

Bases de datos avanzadas

Contents

Bases de datos avanzadas	1
Evaluaciones	2
Temas	2
Primer Parcial	2
Datos sensibles	2
DBMS	3
Dato	3
Registro	3
Entidad (o tabla)	3
Información	3
Objetos	4
ORM	4
DBMS (SMBD en es.)	4
Funciones	5
Términos	5
Proyecto	5
Respaldo	5
Replicación	5
Concurrencia	5
Transacciones	5
Componentes	5
IP y puerto	6
Join	6
Tipos de SQL	7
DDL	7
DML	7
DCL : Son de control	7
TCL : Transacción	7
Seguridad, Permisos, Privilegios	7
Incidencias evitables	8
Protección	9
Metodología	9
Operaciones por prioridad	10
Privilegios y roles	10
Repaso	10
Normalización	10
Paginas web	11
Vistas	11
Procedure	11
Triggers	12
Transacciones	13
Problemas	13
Solución	14
Bloqueos	14
Recuperación	14

Semaforo	14
Fragmentación	15
Replicación	15
Reparación	16
Compactación	16

—

Evaluaciones

1. Parciales | 1.1 con el examen ... 80 % | 50 % ? 1.2. Otra cosa ... 20 % |
3. Proyecto final (no acumulativo) ... 25 % | 25 % – 2 sem antes del examen final
 - 3.1 Desarrollar un dashboard de información sobre una base de datos. En MySQL o Postgres.
 - 3.2 No nos va a dejar que nos lo imaginemos, el nos va a dar un ejemplo exacto de renta de videos.
 - 3.3 Se califica:
 - 3.3.1 Los datos pueden ser de donde sea pero tienen que ser masivos
 - 3.3.2 Identificar los queries importantes que se deben mostrar al usuario:
 - No es un sistema que administre datos, solo es una hoja con graficas que muestra el comportamiento del negocio
 - El chiste es hacer que los datos le hablen al dueño de la empresa, hacer visualizaciones agradables y eso
 - Quien compra mas, edades, donde, cómo no el otro lado.
 - 3.4 Pasos
 - 3.4.1 Determinar una necesidad, o algo que a nosotros nos guste mucho
 - 3.4.2 Proponer la respuesta, como el intentar indicar las esquinas donde mas violencia hay y de que tipo.
 - 3.4.3 Pensar en hacerlo lo más amigable a alguien que lo que quiere es ver datos para tomar decisiones.
 - 3.5 Siempre tener una copia local, no vaya a ser que falla la nube.
 - 3.6 Lenguaje el que queramos.
 - 3.7 Seguridad de las personas o salud
4. Examen final ... 25 % | 25 %

Bases de datos estructuradas con SQL, el proyecto en equipo de 2 a 3.

Temas

1. Queries, queries con joins, triggers, normalización.
2. Replicación, compactación.
3. Bases de datos distribuidas, orientadas a objetos
4. Presentaremos temas relacionados a: Data mining, data warehouse, big data

Primer Parcial

Datos sensibles

Para poder datos sensibles se necesita autorización expresa de la persona y cifrar la base de datos que los contenga.

El genero no es un dato sensible, pero si personal.

1. Datos personales: Nos permiten identificar a una persona, se pueden usar siempre y cuando le digamos al usuario cómo vamos a usar sus datos.
2. Datos patrimonial: Datos personales como cuentas de banco, tarjetas, números, asociaciones, autos, donde se involucra dinero. Esos datos requieren un permiso firmado por autografía de la persona.
3. Datos sensibles: Permiten determinar preferencias de la persona, esto puede provocar discriminación del individuo, por lo que se requiere tener cuidado.

En kaggle podemos sacar datos

DBMS

Es un programa que permite llegar a los datos, manejarlos.

Lo que se suele conocer como base de datos en realidad se refiere al sistema manejador de bases de datos (DBMS por sus siglas en inglés). Los datos se almacenan, en disco en archivos binarios. El sistema manejador es el que se encarga de manejar los datos, así nosotros podemos manejarlos de forma sencilla.

