Teoría de persistencia parcial 1

Estrategias de persistencia

Clase N° 1:

¿Qué son las condiciones que garantizan la integridad de los datos?

Existen dos tipos:

Las restricciones de integridad de usuario: proporcionan un medio de asegurar que los cambios que se hacen en la bdd por los usuarios autorizados no generen inconsistencia de datos. Ej: que no se pueda poner precio negativo en un sistema de proveedores.

Las reglas de integridad de modelo: estandares de base de datos.

Reglas de Integridad

Regla de integridad de unicidad de la clave primaria: la clave primaria tiene que ser única.

Regla de integridad de entidad de la clave primaria: la clave primaria no puede ser nula

Regla de integridad referencial: está relacionada con el concepto de una clave foránea, esta misma permite la conexión entre dos tablas a través de la clave primaria de una, y la clave foránea de la otra

Restricción: permite no borrar una tupla si está referenciada por alguna otra a través de la clave foránea.

Actualización en cascada: permite modificar una clave primaria, siempre y cuando se modifiquen las que mantienen referencia a la misma. Esto permite una actualización en cascada como el nombre lo indica

Anulación: se modifica la tupla referenciada y las tuplas que la referenciaban ponen a nulo la clave foránea (sólo si acepta nulos).

Regla de integridad de dominio: La regla de integridad de dominio está relacionada con la noción de dominio. Esta regla establece dos condiciones.

Primera condición:

- Permite que cuando definas un atributo el mismo sea respetado por el tipo de dato del mismo Ej: DNI = número. No es posible que DNI = Hola

Segunda condición:

- Permite no poder consultar si existe una variable de tipo de dato A, y pedimos que busque que sea de tipo de dato B. Ej: DNI = número? (válido consultar) . DNI = Hola? (no válido)

Clase N° 2:

¿Que es una Base de Datos Distribuida?

Es una colección de datos que pertenecen lógicamente a un solo sistema, pero se encuentra físicamente distribuido en varios computadores o servidores de datos en una red de computadoras.
Es una base de datos construida sobre una red de computadores**.**
La información que estructura la base de datos está almacenada en diferentes sitios en la red, y los diferentes sistemas de información que las utilizan accesan datos en distintas posiciones geográficas.

Base de datos homogénea:
En los sistemas de bases de datos distribuidas homogéneas todos los sitios emplean idéntico software de gestión de bases de datos, son conscientes de la existencia de los demás sitios y acuerdan cooperar en el procesamiento de las solicitudes de los usuarios.

Base de datos heterogénea:
En las bases de datos distribuidas heterogéneas puede que los diferentes sitios utilicen esquemas y software de gestión de sistemas de bases de datos diferentes. Puede que algunos sitios no tengan información de la existencia del resto y que sólo proporcionen facilidades limitadas para la cooperación en el procesamiento de las transacciones.

También dice:
Almacenamiento distribuido de datos
Réplica de datos

Transparencia: no se debe exigir a los usuarios de los sistemas distribuidos de bases de datos que conozcan la ubicación física de los datos ni el modo en que se puede acceder a ellos en cada sitio local concreto.

Transparencia de la fragmentación: no se exige a los usuarios que conozcan el modo en que se ha fragmentado la relación.

Transparencia de la réplica: los usuarios ven cada objeto de datos como lógicamente único. Puede que el sistema distribuido replique los objetos para incrementar el rendimiento del sistema o la disponibilidad de los datos. Los usuarios no deben preocuparse por los objetos que se hayan replicado ni por la ubicación de esas réplicas.
Transparencia de la ubicación: no se exige a los usuarios que conozcan la ubicación física de los datos. El sistema distribuido de bases de datos debe poder hallar los datos siempre que la transacción del usuario facilite el identificador de esos datos.

¿Qué tipos de fallos existen en un sistema de base de datos?

