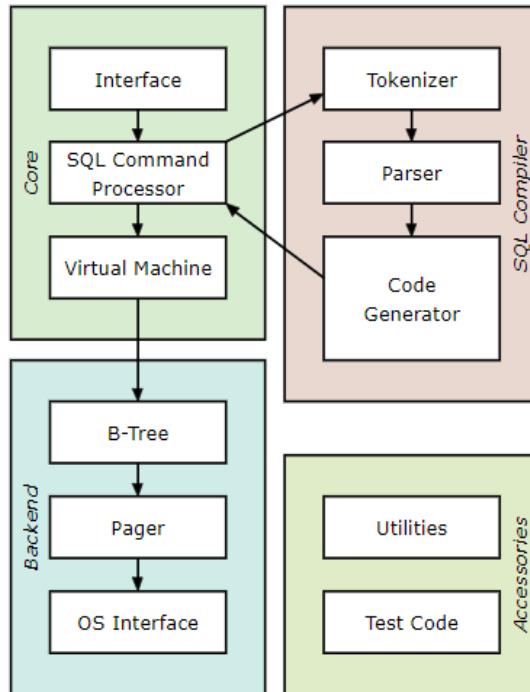


# Acerca de SQLite



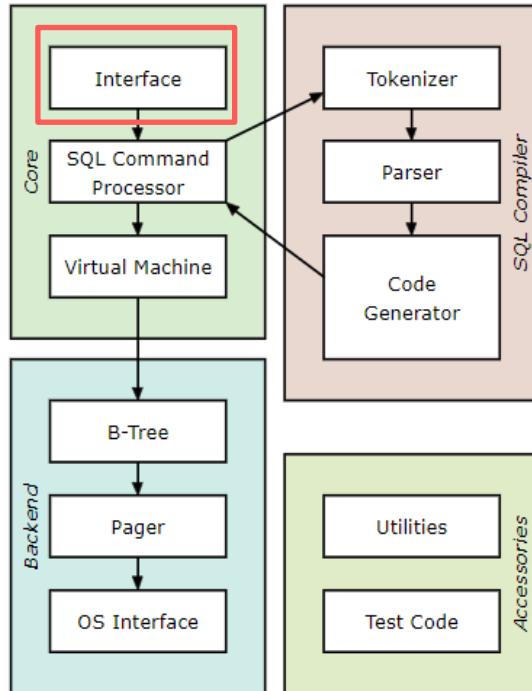
- La base de datos embebida elegida para el desarrollo de este informe es SQLite ya que es la base de datos embebida más popular del mercado.
- SQLite es una biblioteca y un motor de base de datos SQL embebido. A diferencia de la mayoría de las otras bases de datos SQL, SQLite no tiene un proceso de servidor separado. SQLite lee y escribe directamente en archivos de disco ordinarios. Una base de datos SQL completa con varias tablas, índices, disparadores y vistas está contenida en un solo archivo de disco. El formato de archivo de la base de datos es multiplataforma.

# Descripción de la Arquitectura



- SQLite funciona compilando texto SQL en bytecode y luego ejecutando ese bytecode usando una máquina virtual.
- El mismo cuenta con las siguientes interfaces:
  - El `sqlite3_prepare_v2()` y las interfaces relacionadas actúan como un compilador para convertir texto SQL en bytecode.
  - El objeto `sqlite3_stmt` es un contenedor para un único programa en bytecode que implementa una sola instrucción SQL.
  - La interfaz `sqlite3_step()` pasa un programa en bytecode a la máquina virtual y ejecuta el programa hasta que este finaliza o forma una cola de resultados para devolver, se produce un error fatal o se interrumpe.
- La arquitectura se compone de cuatro módulos: Core, Backend, SQL Compiler y Accessories.

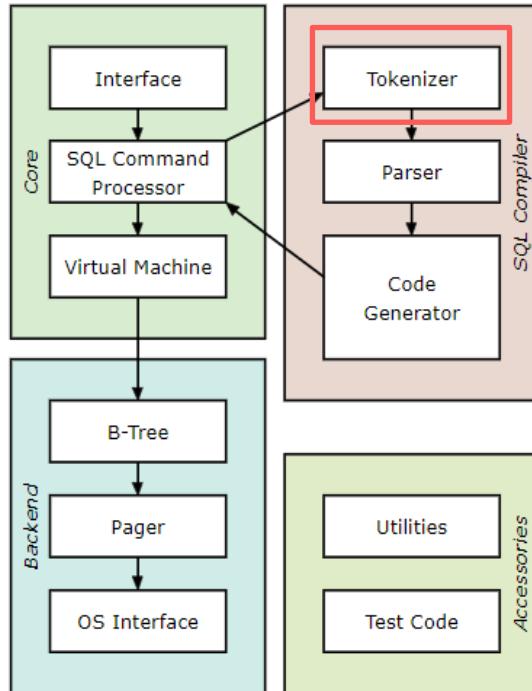
# Descripción de la Arquitectura



## Interfáz (Interface)

- Gran parte de la interfaz en lenguaje C se encuentra en los archivos fuente main.c , legacy.c y vdbeapi.c aunque algunas rutinas están dispersas en otros archivos donde pueden tener acceso a estructuras de datos con alcance de archivo.
- Para evitar colisiones de nombres, todos los símbolos externos en la biblioteca SQLite comienzan con el prefijo sqlite3 . Aquellos símbolos que están destinados para uso externo (en otras palabras, aquellos símbolos que forman la API para SQLite) agregan un guión bajo y, por lo tanto, comienzan con sqlite3.

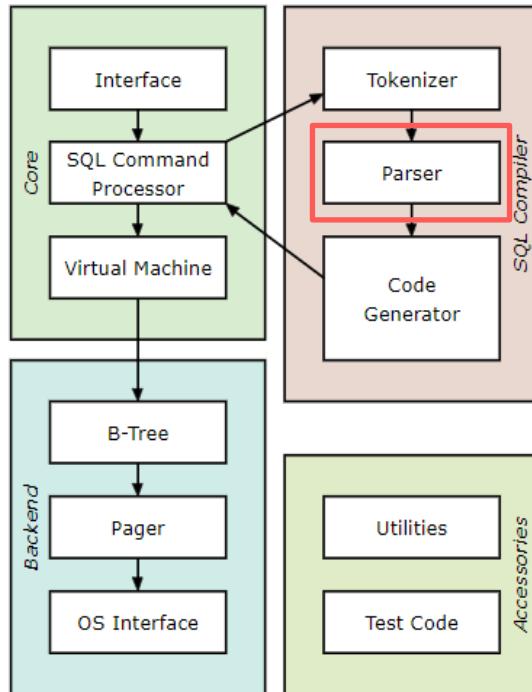
# Descripción de la Arquitectura



## Tokenizador (Tokenizer)

- Cuando se va a evaluar una cadena que contiene instrucciones SQL, primero se envía al tokenizador. El tokenizador divide el texto SQL en tokens y entrega esos tokens uno por uno al analizador. El tokenizador está codificado a mano en el archivo tokenize.c.
- Tenga en cuenta que en este diseño, el tokenizador llama al analizador sintáctico (parser).

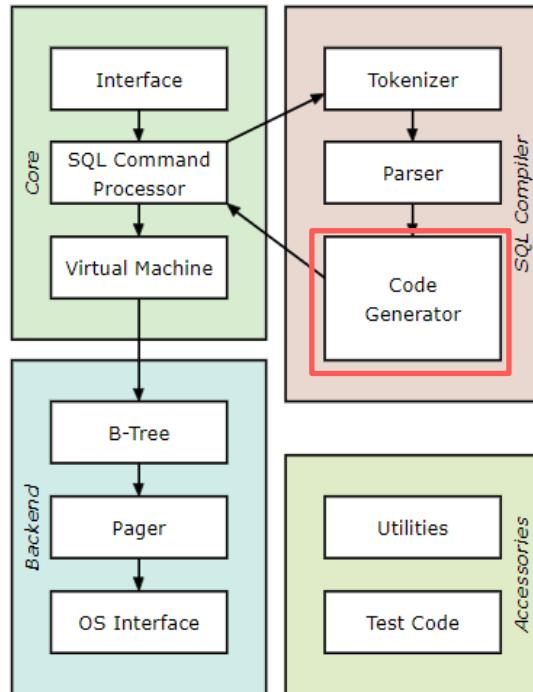
# Descripción de la Arquitectura



## Analizador Sintáctico (Parser)

- El analizador asigna significado a los tokens en función de su contexto. El analizador para SQLite se genera utilizando el generador de analizador Lemon (lemon parser generator).

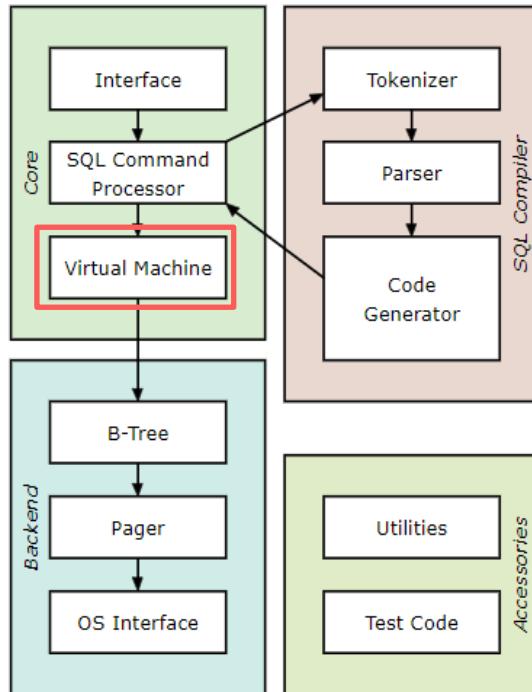
# Descripción de la Arquitectura



## Generador de Código (Code Generator)

- Después de que el analizador ensambla los tokens en un árbol de análisis (parse tree), el generador de código se ejecuta para analizar el árbol de análisis y generar un bytecode que realiza el trabajo de la instrucción SQL. El objeto de declaración preparada (prepared statement object) es un contenedor para este bytecode.

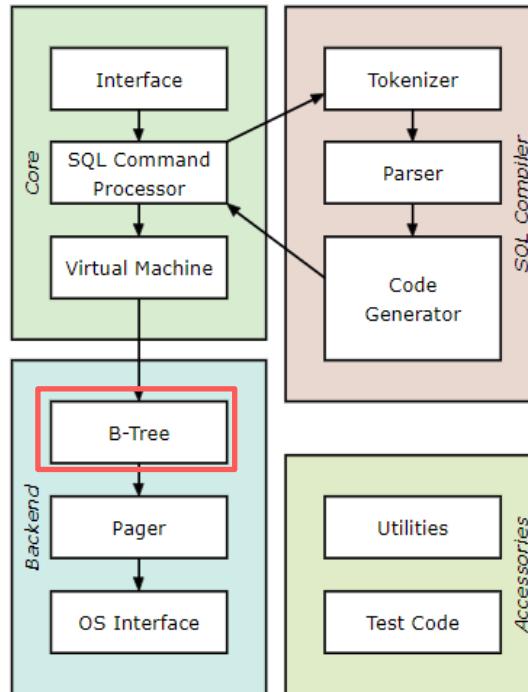
# Descripción de la Arquitectura



## Motor de Bytecode (Bytecode Engine)

- Después de que el analizador ensambla los tokens en un árbol de análisis (parse tree), el generador de código se ejecuta para analizar el árbol de análisis y generar un bytecode que realiza el trabajo de la instrucción SQL. El objeto de declaración preparada (prepared statement object) es un contenedor para este bytecode.