El DBMS tiene distintos componentes en su arquitectura:

- BD: El archivo donde se guardan los datos
- Manejador de BD: Es el programa que se ocupa para administrar los datos, ejemplo Maria, MySQL, etc. Es un servicio, un demonio.
- Administrador del manejador de BD: Es el programa con el que nos comunicamos con el servicio, es el cliente.

Daemon, que es un servicio, viene del griego. Recordemos que a los muertos se les ponían piedrecas en los ojos para Caronte, quien no tenía para pagar se quedaba en el lado de los vivos, pero muerto, por lo que los vivos no lo ven pero tampoco está muerto. Por eso un servicio se le llama daemon, porque los vivos no lo ven

Dato

El dato es algo que podemos corroborar con la realidad. Puede ser:

- Hecho: *hoy es X*
- Antecedente
- Fundamento

Los datos están asociados con un *nombre*, un *tipo* y un *tamaño*.

Registro

Un registro es un conjunto de datos, con una relación entre ellos.

Entidad (o tabla)

Donde se guardan los registros.

Información

Conjunto de datos ordenados en cierto:

- Formato: Como están organizados los datos. La manera en que se almacenan es diferente, lo que permite realizar distintas operaciones (?)
- Contexto:
- Disminuye la incertidumbre: Ayuda al que otro entienda bien. Que no hay certeza, no se entiende bien.
- Para la toma de decisiones: Si no hay decisión entonces no es información, si se escucha y no se actúa no es información.

La cultura es contexto, debido al contexto que tenemos las mujeres en occidente se casan de blanco, en cambio en asia es de rojo. Por el contexto se ve el mundo de forma totalmente diferente, con otra perspectiva.

La información es subjetiva por definición, está rodeada de contexto. Dependiendo de la persona, el mismo hecho puede representar información y para otros no.

Las bases de datos tienen datos, no información. Podemos explotar las bases de datos para ayudar al otro a constituir en él información. Cuando un bot toma decisiones basado en datos si se constituye información, al final fue un humano quien lo decidió y quien actúa por medio de un agente.

Objetos

- **Tabla/Entidad:** Con objeto nos referimos a elementos que la base de datos sabe almacenar.
- **Vista:** Es un sql que se almacena en la base de datos y se muestran como tabla. No se puede escribir en una vista, pero si se puede usar como una tabla.

Las vistas son solo de escritura, se puede inventar campos, no está *materializado* en ningún archivo o tabla, por lo que no hay dónde escribir.

- **Stored procedures:** Código SQL almacenado en el servidor que no se muestra como tabla, sino que se ejecuta, es como una función, así que admite in y out.
- **Jobs:** Como un *chron* en Unix. Tiene un disparador (*trigger*) y un código que ejecuta. El código puede ser un *stored procedure*. Para respaldos, cálculos, etc.
- **Triggers:** Otro tipo de disparadores, asociado a eventos de lectura/escritura. Se asocia a las operaciones que pueden afectar la integridad. Por lo tanto, se suele usar cuando tenemos operaciones que pueden alterar la integridad referencial de la base de datos. Tiene disparador (evento io) y código.
- **Constraints:** Reglas que podemos tener de relación entre tablas, que dejan condiciones para las operaciones con los datos que mantengan la integridad de la base de datos.
- **Formulario:** Solo lo tiene Access, puedes guardar y mostrar/preguntar al usuario que rellene bases de datos.
- **Usuarios:** Los '*usuarios*' son contraseña/usuario configurado para dar permisos específicos a quienes ingresen a la base de datos. Lo que se refiere con que no son los mismos usuarios es que no hay un usuario 1:1.

ORM

Historia: Parte de las prácticas que realizan las bases de datos para conservar sus clientes. Por eso existen bases de datos con planes tipo: Hago el estándar de SQL 92 y *esto más*. A estas modificaciones se les llama dialectos, haciendo a la vez más difícil migrar entre bases de datos.

Patrones de diseño: Debido a los patrones de diseño se pueden llegar a formas comunes de crear software, estos son los patrones de diseño.

No conviene realizar bases de datos que empleen la tecnología específica de la base de datos, como triggers y procedures. Un *Object Relational Mapping* provee una forma en que podemos usar con la misma interfaz distintas bases de datos.