Los sistemas compartidos pueden sufrir varios tipos de errores o fallos, ya sea de software o hardware, al igual que los sistemas centralizados, pero también se pueden producir fallos propios como lo son:

- Fallos de sitios.
- Pérdidas de mensajes.
- Fallos de enlaces de comunicaciones.
- Divisiones de la red.

¿Cuáles son los principios fundamentales de un sistema de base de datos distribuido?

1. Autonomía Local: Los sitios distribuidos deben ser autónomos, es decir que todas las operaciones en un sitio dado se controlan en ese sitio.

2. No dependencia de un sitio central: No debe de haber dependencia de un sitio central para obtener un servicio. La dependencia de un sitio sería indeseable por las siguientes razones: ese sitio podría ser un cuello de botella. El sistema sería vulnerable; si el sitio sufriera un desperfecto, todo el sistema dejaría de funcionar.

3. Operación Continua: Nunca debería haber necesidad de apagarse apropósito para que se pueda realizar alguna función, como añadir un nuevo sitio, o instalar una versión mejorada.

4. Independencia con respecto a la localización: No debe de ser necesario que los usuarios sepan dónde están almacenados físicamente los datos, el usuario debe de ver como si solo existiera un sitio local.

5. Independencia con respecto a la fragmentación: El usuario desconoce el número de fragmentos existentes. La fragmentación es deseable por razones de desempeño, los datos, pueden almacenarse en la localidad donde se utilizan con mayor frecuencia de manera que la mayor parte de las operaciones sean sólo locales y se reduzca el tráfico en la red.

6. Independencia de réplica: Si una relación dada, es decir, un fragmento dado de una relación se puede presentar en el nivel físico mediante varias copias almacenadas o réplicas, en muchos sitios distintos. La réplica es viable por dos razones: las aplicaciones pueden operar sobre copias locales en vez de tener que comunicarse con sitios remotos; una mejor disponibilidad. La desventaja principal de las réplicas es, cuando se pone al día un cierto objeto copiado, deben ponerse al día todas las

réplicas de ese objeto.

7. Procesamiento Distribuido de Consultas: El objetivo es convertir transacciones de usuario en instrucciones para manipulación de datos, y así reducir el tráfico en la red, esto implica que el proceso mismo de optimización de consultas debe ser distribuido.

8. Manejo Distribuido de Transacciones: Tiene dos aspectos principales, el control de recuperación y el control de concurrencia, cada uno de los cuales requiere un tratamiento más amplio en el ambiente distribuido.

9. Independencia con respecto al equipo: El SGBD (Sistema de Gestión de Base de Datos) debe ser ejecutable en diferentes plataformas hardware.

10. Independencia con respecto al Sistema Operativo: El sistema debe ser ejecutable en diferentes Sistemas Operativos.

11. Independencia con respecto a la red: El sistema debe poder ejecutarse en diferentes tipos de redes: topología y tecnología de comunicación

12. Independencia con respecto al SGBD: No se requiere que los SGBD en los diferentes sitios manejen todos la misma interfaz; no necesitan ser por fuerza copias del mismo sistema.

// Resumen:

• Los sistemas distribuidos de bases de datos consisten en un conjunto de sitios, cada uno de los cuales mantiene un sistema local de bases de datos. Cada sitio puede procesar las transacciones locales: las transacciones que sólo acceden a datos de ese mismo sitio. Además, cada sitio puede participar en la ejecución de transacciones globales: las que acceden a los datos de varios sitios. La ejecución de las transacciones globales necesita que haya comunicación entre los diferentes sitios.

• Las bases de datos distribuidas pueden ser homogéneas, en las que todos los sitios tienen un esquema y un código de sistemas de bases de datos comunes, o heterogéneas, en las que el esquema y el código de los sistemas pueden ser diferentes.

• Surgen varios problemas relacionados con el almacenamiento de relaciones en bases de datos distribuidas, incluidas la réplica y la fragmentación. Es fundamental que el sistema minimice el grado de conocimiento por los usuarios del modo en que se almacenan las relaciones.