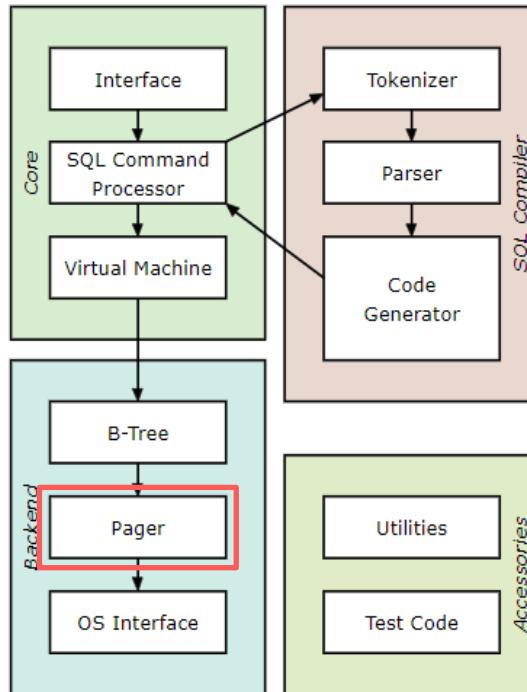
# Descripción de la Arquitectura



## Árbol-B (B-Tree)

- Una base de datos SQLite se mantiene en el disco utilizando una implementación de árbol B que se encuentra en el archivo fuente `btree.c`. Se utilizan árboles B independientes para cada tabla y cada índice de la base de datos. Todos los árboles B se almacenan en el mismo archivo de disco.

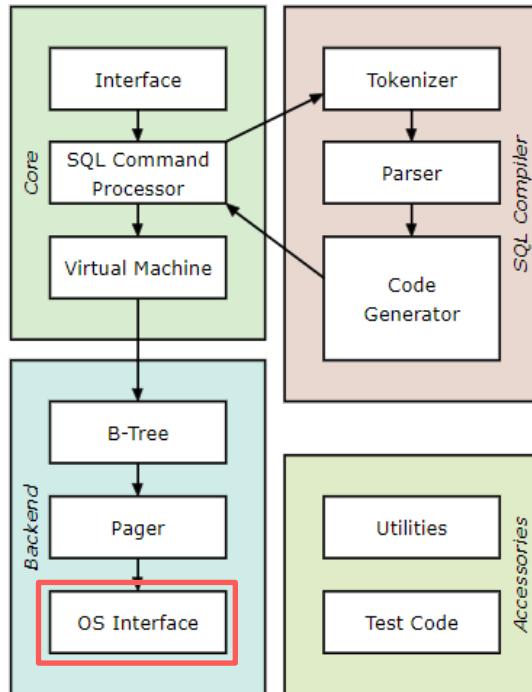
# Descripción de la Arquitectura



## Caché de Página (Page Cache)

- El módulo B-Tree solicita información del disco en páginas de tamaño fijo. El tamaño de página predeterminado es 4096 bytes, pero puede ser cualquier potencia de dos entre 512 y 65536 bytes. El caché de página es responsable de leer, escribir y almacenar en caché estas páginas. La memoria caché de la página también proporciona la abstracción de rollback y commit atómico y se encarga de bloquear (locking) el archivo de la base de datos. El controlador (driver) de árbol B solicita páginas particulares de la page cache y notifica a la page cache cuando desea modificar páginas o confirmar o revertir cambios. El page cache maneja todos los detalles desordenados para asegurarse de que las solicitudes se manejen de manera rápida, segura y eficiente.

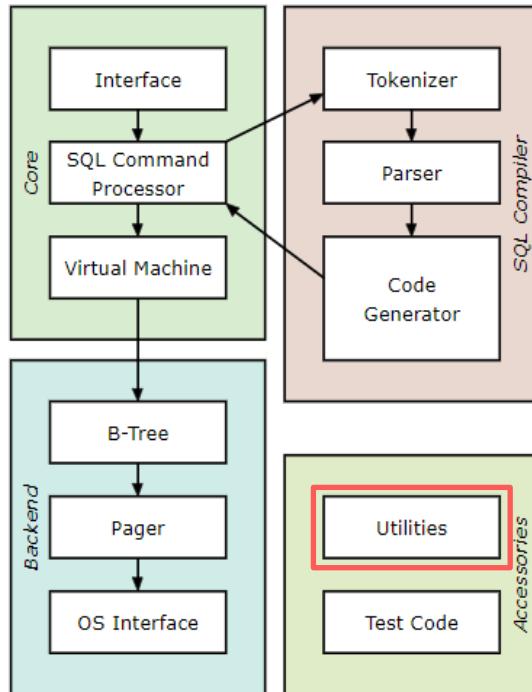
# Descripción de la Arquitectura



## Interfaz con el Sistema Operativo (OS Interface)

- Para proporcionar portabilidad entre sistemas operativos, SQLite usa un objeto abstracto llamado VFS. Cada VFS proporciona métodos para abrir, leer, escribir y cerrar archivos en el disco, y para otras tareas específicas del sistema operativo, como encontrar la hora actual u obtener aleatoriedad para inicializar el generador de números pseudoaleatorios incorporado. SQLite actualmente proporciona VFS para Unix (en el archivo `os_unix.c`) y Windows (en el archivo `os_win.c`).

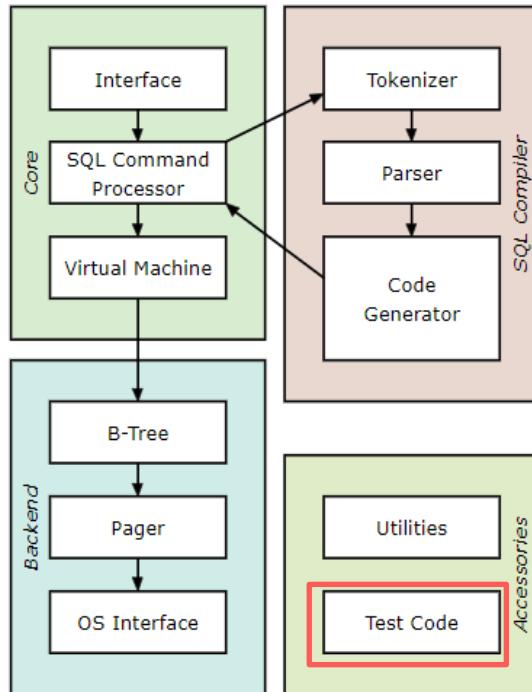
# Descripción de la Arquitectura



## Utilidades (Utilities)

- La asignación de memoria, las rutinas de comparación de cadenas sin mayúsculas y minúsculas, las rutinas portables de conversión de texto a número y otras utilidades se encuentran en util.c.

# Descripción de la Arquitectura

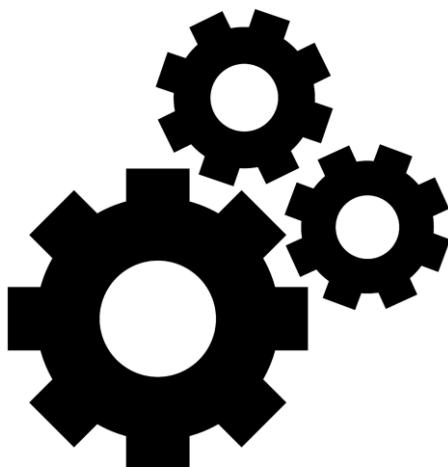


## Código de Prueba (Test Code)

- Los archivos en el directorio "src/" cuyos nombres comienzan con “test” son únicamente para testing y no están incluídos en el build estándar de la biblioteca.



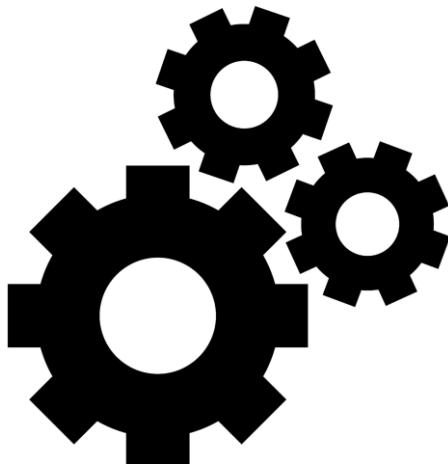
# Características y Funcionalidades



- Las transacciones son atómicas, consistentes, aisladas y duraderas (ACID) incluso después de fallas del sistema y fallas de energía.
- Configuración cero: no se necesita configuración ni administración.
- Implementación completa de SQL con capacidades avanzadas como: índices parciales, índices en expresiones, JSON, expresiones comunes de tablas y funciones de ventana.
  - Similar a una función de agregación, una función de ventana realiza un cálculo (ej: SUM()) en un conjunto de filas. Sin embargo, una función de ventana no hace que las filas se agrupen en una sola fila de salida.
- Una base de datos completa se almacena en un único archivo de disco multiplataforma. Ideal para usar como formato de archivo de aplicación.



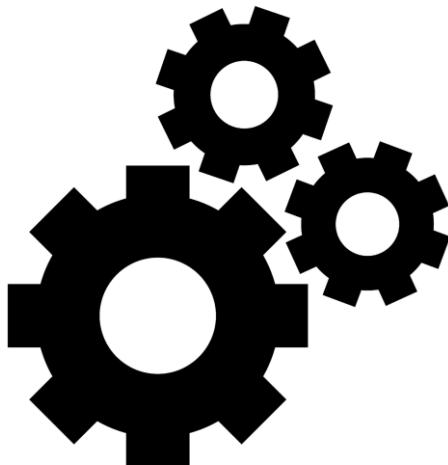
# Características y Funcionalidades



- Admite bases de datos del tamaño de un terabyte y cadenas y blobs del tamaño de un gigabyte.
- Huella de código pequeña: menos de 750 KiB completamente configurado o mucho menos con funciones opcionales omitidas.
- API simple y fácil de usar.
- Rápido: en algunos casos, SQLite es más rápido que la E/S directa del sistema de archivos
- Escrito en ANSI-C. TCL Bindings incluidas. Bindings para docenas de otros idiomas disponibles por separado.
- Código fuente bien comentado con una cobertura de pruebas del 100%.
- Disponible como un solo archivo de código fuente ANSI-C que es fácil de compilar y, por lo tanto, fácil de agregar a un proyecto más grande.



# Características y Funcionalidades



- Autónomo: sin dependencias externas.
- Multiplataforma: Android, \*BSD, iOS, Linux, Mac, Solaris, VxWorks y Windows (Win32, WinCE, WinRT) son compatibles desde el primer momento. Fácil de portar a otros sistemas.
- Los archivos fuente son de dominio público. Se puede utilizar para cualquier propósito.
- Viene con un cliente de interfaz de línea de comandos (CLI) independiente que se puede usar para administrar bases de datos SQLite.