• Los sistemas distribuidos pueden sufrir los mismos tipos de fallos que los sistemas centralizados. No obstante, hay otros fallos con los que hay que tratar en los entornos distribuidos; entre ellos, los fallos de los sitios, los de los enlaces, las pérdidas de mensajes y las divisiones de la red. Es necesario tener en consideración cada uno de esos problemas en el diseño del esquema distribuido de recuperación.

• Para asegurar la atomicidad, todos los sitios en los que se ejecuta la transacción T deben estar de acuerdo en el resultado final de su ejecución. O bien T se compromete en todos los

sitios o se aborta en todos. Para asegurar esta propiedad el coordinador de la transacción de T debe ejecutar un protocolo de compromiso. El protocolo de compromiso más empleado es el protocolo de compromiso de dos fases.

• El protocolo de compromiso de dos fases puede provocar bloqueos, la situación en que el destino de una transacción no se puede determinar hasta que se recupere un sitio que haya fallado (el coordinador). Se puede utilizar el protocolo de compromiso de tres fases para reducir la probabilidad de bloqueo.

•La mensajería persistente ofrece un modelo alternativo para el tratamiento de las transacciones distribuidas. El modelo divide cada transacción en varias partes que se ejecutan en diferentes bases de datos. Se envían mensajes persistentes (que está garantizado que se entregan exactamente una vez, independientemente de los fallos) a los sitios remotos para solicitar que se emprendan acciones en ellos. Aunque la mensajería persistente evita el problema de los bloqueos, los desarrolladores de aplicaciones tienen que escribir código para tratar diferentes tipos de fallos.

• Los diferentes esquemas de control de concurrencia empleados en los sistemas centralizados se pueden modificar para su empleo en entornos distribuidos. ¤ En el caso de los protocolos de bloqueo, la única modificación que hay que hacer es el modo en que se implementa el gestor de bloqueos. Existen varios enfoques posibles. Se pueden utilizar uno o varios coordinadores centrales. Si, en vez de eso, se adopta un enfoque con un gestor distribuido de bloqueos, hay que tratar de manera especial los datos replicados. ¤ Entre los protocolos para el tratamiento de los datos replicados se hallan el protocolo de copia principal, el de mayoría, el sesgado y el de consenso de quórum. Cada uno de ellos representa diferentes equilibrios en términos de coste y de posibilidad de trabajo en presencia de fallos.
¤ En el caso de los esquemas de marcas temporales y de validación, el único cambio necesario es el desarrollo de un mecanismo para la generación de marcas temporales globales únicas. ¤ Muchos sistemas de bases de datos soportan la réplica perezosa, en la que las actualizaciones se propagan a las réplicas ubicadas fuera del ámbito de la transacción que ha llevado a cabo la actualización. Estas facilidades deben utilizarse con grandes precauciones, ya que pueden dar lugar a ejecuciones no secuenciables.

• La detección de interbloqueos en entornos con gestor distribuido de bloqueos exige la colaboración entre varios sitios, dado que puede haber interbloqueos globales aunque no haya ningún interbloqueo local.

• Para ofrecer una elevada disponibilidad, las bases de datos distribuidas deben detectar los fallos, re configurarse de modo que el cálculo pueda continuar y recuperarse cuando se repare el procesador o el enlace correspondiente. La tarea se complica enormemente por el hecho de que resulta difícil distinguir entre las divisiones de la red y los fallos de los sitios. Se puede extender el protocolo de mayoría mediante el empleo de números de versión para permitir que continúe el procesamiento de las transacciones incluso en presencia de fallos. Aunque el protocolo supone una sobrecarga significativa, funciona independientemente del

tipo de fallo. Se dispone de protocolos menos costosos para tratar los fallos de los sitios, pero dan por supuesto que no se producen divisiones de la red.

• Puede que las consultas a las bases de datos distribuidas necesiten tener acceso a varios sitios. Se dispone de varias técnicas de optimización para escoger los sitios a los que hay que tener acceso. Basadas en la fragmentación y en la réplica, las técnicas pueden utilizar técnicas de semi reunión para reducir las transferencias de datos.

• Las bases de datos distribuidas heterogéneas permiten que cada sitio tenga su propio esquema y su propio código de sistema de bases de datos. Los sistemas de bases de datos múltiples ofrecen un entorno en el que las nuevas aplicaciones de bases de datos pueden acceder a los datos de gran variedad de bases de datos ya existentes ubicadas en diferentes entornos heterogéneos de hardware y de software. Puede que los sistemas locales de bases de datos empleen modelos lógicos y lenguajes de definición o de manipulación de datos diferentes y que se diferencien en los mecanismos de control de concurrencia o de administración de las transacciones. Los sistemas de bases de datos múltiples crean la ilusión de la integración lógica de las bases de datos, sin exigir su integración física.

Clase n°5:
Una Aplicación consta de dos módulos, un módulo Frontend y un módulo Backend.
● El módulo FrontEnd es lo que visualiza el usuario, pantallas, formularios, etc.
● El módulo BackEnd es lo que realiza la acción o la funcionalidad propiamente del sistema.

REQUEST: petición a una API.
RESPONSE: respuesta de esa API.

Qué es un ORM y como se utiliza?

Significa Object Relational Mapping.
Nos permite comunicarnos con nuestra base de datos evitando escribir sentencias de SQL, seria como una interfaz más amigable para poder trabajar con un motor de base de datos.
Permite almacenar y leer objetos de tu BD fácilmente.

- *Object:* estos se definen usualmente con un lenguaje de programación, en nuestro caso se define por JS en Node, se crea un archivo que se conoce como el modelo y en él se definen la forma que van a tener nuestros datos.
  Es decir, en el **Object**RM es donde le das la forma al objeto de clientes, iD, con nombre, apellido, tel, etc. y se definen con un lenguaje de programación.
- *Relational:* es la BD, donde se almacenan los objetos.
- *Mapping:* es la unión de ambos, esto me permite que con código de lenguaje de programación, pueda realizar las consultas a la BD, unir los objetos con el Relational y utilizar esos datos y mostrarlos en mi aplicación.

VENTAJAS DEL ORM

Primero, evitemos la repetición de código, ya que el modelo se define en un lugar y solo con importar el modelo al controlador, tendrás acceso a todos los métodos del ORM.

Aceleran mucho el proceso de desarrollo ya que hay una gran cantidad de métodos para el CRUD (?.

No es necesario escribir código SQL.

Seguridad: los ORM cuentan con sentencias preparadas para evitar la inyección de datos no esperados y con una gran cantidad de medidas de seguridad que hacen que no tengas que preocuparte por esto en tus aplicaciones.

Escrito en el lenguaje de tu aplicación web.

DESVENTAJAS DEL ORM

Siempre hay que aprender una sintaxis nueva, eso puede ser una desventaja.

Cuando alguien comienza a utilizar un ORM, comienza a utilizar código SQL, y si es muy fácil la consulta, tiran un SELECT, y si es muy complicada la consulta, deciden utilizar lenguaje ORM, y eso no se hace.

Siempre hay que utilizar los métodos del ORM, buscar y encontrar las funciones que hacen lo que necesitamos.

Instalarlo a veces no es tan sencillo, e instalarlo localmente es un problema, y en un servidor, mucho mas aún. (aunque NODE lo hace muy sencillo gracias a NPM)

En cuanto a performance es un poco mas lento que las consultas SQL nativas.

Un ORM es una capa de abstracción sobre la BD, si utilizar un método para crear un registro, el ORM se encarga de escribir el código SQL, pero tu estás escribiendo código en tu lenguaje de programación. Es decir en lo que se interpreta tu lenguaje, mas lo que se ejecuta, puede llegar a ser un poco mas lento que si estuvieras escribiendo las consultas nativas SQL. Nada es mas rápido que consulta nativa, el ORM puede ser un poquito mas lento, pero te salva de aprender código SQL si no lo sabes.