# Usos sugeridos de SQLite



## Base de datos para Internet de las cosas (IoT)

SQLite es una opción popular para el motor de base de datos en teléfonos celulares, PDA, reproductores de MP3, decodificadores y otros dispositivos electrónicos. SQLite tiene una huella de código pequeña, hace un uso eficiente de la memoria, el espacio en disco y el ancho de banda del disco, es altamente confiable y no requiere mantenimiento por parte de un administrador de base de datos.



# Usos sugeridos de SQLite



## Formato de Archivo de la Aplicación

En lugar de utilizar fopen() para escribir XML, JSON, CSV o algún formato propietario en archivos de disco utilizados por su aplicación, utilice una base de datos SQLite. Evitará tener que escribir y solucionar problemas de un analizador, sus datos serán más fácilmente accesibles y multiplataforma, y sus actualizaciones serán transaccionales.



# Usos sugeridos de SQLite



## Base de Datos de Sitio Web

Debido a que no requiere configuración y almacena información en archivos de disco ordinarios, SQLite es una opción popular como base de datos para respaldar sitios web pequeños y medianos.



# Usos sugeridos de SQLite

## Sustituto de una RDBMS Empresarial



SQLite se utiliza a menudo como sustituto de un RDBMS empresarial con fines de demostración o de prueba. SQLite es rápido y no requiere configuración, lo que elimina muchas molestias de las pruebas y hace que las demostraciones sean alegres y fáciles de iniciar.



# Ventajas de SQLite



## Es fácil de usar

SQLite es muy sencillo de utilizar, ya que no utiliza una comunicación cliente-servidor para las consultas, ya que se comunica con un archivo que es la base de datos y que puede ser autogenerado por la propia aplicación.

## Ideal para el desarrollo de apps móviles

Sus características lo convierten en una alternativa ideal para el desarrollo de aplicaciones para celulares. Se puede utilizar fácilmente para gestionar bases de datos en app que usen motores como Java, o en proyectos desarrollados con Flutter.

Como la base es un archivo, si se apaga el celular o no hay conexión a internet, el almacenamiento de datos no se ve afectado.



# Ventajas de SQLite



## Utiliza SQL

Las consultas a la base de datos se realizan en SQL, reduciendo la complejidad del código de la app. SQLite es una versión reducida de SQL que sigue utilizando este estándar, aunque con pequeñas modificaciones, a la hora de realizar consultas a las bases de datos.

## Ocupa poco espacio

El almacenamiento de una base de datos SQLite se realiza en un solo archivo y tiene una huella de código pequeña (ocupa poco espacio). En comparación con MySQL, SQLite es una alternativa mucho más ligera, por lo que puede ser utilizada como software integrado en dispositivos como celulares, Smart TV, cámaras...



# Desventajas de SQLite



## No es fácilmente escalable

No se adapta bien a grandes bases de datos, por lo que si una app comienza a crecer se complica su gestión utilizando SQLite.

## Problemas de seguridad

Al no contar con funciones de seguridad y administración de usuarios puede presentar problemas en cuanto a seguridad.

## Monousuario para escrituras

No permite que un usuario modifique datos, si otro se encuentra conectado y realizando acciones sobre la base de datos.



# Desventajas de SQLite

## Limitación de almacenamiento



El tamaño de la base de datos se encuentra normalmente restringido a un tamaño chico en comparación con otras bases de datos (no es ideal para grandes bases de datos).



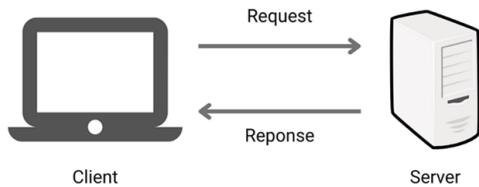
# Comparación con otras bases de datos



- SQLite no es directamente comparable con los motores de base de datos SQL de cliente/servidor como MySQL, Oracle, PostgreSQL o SQL Server, ya que SQLite está tratando de resolver un problema diferente.
- Los motores de base de datos SQL cliente/servidor se esfuerzan por implementar un repositorio compartido de datos empresariales. Destacan la escalabilidad, la concurrencia, la centralización y el control. SQLite se esfuerza por proporcionar almacenamiento de datos local para aplicaciones y dispositivos individuales. SQLite enfatiza la economía, la eficiencia, la confiabilidad, la independencia y la simplicidad.



# Situaciones en las que un RDBMS cliente/servidor funciona mejor



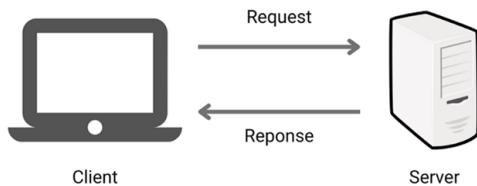
*Una buena regla general es evitar el uso de SQLite en situaciones en las que se accede a la misma base de datos directamente (sin un servidor de aplicaciones intermedio) y simultáneamente desde muchas computadoras en una red.*

## Aplicaciones cliente/servidor

Si hay muchos programas cliente que envían SQL a la misma base de datos a través de una red, es mejor utilizar un motor de base de datos cliente/servidor en lugar de SQLite. SQLite funcionará en un sistema de archivos de red, pero debido a la latencia asociada con la mayoría de los sistemas de archivos de red, el rendimiento no será muy bueno. Además, la lógica de bloqueo de archivos tiene errores en muchas implementaciones de sistemas de archivos de red (tanto en Unix como en Windows). Si el bloqueo de archivos no funciona correctamente, es posible que dos o más clientes intenten modificar la misma parte de la misma base de datos al mismo tiempo, lo que provocaría daños. Debido a que este problema se debe a errores en la implementación del sistema de archivos subyacente, SQLite no puede hacer nada para evitarlo.



## Situaciones en las que un RDBMS cliente/servidor funciona mejor

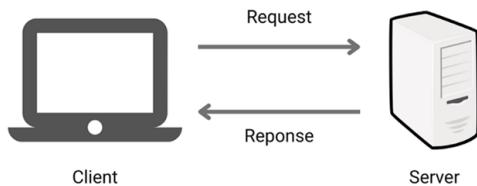


### Sitios web de alto volumen

SQLite normalmente funcionará bien como base de datos de un sitio web. Pero si el sitio web requiere mucha escritura o está tan ocupado que requiere varios servidores, hay que considerar el uso de un motor de base de datos cliente/servidor de clase empresarial en lugar de SQLite.



## Situaciones en las que un RDBMS cliente/servidor funciona mejor

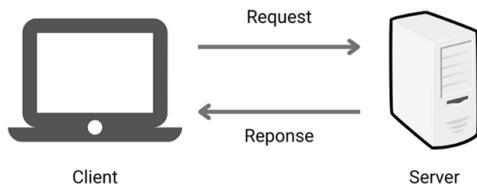


### Conjuntos de datos muy grandes

Una base de datos SQLite tiene un tamaño limitado. Incluso si pudiera manejar bases de datos más grandes, SQLite almacena la base de datos completa en un solo archivo de disco y muchos sistemas de archivos limitan el tamaño máximo de los archivos a algo menos que esto. Entonces, si está contemplando bases de datos de esta magnitud, haría bien en considerar el uso de un motor de base de datos de cliente/servidor que distribuye su contenido en varios archivos de disco y quizás en varios volúmenes.



## Situaciones en las que un RDBMS cliente/servidor funciona mejor



### Alta concurrencia

SQLite admite una cantidad ilimitada de lectores simultáneos, pero solo permitirá un escritor en cualquier momento. Para muchas situaciones, esto no es un problema. Los escritores hacen cola. Cada aplicación hace su trabajo de base de datos rápidamente y continúa, y ningún bloqueo dura más de unos pocos milisegundos. Pero hay algunas aplicaciones que requieren más simultaneidad, y es posible que esas aplicaciones deban buscar una solución diferente.



## Lista de verificación para elegir el motor de base de datos adecuado



**¿Los datos están separados de la aplicación por una red?**

→ elegir cliente/servidor

Los motores de bases de datos relacionales actúan como filtros de datos que reducen el ancho de banda. Por lo tanto, es mejor mantener el motor de base de datos y los datos en el mismo dispositivo físico para que el enlace de motor a disco de ancho de banda alto no tenga que atravesar la red, solo el enlace de aplicación a motor de ancho de banda más bajo.



## Lista de verificación para elegir el motor de base de datos adecuado



### ¿Muchos escritores concurrentes?

→ elegir cliente/servidor

Si muchos subprocessos y/o procesos necesitan escribir la base de datos en el mismo instante (y no pueden hacer cola y tomar turnos), entonces es mejor seleccionar un motor de base de datos que admita esa capacidad, lo que siempre significa un motor de base de datos cliente/servidor.



## Listas de verificación para elegir el motor de base de datos adecuado



### ¿Grandes datos?

→ elegir cliente/servidor

Si sus datos crecerán a un tamaño que no le resulte cómodo o que no pueda caber en un solo archivo de disco, entonces debe seleccionar una solución que no sea SQLite. SQLite admite bases de datos de hasta 281 terabytes de tamaño, suponiendo que pueda encontrar una unidad de disco y un sistema de archivos que admita archivos de 281 terabytes. Aun así, cuando parece que el tamaño del contenido podría llegar al rango de terabytes, sería bueno considerar una base de datos cliente/servidor centralizada.

## **Lista de verificación para elegir el motor de base de datos adecuado**



En caso de que la respuesta a estas tres preguntas sea No, entonces la utilización de una base de datos embebida como SQLite es una muy buena opción.



# ¿Cuándo usar una base de datos embebida?

- **Portabilidad:** La base de datos embebida se encuentra incluida en la aplicación
- **Rapidez:** La base de datos embebida se encuentra en la misma máquina que la aplicación
- **Recursos de hardware bajos:** La base de datos embebida generalmente requiere menos recursos de hardware, como CPU y RAM



# Proveedores

- Advantage Database Server de Sybase Inc
- Berkeley DB de Oracle Corporation
- CSQL de csqlcache.com
- Extensible Storage Engine de Microsoft
- Firebird Embedded
- HSQLDB de HSQLDB.ORG
- HarperDB
- Informix Dynamic Server (IDS) de IBM
- InfinityDB de Boiler Bay Inc
- InnoDB de Oracle Corporation
- InterBase de Embarcadero Technologies
- RDM Database Manager de Raima
- solidDB
- SQLite
- SQL Server Compact de Microsoft Corporation
- Sophia Embeddable



# Licenciamiento

SQLite: Open-Source - not Open-Contributioin sin embargo algunas compañías necesitar prueba legal del uso. Precio: USD 6000

Berkeley DB: Se pueden apreciar los precios en la imagen.

Firebird: Los módulos de Firebird se publican bajo la licencia pública de desarrollador. InterBase fue lanzado por Inprise bajo la licencia Pública de InterBase.