Un ORM no es específico de NODE. Muchos lenguajes de programación tienen un ORM:
PHP: Propel y Doctrine.
JAVA: Hibernate.
Python: SQL Alchemy.
C#: Entity Framework.

Y en muchos lenguajes vas a encontrar un ORM. Es una herramienta muy común que ahorra mucho tiempo.

En cuanto a NODE, existen dos opciones muy claras:
SEQUELIZE soporta MySQL, PostgreSQL, SQL Server y SQLite.
Y otro es MONGOOSE que permite conectar MongoDB con tus aplicaciones NodeJS.

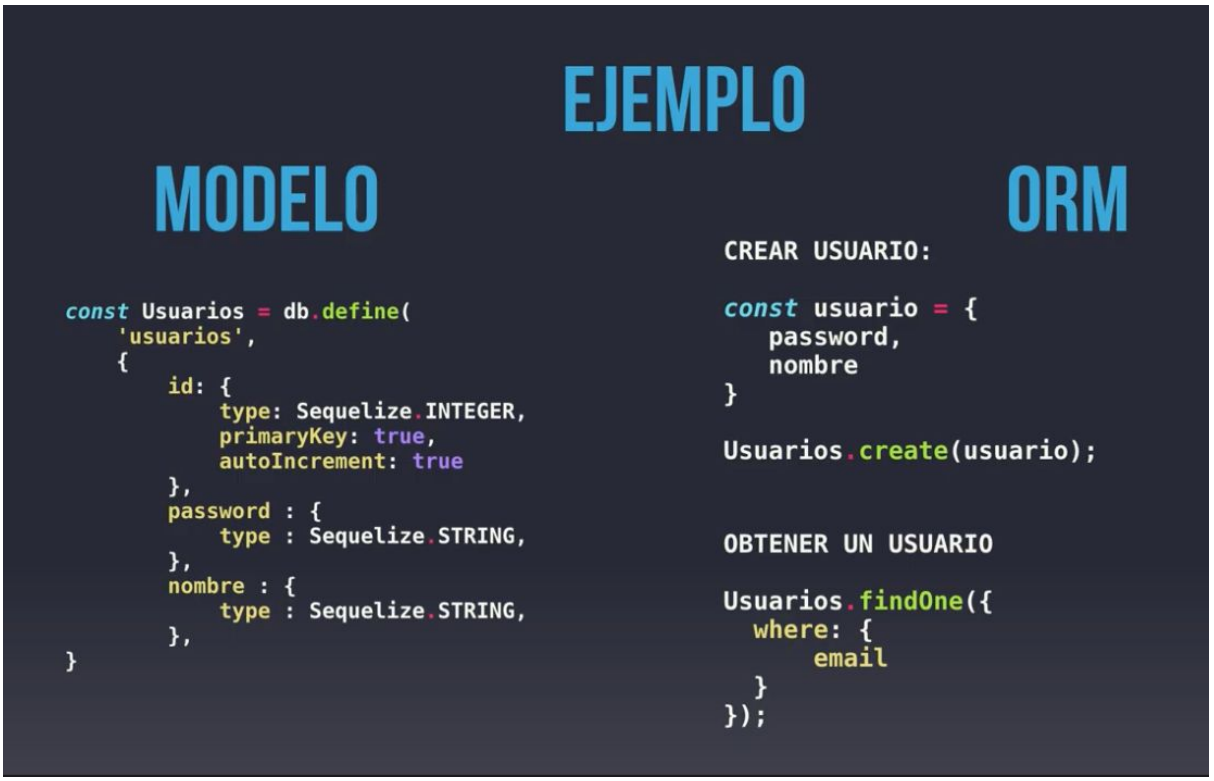
Muchas veces MODELO y ORM **están muy relacionados** y puede ser posible que sea difícil de comprender qué es lo que hace cada uno.

- El modelo es un lugar único donde se describe la forma de los objetos y esta forma hace que sea fácil agregar y quitar campos para la BD.
  Este describe la tabla de productos que tiene un ID, un nombre, un código, un precio, que type va a ser, todo esto se define en el modelo.

- Mientras que el ORM tiene los métodos para la BD.
  Este cuenta con todos los métodos, para crear, para eliminar, para actualizar, para relacionar productos con una categoría, productos con un cliente, etc.

La clara diferencia acá, es que el modelo da la estructura de todos los datos, mientras que el ORM, nos da los métodos para trabajar o consultar la BD.

Para ejemplificar citamos una imagen:



CLASE 6

Cuestionario Base de Datos NoSql

1) ¿Qué entiende por base de datos No Relacional?

NoSQL (Not Only SQL) realmente es una categoría muy amplia para un grupo de soluciones de persistencia que no siguen el modelo de datos relacional, y que no utilizan SQL como lenguaje de consulta; pero en resumen, las bases de datos NoSQL pueden clasificarse en función de su modelo de datos en las siguientes cuatro categorías:

• Orientadas a clave-valor (Key-Value stores)
• Orientadas a columnas (Wide Column stores)
• Orientadas a documentos (Document stores)
• Orientadas a grafos (Graph databases)

2) ¿ qué es una base de datos orientadas a documentos?

Una base de datos orientada a documentos está diseñada para gestionar información orientada a documentos o datos semi-estructurados. Este tipo de bases de datos constituye una de las principales categorías de las llamadas bases de datos NoSQL.
Almacenar y recuperar todos los datos relacionados como una sola unidad puede entregar ventajas enormes en el rendimiento y la escalabilidad. De este modo, los gestores de datos no tienen que hacer operaciones complejas como las uniones para encontrar los datos que normalmente están relacionados, ya que todo se encuentra en un mismo lugar

3) ¿ qué es un Json?

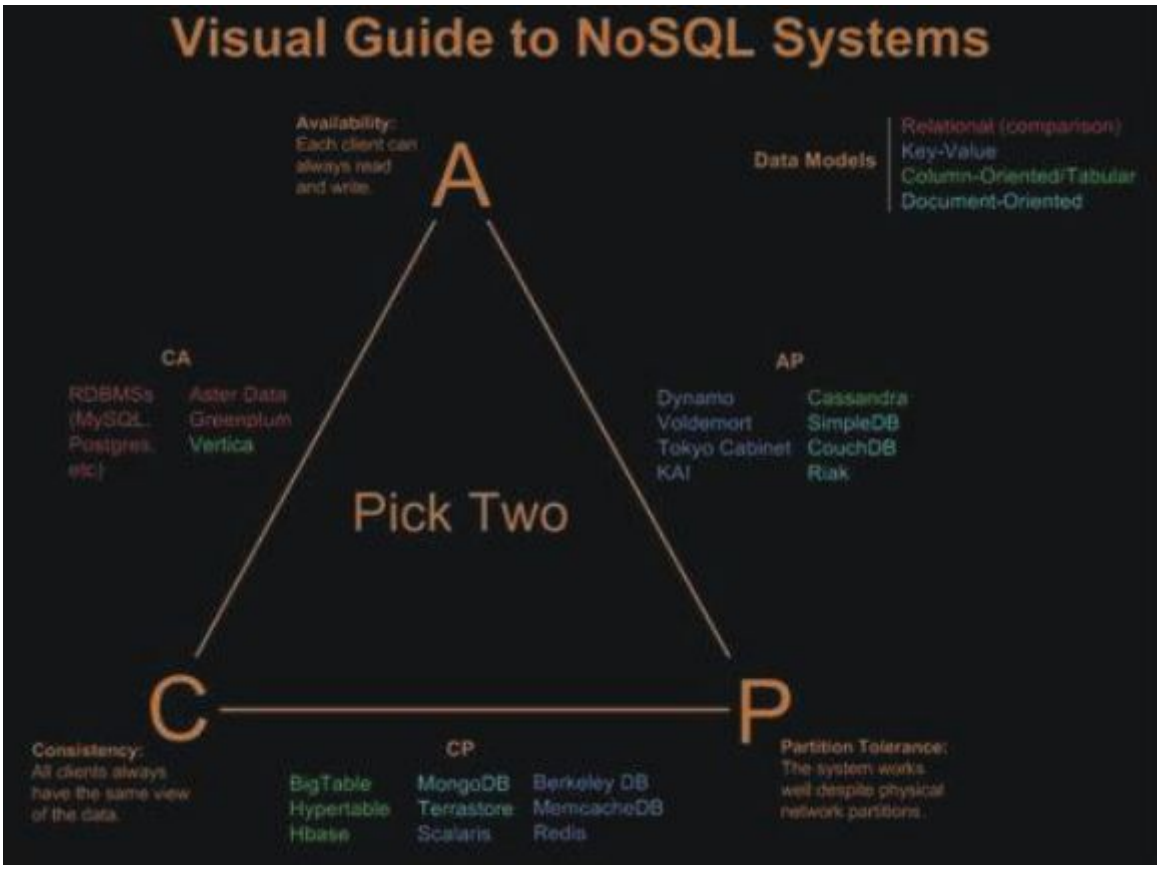
JSON (acrónimo de **JavaScript Object Notation**, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera (año 2019) un formato independiente del lenguaje.