## Berkeley Database

Berkeley DB - High Availability

Berkeley DB - Transactional Data Store  
Berkeley DB - Concurrent Data Store  
Berkeley DB - Data Store

Berkeley DB - Transactional Data Store  
Berkeley DB - Concurrent Data Store  
Berkeley DB - Data Store  
Berkeley DB Java Edition - High Availability  
Berkeley DB Java Edition - Transactional Data Store  
Berkeley DB Java Edition - Concurrent Data Store  
Berkeley DB XML - High Availability  
Berkeley DB XML - Transactional Data Store  
Berkeley DB XML - Concurrent Data Store  
Berkeley DB XML - Data Store

Per Wireless Handset	Software Update License & Support	Processor License	Software Update License & Support
6	1.32	5,800	1,276.00
6	1.32	1,800	396.00
6	1.32	900	198.00
-	-	5,800	1,276.00
-	-	1,800	396.00
-	-	900	198.00
-	-	9,800	2,156.00
-	-	5,800	1,276.00
-	-	1,800	396.00
-	-	900	198.00
-	-	5,800	1,276.00
-	-	1,800	396.00
-	-	9,800	2,156.00
-	-	5,800	1,276.00
-	-	1,800	396.00
-	-	900	198.00
-	-	13,800	3,036.00
-	-	8,100	1,782.00
-	-	2,600	572.00
-	-	1,800	396.00



# Instalación del Motor

- Estos tipos de DB no poseen una instalación en sí.
- SQLite crea un archivo y dentro de este se guardan los datos

Nombre	Fecha de modificación	Tipo	Tamaño
sqldiff.exe	22/3/2023 10:46	Aplicación	574 KB
sqlite3.exe	22/3/2023 10:46	Aplicación	1,107 KB
sqlite3_analyzer.exe	22/3/2023 10:46	Aplicación	2,054 KB

```
C:\Users\mhcor>sqlite3
Microsoft Windows [Versión 10.0.22000.1574]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\mhcor>sqlite3
SQLite version 3.41.2 2023-03-22 11:56:21
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
sqlite>
```

# Instalación del Motor - Cont.

Código ejemplo.

```
1 import sqlite3
2
3 # Crear conexión
4 conn = sqlite3.connect('customer.db')
5
6 # Crear una instancia de cursor
7 c = conn.cursor()
8
9 # Crear una tabla con el metodo execute
10 # Hay 5 tipos de DATATYPE en sqlite: NULL, INTEGER, REAL, TEXT y BLOB
11 c.execute(""" CREATE TABLE customers (
12     first_name text,
13     last_name text,
14     email_address text)
15 """)
16
17 many_customers = [('Wes', 'Brown', 'wes@brown.com'),
18                   ('Steph', 'Kuewa', 'steph@kuewa.com'),
19                   ('Dan', 'Pas', 'dan@pas.com')]
20
21 # Se pone un marcador donde se van a enviar los valores
22 c.executemany("INSERT INTO customers VALUES (?,?,?)", many_customers)
23
24 # Realizar un commit
25 conn.commit()
26
27 # Cerrar la conexión
28 conn.close()
```

The screenshot shows two windows of the SQLite Database Browser. The top window displays the 'Database Structure' tab, specifically the 'Tables' section. It shows a single table named 'customers' with three columns: 'first\_name', 'last\_name', and 'email\_address', all defined as 'text'. Below the table, there are sections for 'Indices', 'Views', and 'Triggers', each with a count of 0. The bottom window shows the 'Browse Data' tab for the 'customers' table. It contains a table with six rows of data, each representing a customer with their first name, last name, and email address.

first_name	last_name	email_address
Jhon	Elder	jhon@test.com
Tim	Elder	tim@test2.com
Maria	Test	mariam@test3.com
Wes	Brown	wes@brown.com
Steph	Kuewa	steph@kuewa.com
Dan	Pas	dan@pas.com

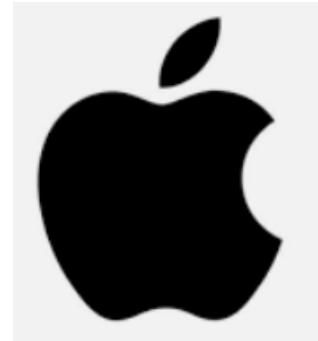


# Años de antigüedad

- SQLite: Lanzamiento: 17 de agosto de 2000
- Barkeley DB: Relase inicial 1994
- Firebird: Lanzamiento inicial 2000



## Empresas





# Demo





# Repositorio - Demo

- [https://github.com/andresbiso/laboratorio\\_IV\\_bd\\_embebida](https://github.com/andresbiso/laboratorio_IV_bd_embebida)



# ¿Preguntas?





# Cuestionario





# Pregunta 1

¿Las bases de datos embebidas aparecieron en el mercado a principios del año 2000?

- A. Verdadero
- B. Falso



# Pregunta 1

Las bases de datos embebidas aparecieron en el mercado a principios del año 2000. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso

Opción Correcta: B



## Pregunta 2

Un sistema de base de datos embebido...

- A. Es un sistema de administración de base de datos (DBMS) que está estrechamente integrado con un software de aplicación.
- B. Está incrustado en la aplicación.
- C. Todas las anteriores.



## Pregunta 2

Un sistema de base de datos embebido...

- A. Es un sistema de administración de base de datos (DBMS) que está estrechamente integrado con un software de aplicación.
- B. Está incrustado en la aplicación.
- C. Todas las anteriores.

Opción Correcta: C



## Pregunta 3

Una base de datos SQL completa con varias tablas, índices, disparadores y vistas está contenida en un solo archivo de disco. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso



## Pregunta 3

Una base de datos SQL completa con varias tablas, índices, disparadores y vistas está contenida en un solo archivo de disco. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso

Opción Correcta: A



## Pregunta 4

El módulo core de la arquitectura de SQLite está compuesto por:

- A. Interface, Tokenizer, Parser
- B. Interface, Sql Command Processor, Virtual Machine
- C. B-Tree, Utilities, Test Code



## Pregunta 4

El módulo core de la arquitectura de SQLite está compuesto por:

- A. Interface, Tokenizer, Parser
- B. Interface, Sql Command Processor, Virtual Machine
- C. B-Tree, Utilities, Test Code

Opción Correcta: B



## Pregunta 5

¿En cuál de las siguientes situaciones conviene usar SQLite?

- A. Base de datos para internet de las cosas
- B. Base de datos de Sitios Web pequeños
- C. Sustituto de RDBMS para pruebas
- D. Todas las anteriores.



## Pregunta 5

¿En cuál de las siguientes situaciones conviene usar SQLite?

- A. Base de datos para internet de las cosas
- B. Base de datos de Sitios Web pequeños
- C. Sustituto de RDBMS para pruebas
- D. Todas las anteriores.

Opción Correcta: D



## Pregunta 6

¿Cuál de las siguientes opciones NO es una desventaja de SQLite?

- A. Ocupa poco espacio
- B. No es fácilmente escalable
- C. Problemas de Seguridad



## Pregunta 6

¿Cuál de las siguientes opciones NO es una desventaja de SQLite?

- A. Ocupa poco espacio
- B. No es fácilmente escalable
- C. Problemas de Seguridad

Opción Correcta: A



## Pregunta 7

Si tenemos una situación donde los datos están separados de la aplicación por una red. Es mejor hacer uso de:

- A. RDBMS Cliente/Servidor
- B. Base de datos embebida



## Pregunta 7

Si tenemos una situación donde los datos están separados de la aplicación por una red. Es mejor hacer uso de:

- A. RDBMS Cliente/Servidor
- B. Base de datos embebida

Opción Correcta: A



## Pregunta 8

¿Cuáles de las siguientes organizaciones ofrecen base de datos embebidas?

- A. Advantage Database Server de Sybase Inc
- B. Berkeley DB de Oracle Corporation
- C. SQLite
- D. Firebird Embedded
- E. Todas las anteriores



## Pregunta 8

¿Cuáles de las siguientes organizaciones ofrecen base de datos embebidas?

- A. Advantage Database Server de Sybase Inc
- B. Berkeley DB de Oracle Corporation
- C. SQLite
- D. Firebird Embedded
- E. Todas las anteriores

Opción Correcta: E



## Pregunta 9

Mozilla Firefox usa SQLite para almacenar, entre otros, las cookies, los favoritos, el historial y las direcciones de red válidas. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso



## Pregunta 9

Mozilla Firefox usa SQLite para almacenar, entre otros, las cookies, los favoritos, el historial y las direcciones de red válidas. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso

Opción Correcta: A



## Pregunta 10

SQLite NO hace uso del lenguaje SQL. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso



## Pregunta 10

SQLite NO hace uso del lenguaje SQL. ¿Verdadero o Falso?

- A. Verdadero
- B. Falso

Opción Correcta: B



# Gracias!

**Base de datos Relacional**

<https://view.genial.ly/6414e9d1bdee470018fa9ec4/guide-bases-de-datos-relacional>



# Base de datos Relacional

EMPEZAR

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas

# Base de datos Relacional

## *Definición*



Una base de datos relacional es un conjunto de datos formado por una o más tablas organizadas en registros y campos relacionados entre sí, los cuales poseen las mismas características.

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas

# Arquitectura

*Estructura de una base de datos relacional*



+INFO

- Diseño Conceptual
  - ① Diseño de una base de datos
  - ② Universo de discurso
  - ③ Identificación de entidades
- Diseño Lógico
  - ① Modelo entidad relación
  - ② Normalización de la base de datos
- Diseño Físico
  - ① Tipos de datos, dominios, índices.
  - ② Creación de la base de datos

[Arquitectura](#)

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas

# Arquitectura

## *Elementos principales de una base de datos relacional*

En una base de datos relacional, cada fila en una tabla es un registro con una ID única. Las columnas contienen los atributos de los datos y cada registro suele tener un valor para cada atributo.

### Relaciones



Relaciones base. Relaciones derivadas.

### Dominios



Enteros. Cadenas de texto. Fecha. No procedurales.

### Restricciones



Determinadas por el usuario. No determinadas por el usuario.

### Claves



Única. Primaria. Foránea. Índice.

### Procedimientos almacenados



Insertar un registro en tabla. Recopilar info.

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas



# Características

*Propiedades de ACID y sistema de gestión de bases de datos relacionales*

## 1 Atomicidad

Elementos que componen una transacción completa de base de datos.

+

## 3 Aislamiento

Efecto de una transacción visible a otros para evitar confusiones.

+

## 2 Uniformidad

Reglas que mantienen los datos en un estado correcto después de una transacción.

+

## 4 Durabilidad

Cambios en los datos sean permanentes cuando la transacción se haya fijado y se llega a un compromiso.

+



Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas



# Funcionalidades

Base de datos relacional

*"Las bases de datos relacionales se usan para rastrear inventarios, procesar transacciones de comercio electrónico, administrar cantidades enormes y esenciales de información de clientes y mucho más."*

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas

# Ventajas del SGDB relacionales

- Sencillez
  - Escasa redundancia de datos
  - Alta consistencia de datos
  - Procesamiento de datos orientado a conjuntos
  - Lenguaje de consultas homogéneo
- Arquitectura
- Características
- Funcionalidades
- Ventajas y Desventajas
- Otras bases de datos
- Gestores o manejadores
- Instalación
- Test de preguntas

# Desventajas del SGDB relacionales

- Presentación de los datos en tablas
- Sistema no jerárquico
- Segmentación de los datos
- Peor rendimiento frente a las bases de datos NOSQL

Arquitectura  
Características  
Funcionalidades  
Ventajas y Desventajas  
Otras bases de datos  
Gestores o manejadores  
Instalación  
Test de preguntas

## Base de datos relacionales Vs. Bases de datos no relacionales

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas



## Base de datos relacionales Vs. Bases de datos orientada a objetos

01

### ORGANIZACIÓN DE DATOS

Diferentes objetos que se componen de archivos e información agrupados y procedimientos

02

### ALMACENAMIENTO DE DATOS

Uso de identificadores y técnicas de indexación

03

### RESTRICCIONES ESTRUCTURALES

Amplia variedad de permisos y menos restricciones

04

### LENGUAJE DE PROGRAMACIÓN

Lenguaje de manipulación que utiliza C, C#, Java y C++.

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas

## Base de datos relacionales Vs. Bases de datos jerarquicas

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas



## Base de datos relacionales Vs. Bases de datos transaccionales

- 
-  **01** **ORGANIZACIÓN DE DATOS**  
No tienen un identificador que sirva para relacionar dos o más datos.
  -  **02** **REDUNDANCIA DE DATOS**  
Redundancia y duplicidad de información.
  -  **03** **ALMACENAMIENTO DE DATOS**  
Mayor capacidad de almacenamiento.

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

**Otras bases de datos**

Gestores o manejadores

Instalación

Test de preguntas

# Gestores o manejadores

## De bases de datos relacionales

ORACLE®

Oracle

El motor relacional comercial más antiguo.



MariaDB

Fork derivado de MySQL a partir de su compra por Oracle.



MySQL

Es el motor mas usado en la web.



SQLite

Es una base de datos embebida en el programa.



PostgreSQL

Inicio como un proyecto universitario.



SQL Server

Desarrollado por Microsoft que incluye mejoras al lenguaje SQL

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas

# Instalacion en MySQL

De bases de datos relacionales



Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas



## ¿Las bases de datos relacionales consisten en un hardware o un software?

Es un hardware que guarda información

Aplicación cliente servidor que sirve para las comunicaciones

Es un hardware que esta sobre un software

Es un software con rutinas especializadas en almacenamiento de información

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

[Test de preguntas](#)

ENVIAR



## ¿Cuáles de ellas son unas de las desventajas de la base de datos relacional?

Puedes seleccionar más de una respuesta

Sistema no jerárquico

Falta de estándares

Gran complejidad

Segmentación de los datos

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

[Test de preguntas](#)

ENVIAR



## Las fases del diseño de una base de datos relacional son:

Diseño conceptual y diseño fisico

Arquitectura

Diseño conceptual, diseño logico y diseño fisico

Características

Diseño conceptual, diseño analogico y diseño fisico

Funcionalidades

Diseño digital, diseño logico y diseño global

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

ENVIAR

Test de preguntas



## ¿Cuáles de ellas son unas de las ventajas de la base de datos relacional?

Puedes seleccionar más de una respuesta

Procesamiento de datos orientado a conjuntos

Ampliabilidad

Escasa redundancia de datos

Lenguaje de consulta más expresivo

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

[Test de preguntas](#)

ENVIAR



**Las bases de datos relacionales tienen un esquema dinámico o flexible, mientras que las bases de datos no relacionales tienen un esquema estricto.**

Verdadero

Falso

ENVIAR

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas



## De los siguientes acronimos, indica cuales son un Sistema Gestor de Bases de Datos Relacionales

Puedes seleccionar más de una respuesta

Base

MySQL

MariaDB

Excel

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

ENVIAR

[Test de preguntas](#)



**Las bases datos orientada a objetos tienen mas restricciones y menos permisos en comparación con las bases de datos relacionales.**

Falso

Verdadero

ENVIAR

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas



**La base de datos jerárquicas contiene datos duplicados a diferencia de las bases de datos que no tienen.**

Falso

Verdadero

ENVIAR

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas



## ¿Cuáles son los principales elementos de una base de datos relacional?

Restricciones, Relaciones, Claves y Dominios

Formularios, Relaciones, Claves, Procedimientos almacenados y Dominios

Restricciones, Relaciones, Claves, Dominios y Procedimientos almacenados

Consultas, Formularios, Informes y Macros

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

[Test de preguntas](#)

ENVIAR



## ¿Cuáles son las características de una base de datos relacional?

Ninguna de las anteriores

Atomicidad, Consistencia, Aislamiento y Durabilidad

Atomicidad, Rendimiento, Deficiencia y Lentitud

Eficiencia, Eficacia, Rendimiento y Escalabilidad

Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

[Test de preguntas](#)

ENVIAR



# ***¡TERMINAMOS!***



Arquitectura

Características

Funcionalidades

Ventajas y Desventajas

Otras bases de datos

Gestores o manejadores

Instalación

Test de preguntas

# ***Trabajo práctico: Bases de datos en la nube***

***Laboratorio IV***



Favio Ontón  
Legajo: 0121231  
4 de mayo de 2023

<b>Introducción: ¿Qué es la nube?.....</b>	<b>3</b>
<b>Descripción, características y funcionalidades.....</b>	<b>3</b>
<b>Ventajas y desventajas.....</b>	<b>4</b>
Ventajas.....	4
Desventajas.....	4
<b>¿Cuándo conviene usar una base de datos en la nube?.....</b>	<b>5</b>
<b>Proveedores, servicios y modos de licenciamiento.....</b>	<b>5</b>
<b>¿Cómo usar una BD en la nube?.....</b>	<b>6</b>
<b>Breve historia de las BD en la nube.....</b>	<b>7</b>
¿Qué empresas usan bases de datos en la nube?.....	8

# Introducción: ¿Qué es la nube?

Antes de pasar a hablar de bases de datos en la nube, me parece muy importante definir a qué nos referimos exactamente al decir nube. Con términos como computación en la nube, servicios en la nube, informática en la nube o simplemente la nube, nos referimos al uso de servidores remotos alojados en el internet interconectados a través de una red de alta velocidad para acceder a recursos computacionales a gran escala (como capacidad de almacenamiento, capacidad de procesamiento, ancho de banda, etc.). Es decir, la nube es una tecnología que ofrece prestación de servicios a través de la red y bajo demanda, gracias a esto los usuarios no tienen que poseer estos recursos físicamente o en sus computadoras, sino que pueden “pedirle prestado” estos recursos a una empresa o proveedor que los tenga pagando por el uso de estos para facilitarle el trabajo en la construcción de aplicaciones, almacenar sus datos,

La nube se ha convertido en una herramienta muy importante en la actualidad donde existen una enorme variedad de aplicaciones con muchos usuarios que las utilizan simultáneamente, los desarrolladores no tienen que preocuparse por problemas de infraestructura, hardware o almacenamiento gracias a los servicios que se ofrecen en esta tecnología.

## Descripción, características y funcionalidades

Debido al avance de la computación en la nube(Cloud Computing) y a la creciente demanda de aplicaciones que se ejecutan en el internet y sus usuarios, aparecieron los servicios de bases de datos en la nube, que son Sistemas de gestión de bases de datos que se ejecutan en infraestructuras de Cloud Computing, es decir, en servidores remotos que están alojados en centros de datos.

Las principales características que diferencian este tipo de bases de datos de las tradicionales son:

1. Escalabilidad: estas bases de datos están diseñadas con el objetivo de crecer junto con las necesidades del negocio, por lo que se pueden escalar horizontal o verticalmente, a medida se necesite más capacidad de almacenamiento.
2. Accesibilidad: son accesibles desde cualquier lugar gracias al internet.
3. Redundancia: las bases de datos en la nube se replican en otros servidores remotos para garantizar la recuperación ante desastres e imprevistos.
4. Disponibilidad: por lo mencionado anteriormente, también hace que estas bases de datos estén siempre a disposición del usuario y no se pierda información importante.

La principal funcionalidad de las bases de datos en la nube es integrarse con otras

aplicaciones que también se ejecutan en la nube y soportar análisis de datos en tiempo real para ayudar al cliente a tomar mejores decisiones para su negocio.

## Ventajas y desventajas

### Ventajas

- Costo inicial reducido: podemos usar los servidores de los proveedores así no tenemos que hacer una gran inversión en hardware, software y mantenerlos.
- Escalabilidad: estas bases de datos están diseñadas para ser escalados horizontal y verticalmente.
- Alta disponibilidad: las bases de datos en la nube pueden ser accedidos desde cualquier lugar con conexión a internet.
- Fácil recuperación de datos: ofrecen servicios de backup y recuperación de datos automáticos y programados, lo que garantiza la disponibilidad de los datos en caso de fallas.
- Actualizaciones automáticas: las nuevas versiones de la base de datos se implementan sin la necesidad de interrumpir el servicio.
- Mayor seguridad: los proveedores ofrecen medidas de protección como autenticación y autorización de usuarios, la encriptación de datos, y el monitoreo de actividad sospechosa; además cumplen con estándares de calidad y normas para garantizar la seguridad e integridad de la información de sus clientes.

### Desventajas

- Dependencia de la conexión a internet: cuando no dispongamos de una conexión a internet estable, el rendimiento de la base de datos puede verse afectado.
- Dependencia del proveedor: al confiarle a un proveedor nuestra base de datos se puede perder cierto control sobre los datos y en algunos casos la capacidad de cambiar a un proveedor diferente.
- Personalización limitada: las opciones de personalización de una base de datos en la nube pueden ser limitadas en comparación con una base de datos alojada localmente.
- Incremento de costo: si bien es cierto que el costo inicial al contratar un servicio de BD en la nube es menor comparado a adquirir y configurar servidores propios, este costo irá aumentando a medida que las aplicaciones y los datos vayan creciendo.

# ¿Cuándo conviene usar una base de datos en la nube?

Una base de datos en la nube será la opción ideal si:

- Estamos desarrollando una aplicación web o basada en la web.
- Nuestra aplicación está pensada para crecer y albergar a más usuarios en el futuro.
- No dispongamos de servidores propios/software/hardware para implementar una base de datos propia de acuerdo a nuestras necesidades.
- Necesitamos procesar grandes cantidades de datos y mantener nuestra información íntegra.

Por otro lado, hay algunas situaciones en las cuales una base de datos en la nube no es la opción más conveniente, las cuales son:

- Cuando no queremos depender de un tercero para la administración de nuestros datos.
- Cuando necesitamos mayor personalización y menos limitaciones.
- Cuando no queremos depender de la conexión a internet todo el tiempo.

## Proveedores, servicios y modos de licenciamiento

En la actualidad existe una gran variedad de proveedores de estos servicios, desde las empresas más grandes de la industria de la tecnología hasta medianas y pequeñas empresas.