4) ¿Qué es una base de datos orientadas a grafos?

Por definición, una base de datos orientada a grafos es cualquier sistema de almacenamiento que permite la adyacencia libre de índice. Esto quiere decir que cada elemento contiene un puntero directo a sus elementos adyacentes por lo cual no es necesario realizar consultas por índices
Este tipo de base de datos está diseñada para los datos cuyas relaciones son bien representadas en forma de grafo, o sea, los datos son elementos interconectados con un número no determinado de relaciones entre ellos. La información a gestionar por este tipo de almacenamiento pudiera ser las relaciones sociales, el transporte público, mapas de carreteras o topologías de red, entre otros ejemplos.

5) ¿A qué hace referencia el teorema de Brewer?

El teorema de Brewer hace referencia a la coherencia (Consistency), disponibilidad (Availability) y tolerancia a la partición (Partition Tolerance). Plantea que en los sistemas distribuidos sólo podemos tener dos de las tres garantías (la C, la A o la P), y por lo tanto es preciso elegir la más importante. Es preciso tener en cuenta que si lo que más importa es la coherencia, entonces se debe optar por una base de datos relacional. El siguiente esquema explica gráficamente el teorema de Brewer



Clase N°8:

1) Explique la arquitectura centralizada y la Cliente Servidor.
2) Explique qué tipo de arquitectura de servidores existen.
3) Explique a que hace referencia en una arquitectura compartida el término “Memoria Compartida, Disco Compartido, Sin compartimiento y Jerarquía”

1)

Arquitectura Centralizada y Cliente – Servidor:

Los **sistemas de bases de datos centralizados** son aquellos que **se ejecutan en un único sistema informático sin interaccionar con ninguna otra computadora**. Tales sistemas comprenden el rango desde los sistemas de bases de datos monousuario ejecutándose en computadoras personales hasta los sistemas de bases de datos de alto rendimiento ejecutándose en grandes sistemas. Por otro lado, **los sistemas cliente –servidor tienen su funcionalidad dividida entre el sistema servidor y múltiples sistemas clientes.**

Se distinguen dos formas de utilizar las computadoras: como **sistemas monousuario** o **multiusuario**. En la primera categoría se encuentran las computadoras personales y las estaciones de trabajo. Un sistema monousuario típico es una unidad de sobremesa utilizada por una única persona que dispone de una sola CPU, de uno o dos discos fijos y que trabaja con un sistema operativo que sólo permite un único usuario. Por el contrario, un sistema multiusuario típico tiene más discos y más memoria, puede disponer de varias CPU y trabaja con un sistema operativo multiusuario. Se encarga de dar servicio a un gran número de usuarios que están conectados al sistema a través de terminales.