Los principales proveedores de bases de datos en la nube son:

- Amazon Web Services(AWS)
- Google Cloud
- Microsoft Azure

También podemos mencionar otras empresas grandes que brindan servicios de computación en la nube como:

- Oracle Cloud
- IBM Cloud
- Alibaba Cloud
- Salesforce.com

Existen sin embargo, muchas herramientas de Plataformas de servicios en la nube(Platforms as a service o PaaS) que ofrecen servicios en la nube(entre ellos bases de datos) para desarrolladores de aplicaciones web o móvil, es importante aclarar que estas plataformas no hostean las bases de datos de manera nativa, si no que se integran con otros proveedores de bases de datos para ofrecer estos servicios. Algunas de estas cuentan con un plan gratuito al inicio:

- Heroku (ya no es gratis)

- Firebase(Google)
- Planetscale(hosting de MySQL)
- Vercel
- Netlify
- Render

Los modos de licenciamiento son muy variados entre los diversos proveedores y los diferentes servicios que ofrecen por lo que detallaré solo algunos de los más importantes.

En AWS el modo de licenciamiento de los servicios de bases de datos depende del tipo de base de datos y de si es una opción administrada o no administrada. Por ejemplo, para Amazon RDS (Relational Database Service) se ofrece la opción de utilizar licencias de software propias (BYOL, Bring Your Own License) o de utilizar licencias proporcionadas por AWS (Licencia Incluida). Para Amazon Aurora, que es una base de datos relacional compatible con MySQL y PostgreSQL, se utiliza un modelo de pago por uso y no se requiere una licencia de software adicional. En el caso de Amazon DynamoDB, que es una base de datos NoSQL, también se utiliza un modelo de pago por uso sin necesidad de una licencia adicional.

Para los servicios de Google Cloud como Cloud SQL, Cloud Spanner, Firestore o Bigtable cuentan con licencias de pago por uso, es decir, se paga por los recursos utilizados, aunque también ofrecen suscripciones mensuales y anuales.

Para Microsoft Azure, algunas bases de datos como Azure SQL Database y Azure Cosmos DB vienen incluidos con la suscripción a este servicio.

## ¿Cómo usar una BD en la nube?

Para poder utilizar una base de datos en la nube primero debemos elegir un proveedor y un servicio específico, tendremos que crear una cuenta para la plataforma de la nube, una vez creada la cuenta ,crearemos la base de datos, la configuramos, conectamos nuestras aplicaciones a esta y ya podremos usarla para almacenar nuestros datos y realizar todas las operaciones de las bases de datos.

Ejemplo:

Planetscale, hosting de MySQL en la nube que cuenta con un plan gratuito inicial  
<https://planetscale.com/>

créditos: Youtube - Fazt

Login: creamos nuestra cuenta con nuestro correo o con github

## Sign in

New to PlanetScale? [Sign up for an account.](#)

Email

Password

[Forgot password?](#)



[Sign in](#)

Or with



[GitHub](#)

[Single sign-on](#)

Creamos nuestra primera base de datos(Create your first database)



### Insights and monitoring

Get insight into production traffic with graphs tracking database reads and writes. Identify and optimize slow queries using the query statistics feature.

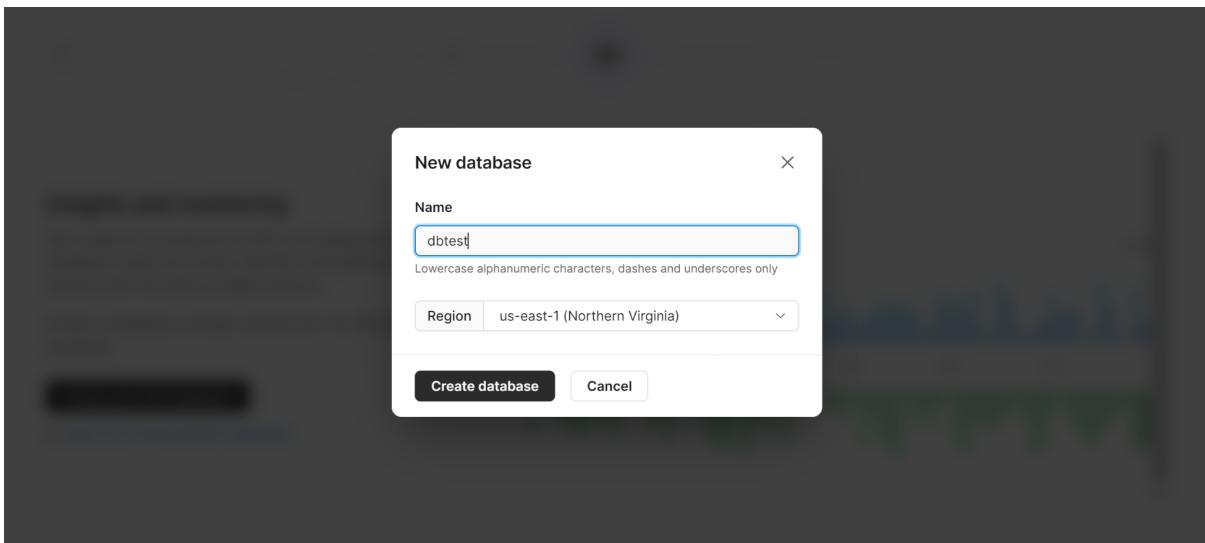
Create a database and get started with the PlanetScale workflow.

[Create your first database](#)

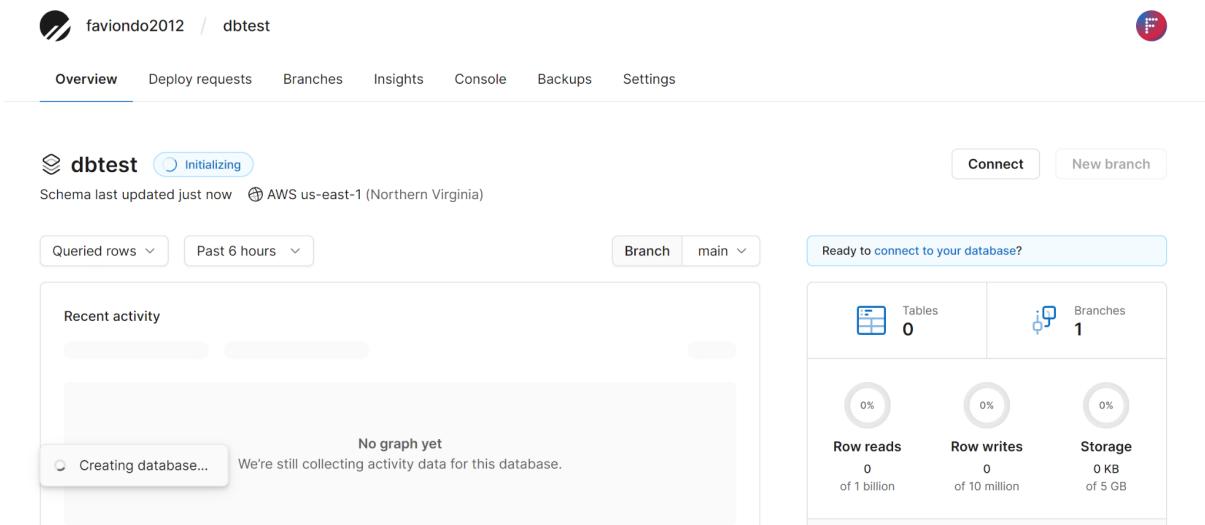
or import an existing MySQL database



## Nombrar la base de datos

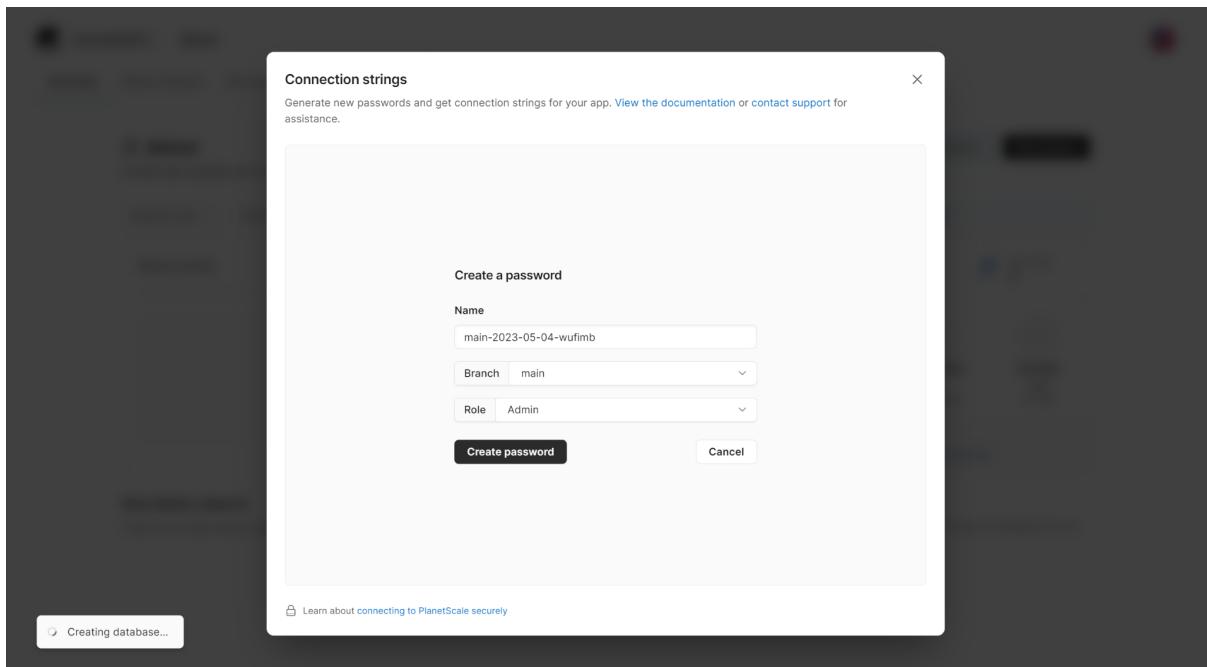


## Hacemos click en connect

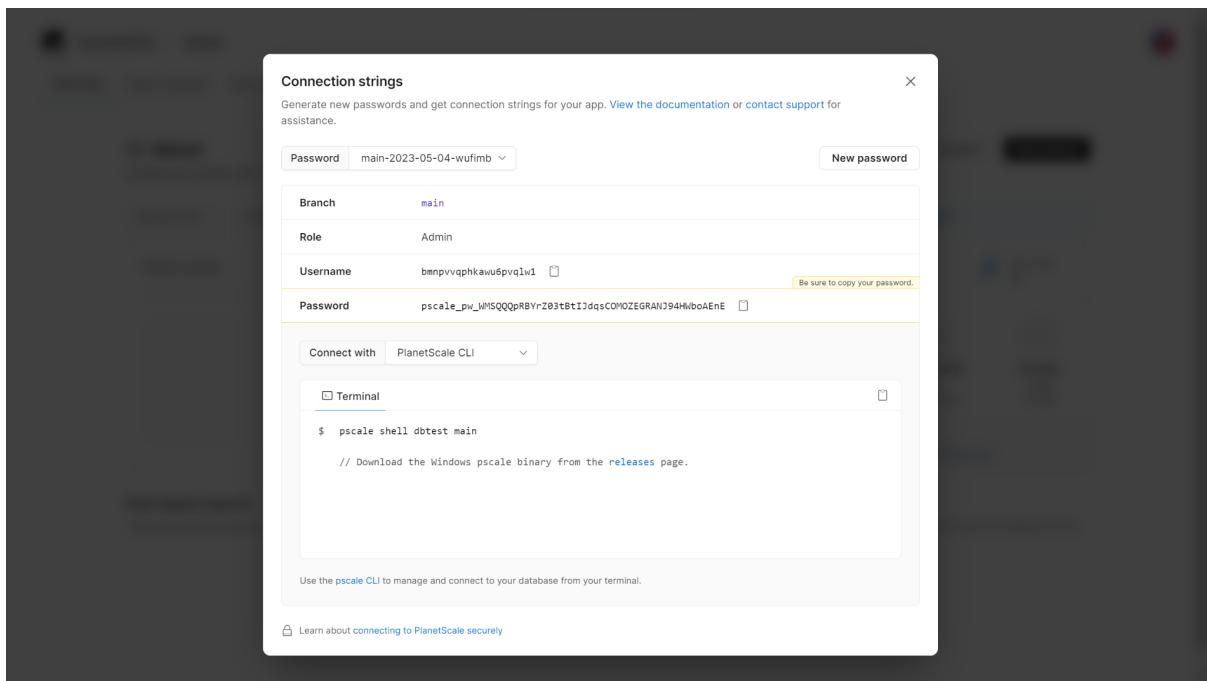


The screenshot shows the AWS Amplify Database console for the dbtest database. At the top, there's a navigation bar with the user icon, 'faviondo2012', and the database name 'dbtest'. Below the navigation bar, there are tabs: Overview (which is selected), Deploy requests, Branches, Insights, Console, Backups, and Settings. The main content area has a header 'dbtest' with a status indicator 'Initializing'. It also shows the last update time 'Schema last updated just now' and the region 'AWS us-east-1 (Northern Virginia)'. On the left, there's a 'Recent activity' section with a note 'Creating database...' and a message 'No graph yet'. On the right, there's a summary card with the heading 'Ready to connect to your database?'. It displays the following metrics: Tables (0), Branches (1), Row reads (0% of 1 billion), Row writes (0% of 10 million), and Storage (0 KB of 5 GB). The 'Connect' button is located in the top right corner of the main content area.

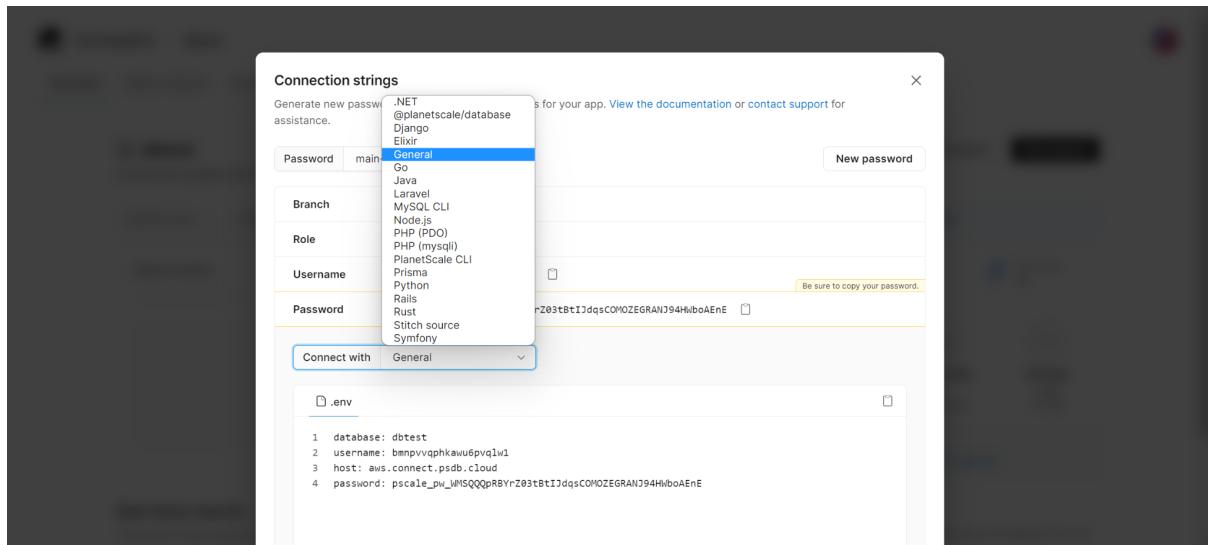
Generamos una contraseña para conectarnos a la base de datos



Obtendremos las credenciales que nos servirán para conectarnos desde nuestra aplicación



Podemos seleccionar distintas estructuras de string para conectarnos



Nos conectamos en nuestra aplicación copiando las credenciales y ya podemos usar la base de datos para ejecutar consultas.

Ej en python:

```
import MySQLdb

DB_HOST = 'host'
DB_USER = 'username'
DB_PASS = 'password'
DB_NAME = 'database'

def run_query(query=''):
    datos = [DB_HOST, DB_USER, DB_PASS, DB_NAME]

    conn = MySQLdb.connect(*datos) # Conectar a la base de datos
    cursor = conn.cursor()        # Crear un cursor
    cursor.execute(query)         # Ejecutar una consulta

    if query.upper().startswith('SELECT'):
        data = cursor.fetchall()   # Traer los resultados de un select
    else:
        conn.commit()             # Hacer efectiva la escritura de los cambios
    datos

    data = None

    cursor.close()               # Cerrar el cursor
    conn.close()                 # Cerrar la conexión

return data
```

# Breve historia de las BD en la nube

1960: IBM lanza el primer modelo de computación en la nube conocido como "servicio de tiempo compartido".

1999: Salesforce.com lanza su primer servicio de gestión de relaciones con el cliente (CRM) basado en la nube.

2006: Amazon Web Services (AWS) lanza su primer servicio de bases de datos en la nube, Amazon SimpleDB.

2007: Google lanza Google App Engine, una plataforma de aplicaciones web en la nube.

2009: Microsoft lanza Windows Azure, su plataforma en la nube.

2011: Oracle lanza Oracle Database Cloud Service, una plataforma de bases de datos en la nube.

2012: Amazon lanza Amazon RDS, un servicio de bases de datos relacionales en la nube.

2015: Google lanza Google Cloud SQL, su servicio de bases de datos relacionales en la nube.

2016: Microsoft lanza Azure SQL Database, su servicio de bases de datos en la nube.

2017: AWS lanza Amazon Aurora Serverless, un servicio de bases de datos sin servidor.

2018: IBM lanza IBM Cloud Databases, una plataforma de bases de datos en la nube.

2019: Alibaba Cloud lanza PolarDB, su servicio de bases de datos relacional en la nube.

2020: AWS lanza Amazon Keyspaces (para Apache Cassandra), un servicio de bases de datos en la nube para aplicaciones de escala web.

2021: Google lanza Firestore, una base de datos NoSQL en tiempo real en la nube.

## ¿Qué empresas usan bases de datos en la nube?

Empresas que usan servicios de AWS:

- Netflix (DynamoDB)
- Airbnb(RDS)
- Pfizer(Aurora)
- Samsung (Redshift)
- Unilever(Neptune)

Empresas que usan servicios de Microsoft Azure:

- Ford
- HP
- Boeing
- Siemens

1. Descripción de la arquitectura, las características y sus funcionalidades.
2. Ventajas y desventajas.
3. Comparación contra los otros tipos de datos, cuándo conviene usarla y cuándo no.
4. Modo de licenciamiento, de ser posible comentar acerca de los precios.
5. Mostrar la instalación del motor de la BD siempre que sea posible.
6. Años de antigüedad de su creación, principales empresas en donde funciona, proveedor que facilita el producto.

El software de código abierto, se caracteriza porque el código fuente de este es publicado bajo una licencia de código abierto o forma parte del dominio público, lo que permite que cualquier usuario libremente pueda utilizar, cambiar y redistribuir, total o parcialmente, el software producto de ese código abierto; estas atribuciones normalmente son exclusivos de quienes poseen el derecho de autor sobre ese software. Al referirse principalmente al licenciamiento y derechos de autoría, el software de código abierto abarca todo tipo de software, no solamente bases de datos.

En ese sentido, no hay una arquitectura, características o funcionalidades propias de las bases de datos por ser de código abierto, más allá de las características mencionadas antes respecto a los derechos de explotación y distribución relacionadas al tipo de licenciamiento.

Las bases de datos de código abierto presentan una ventaja económica importante para los usuarios finales (personas o empresas) frente a las bases de datos con software propietario, y la creciente utilización de bases de datos de código abierto por las grandes empresas ha fomentado el desarrollo y perfeccionamiento de distintas alternativas de código abierto para suprir todo tipo de necesidades, desde bases de datos relacionales de alta fiabilidad como PostgreSQL a opciones NoSQL como MongoDB y Neo4j, bases de datos documentales y de grafos, respectivamente.

La utilización masiva de bases de datos ha creado un amplio espectro de explotación comercial de estos productos. Si bien el licenciamiento de código abierto implica una utilización libre del software y su código fuente/binarios, el paso a la nube ha permitido a los distintos proveedores explotar comercialmente estos productos al proveer de la infraestructura y todo lo necesario para correr las bases de datos en la nube en vez de “on-premise”. Otra forma de lucrar a partir de un software de código abierto ha sido a través de suscripciones a servicios de soporte o a funcionalidades adicionales que no formaban parte del código original, y dependiendo del licenciamiento, no está obligado a distribuirse libremente por lo que se pueden ejecutar mayores restricciones en cuanto a explotación/distribución de las funcionalidades adicionales, bajo el mismo, o distinto nombre.

Entre las bases de datos de código abierto más famosas tenemos:

- PostgreSQL: base de datos relacional, clientes: Apple, IMDb, Instagram, Reddit.
- MariaDB: base de datos relacional, clientes: AWS, MS Azure, Red Hat, Google, Mozilla.
- MongoDB: base de datos documental, clientes: Verizon, Adobe.
- Neo4j: base de datos de grafos, clientes: Levi Strauss & Co., PWC, eBay, JPMorgan, Chase, Citi, UBS.

- Redis: base de datos en memoria/clave-valor, clientes: Twitter, GitHub, Snapchat, StackOverflow.
- Cassandra: base de datos columnar, clientes: AT&T, T-Mobile, Tata Motors, Activision Blizzard Inc.

Además, tenemos otras bases de datos innovadoras que cumplen funciones más específicas como Timescale, que es una base de datos especializada para series temporales, utilizada por compañías como Marvel Studios, Apple, Warner Music Group, Walmart y Salesforce.

Como ya se mencionó previamente, el software de código abierto permite que cualquier usuario libremente pueda utilizar, cambiar y redistribuir, total o parcialmente, el software producto de ese código abierto. Debido a eso, todas estas bases de dato tienen licencias de código abierto que respaldan esta decisión de permitir el acceso gratuitamente a usuarios. La mayoría de ellas, como MariaDB, Neo4J y MongoDB usan General Public License o similares. También se usa la licencia Berkeley Software Distribution, usada por Redis y PostgreSQL. Ambas licencias permiten la ejecución, uso, estudio y modificación del código de la base de datos, teniendo su única diferencia en que la licencia BSD permite la ejecución del código fuente en software no libre.