2)

Los sistemas servidores pueden dividirse en servidores de transacciones y servidores de datos.

- Los **sistemas servidores de transacciones**, también llamados **sistemas servidores de consultas**, proporcionan una interfaz a través de la cual los clientes pueden enviar peticiones para realizar una acción que el servidor ejecutará y cuyos resultados se devolverán al cliente. Normalmente, las máquinas cliente envían las transacciones a los sistemas servidores, lugar en el que estas transacciones se ejecutan, y los resultados se devuelven a los clientes que son los encargados de visualizar los datos. Las peticiones se pueden especificar utilizando SQL o mediante la interfaz de una aplicación especializada.

- Los **sistemas servidores de datos** permiten a los clientes interaccionar con los servidores realizando peticiones de lectura o modificación de datos en unidades tales como archivos o páginas.

  Proporcionan facilidades de indexación de los datos así como facilidades de transacción de modo que los datos nunca se quedan en un estado inconsistente si falla una máquina cliente o un proceso.

3)

Memoria compartida: En una arquitectura de memoria compartida **los procesadores y los discos tienen acceso a una memoria común**, normalmente a través de un bus o de una red de interconexión. El beneficio de la memoria compartida es la **extremada eficiencia en cuanto a la comunicación entre procesadores**. Cualquier procesador puede acceder a los datos de la memoria compartida sin necesidad de la intervención del software.

Las arquitecturas de memoria compartida suelen dotar a cada procesador de una memoria **caché muy grande** para evitar las referencias a la memoria compartida siempre que sea posible. No obstante, en la caché no podrán estar todos los datos y **no podrá evitarse el acceso a la memoria compartida.**

Disco compartido: En el modelo de disco compartido **todos los procesadores pueden acceder directamente a todos los discos a través de una red de interconexión**, pero los procesadores tienen memorias privadas.Las arquitecturas de disco compartido ofrecen dos ventajas respecto de las de memoria compartida. Primero, el bus de la memoria deja de ser un cuello de botella ya que cada procesador dispone de memoria propia. Segundo, esta arquitectura ofrece una forma barata para proporcionar una cierta **tolerancia ante fallos**: si falla un procesador (o su memoria) los demás procesadores pueden hacerse cargo de sus tareas ya que la base de datos reside en los discos, a los cuales tienen acceso todos los procesadores.

Sin compartimiento: En un sistema sin compartimiento **cada nodo de la máquina consta de un procesador, memoria y uno o más discos**. Los procesadores de un nodo pueden comunicarse con un procesador de otro nodo utilizando una red de interconexión de alta velocidad. Un nodo funciona como el servidor de los datos almacenados en los discos que posee. El modelo sin compartimiento salva el inconveniente de requerir que todas las operaciones de E/S vayan a través de una única red de interconexión, ya que las referencias a los discos locales son servidas por los discos locales de cada procesador; solamente van por la red las peticiones, los accesos a discos remotos y las relaciones de resultados.

Jerarquía: La arquitectura jerárquica **combina las características de las arquitecturas de memoria compartida, de disco compartido y sin compartimiento.** A alto nivel el sistema está formado por nodos que están conectados mediante una red de interconexión y que no comparten ni memoria ni discos. Así, el nivel más alto es una arquitectura sin compartimiento. Cada nodo del sistema podría ser en realidad un sistema de memoria compartida con algunos procesadores. Alternativamente, cada nodo podría ser un sistema de disco compartido y cada uno de estos sistemas de disco compartido podría ser a su vez un sistema de memoria compartida.