Sin embargo, a pesar de ser gratuitas y libres de usar, las empresas que las administran ofrecen servicios de pago relacionados al uso en la nube y el soporte comercial a empresas. Por ejemplo, MariaDB ofrece soluciones a escala productiva a demanda, y la posibilidad de trabajar y respaldar en la nube por alrededor de 2 dólares por día. AuraDB ofrece servicio profesional por 65 dólares al mes, y soluciones empresariales también a demanda. Y MongoDB ofrece servicios en servidor a 57 dólares al mes, además de servicio sin servidor por 10 centavos de dólar cada 1 millón de lecturas.

La historia de las bases de datos open source comenzó en los inicios de los años 80, cuando el código abierto en general comenzó a desarrollarse debido a la necesidad de que no todo el software sea privativo. Sin embargo, fue recién hacia fines de los años 90 cuando se popularizó la práctica, con el lanzamiento de Netscape y StarOffice.

Al día de la fecha, las bases de datos de código abierto son uno de los segmentos crecientes en el mercado internacional, representando una gran parte de las bases de datos usadas a nivel global. Tal es el punto, además, que como se vio antes, muchas grandes empresas de prestigio internacional usan estas bases de datos como parte de su esquema de negocio. Algunas de ellas son Apple, IMDb, Instagram y Reddit (PostgreSQL), AWS, MS Azure, Red Hat, Google y Mozilla (MariaDB), Verizon y Adobe (MongoDB), Levi Strauss & Co., PWC, eBay, JPMorgan, Chase, Citi y UBS (Neo4J), Twitter, GitHub, Snapchat y StackOverflow (Redis) y AT&T, T-Mobile, Tata Motors y Activision Blizzard Inc (Cassandra). En su mayoría, estas bases de datos están desarrollados y son proveídas por las mismas entidades, siendo estas empresas (MongoDB Inc, Redis Ltd, Neo Technology, PostgreSQL Group) o fundaciones para desarrollar el programa (MariaDB Foundation, Apache Software Foundation).



# BASES DE DATOS OPEN SOURCE

## LABORATORIO IV

ENOC GARCÍA

JULIÁN MARTÍNEZ FONTAÑA

# SOFTWARE DE CÓDIGO ABIERTO

- Publicado bajo licencia de código abierto, forma parte del dominio público.
- Utilización, modificación y distribución libre.



# CARACTERÍSTICAS

- Más económicas.
- Fomento del desarrollo de alternativas.
- Amplia variedad de servicios comerciales y soporte.



# EJEMPLOS DE BD OPEN SOURCE

- PostgreSQL → BD relacional. Apple, IMDb, Instagram, Reddit.



# EJEMPLOS DE BD OPEN SOURCE

- MariaDB → BD relacional. AWS, MS Azure, Red Hat, Google, Mozilla.



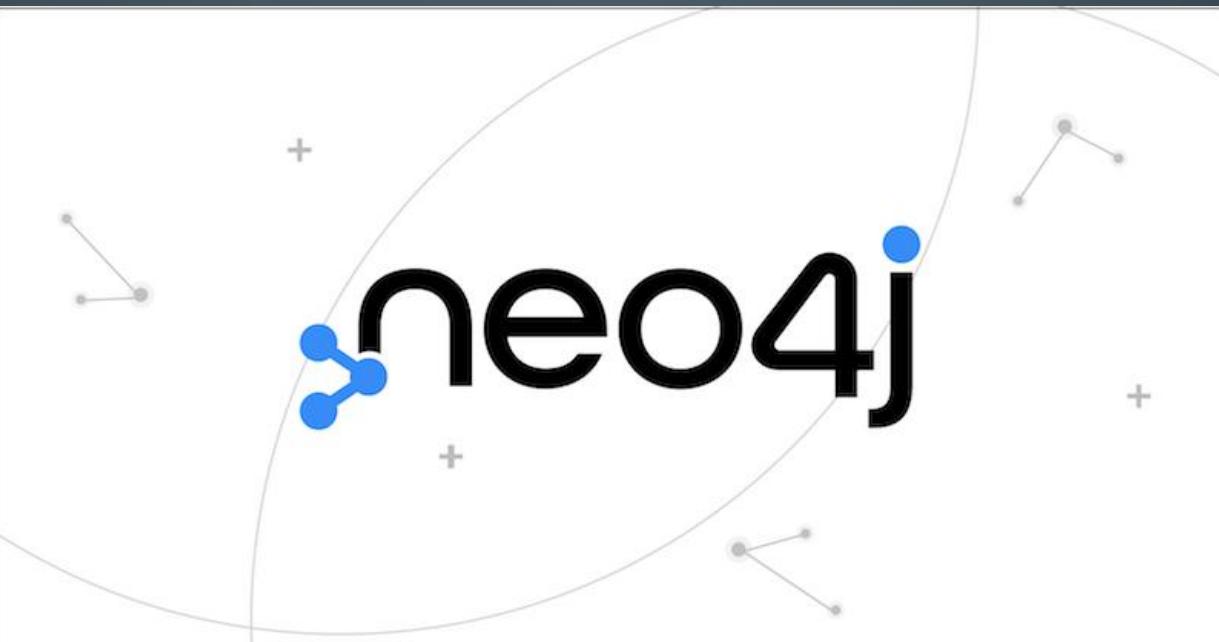
# EJEMPLOS DE BD OPEN SOURCE

- MongoDB → BD documental. Verizon, Adobe.



# EJEMPLOS DE BD OPEN SOURCE

- Neo4J → BD de grafos. Levi Strauss & Co., PWC, eBay, JPMorgan, Chase, City, UBS.



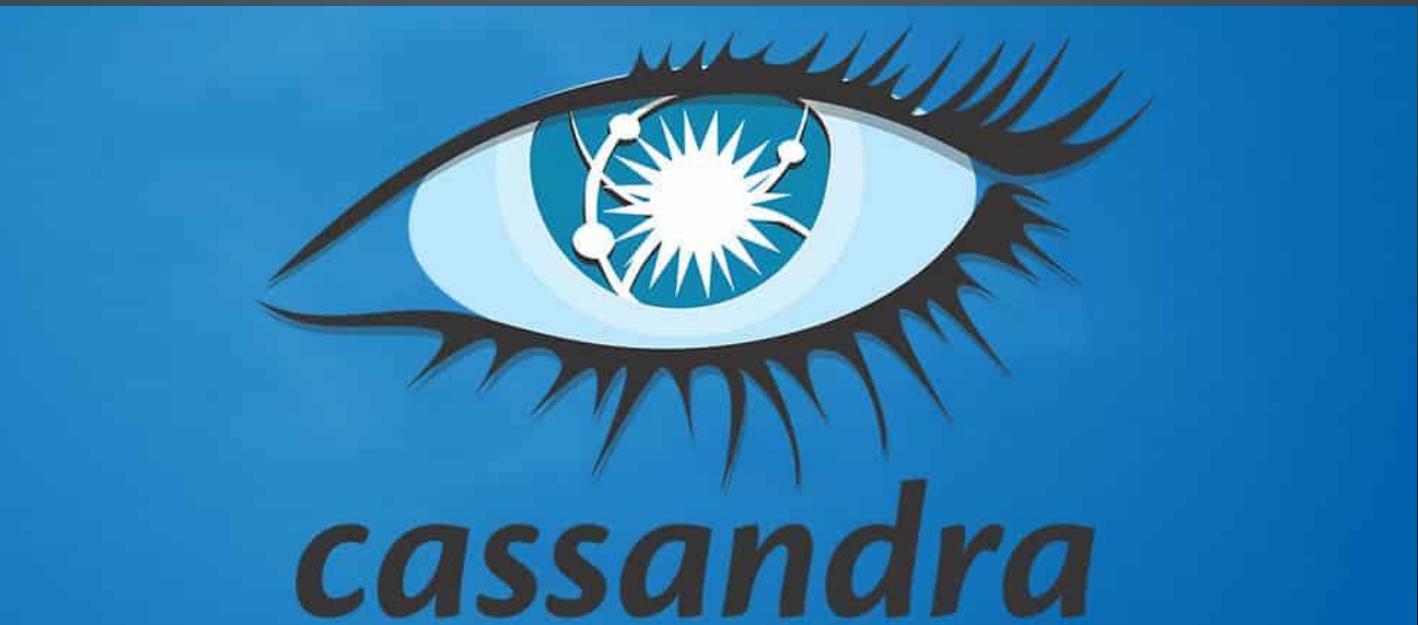
# EJEMPLOS DE BD OPEN SOURCE

- Redis → BD en memoria/clave-valor. Twitter, GitHub, Snapchat, StackOverflow.



# EJEMPLOS DE BD OPEN SOURCE

- Cassandra → BD columnar. AT&T, T-Mobile, Tata Motors, Activision Blizzard Inc.



# EJEMPLOS DE BD OPEN SOURCE

- Timescale → BD especializada para series temporales. Marvel Studios, Apple, Warner Music Group, Walmart, Salesforce.



Timescale

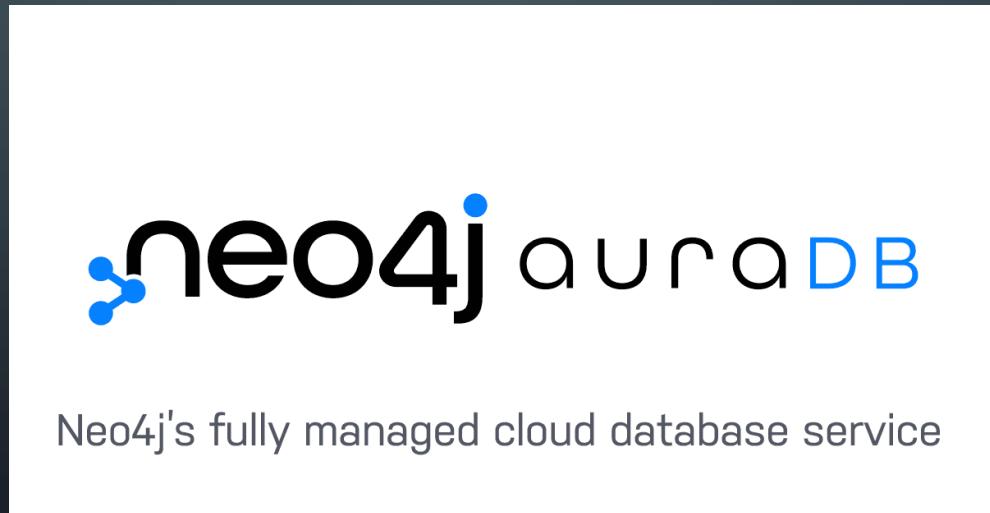
# MODO DE LICENCIAMIENTO

- General Public License (MariaDB, Neo4J, MongoDB).
- Berkeley Software Distribution (Redis, PostgreSQL).



# SERVICIOS ADICIONALES

- Empresas ofrecen servicios de pago para soporte comercial.
- Servidores entre 57 y 60 dólares al mes.
- Arquitecturas sin servidor.



# HISTORIA DE LAS BD OPEN SOURCE

- 1980 → Desarrollo del código abierto en general.
- Fines de los 90 → Lanzamiento de Netscape y StarOffice.
- Posterior aumento del desarrollo global.