Si mi aplicación emplea un ORM podemos desarrollar aplicaciones sin necesidad de casarse con una base de datos específica.

DBMS (SMBD en es.)

Es un conjunto de varios procedimientos/programas para que podamos recuperar ({} SELECT), describir (CREATE/ALTER) y manipular (INSERT/UPDATE/DELETE) datos que se almacenan en la base de datos.

Esto de forma que se mantenga la:

- integridad: que se mantenga válida la información y sus relaciones
- confidencialidad: que solo pueda acceder quien tiene permiso a lo que tiene permiso
- seguridad: asegurar los datos.

Funciones

Lo mismo, describir, manipular y utilización (accessible) de los datos. Si los datos no llegan no hay chiste.

- Interacción con el fs: Antes se especificaba SO pero ahora se confía en el nativo
- Implantación de seguridad: Lo mismo de las operaciones peligrosas
- Seguridad: Verifica que los accesos a la base de datos estén realizados por los usuarios. Verificar brechas de seguridad, monitorear.

El 60% de los ataques en las empresas son de empleados

Términos

- **Integridad referencial:** Hacer referencia a otro dato, lo que implica que el otro existe. Que las llaves foráneas/primarias si estén relacionadas.
- **Operaciones fundamentales:** Leer, insertar, modificar, borrar

Proyecto

Mostrar controles, todo fácil de digerir, chance simular. Lo importante es extraer los datos.

Respaldo

Tenemos un disco, hacemos una copia de respaldo, almacenamos las copias en lugares seguros, distribuidos, fáciles y rápidos de acceder.

En la nube puede ser un lugar pero hay que considerar los tiempos de subida/bajada.

Se debe ejercitar el poder respaldar, simulacros donde se pruebe la facilidad de respaldo, todo el ciclo del mismo.

El respaldo se debe hacer considerando la cantidad de información que entra, las operaciones importantes que suceden con la importación, etc. Cada que se haga algo relevante.

Replicación

En el caso de sistemas con cantidades de eventos relevantes altas en muy poco tiempo se debe tener *replicación*, otra(s) base(s) de datos que sean espejo en tiempo real.

Concurrencia

El SMBD se encarga de manejar la concurrencia de forma que no haya colisiones, donde se corrompen los datos desde el nivel de memoria.

Transacciones

Una transacción en el contexto de la base de datos es un conjunto de tareas para la base de datos. O se hacen exitosamente todas las tareas o se cancela el grupo como conjunto.

Componentes

- Tenemos el **gestor de seguridad**, que valida que los usuarios (*de acceso*) pueden entrar cuando deben de entrar
- El **gestor de consultas** que se encarga de recibir las consultas y los almacena en una cola
- El **optimizador de consultas** verifica si puede modificar los queries (sin cambiar su efecto) para hacer el query más rápido, así podemos atender más consultas en menos tiempo.

- El **planificador**, o *scheduler*, el orquestador de tareas se encarga de dar prioridad a las consultas, es decir, influir en el orden en que las consultas se ejecutan para evitar colisiones, optimizar los recursos y reducir el tiempo de ejecución.
- **Procesador de consultas/transacciones** ejecuta el query.
- **Gestor de archivos**, que se encarga de administrar los archivos, en estos tiempos no se nota tanto pero en la antigüedad optimizaba la escritura en momento adecuado.
- **Buffers**: Transfiere datos entre memoria y secundarios, los buffers son el paso intermedio, si se apaga el servidor es necesario que se almacene, puede ser parte de la RAM.
- El **Gestor de recuperación** garantiza la consistencia de la base de datos.

Los SMBD hoy en día no se almacenan en SSDs, porque son caros y no alcanzan el tamaño que si pueden almacenar discos de estado sólido. Además se puede utilizar el esquema *raid* que permite almacenar copias entre múltiples discos para siempre asegurar la información.

IP y puerto

El puerto da un identificador a los servicios y programas para que puedan recibirse y enviarse los paquetes asignando a un puerto distinto cada aplicación.

Join

Por medio de un producto cruz:

```
SELECT
    productCode, productName,
    productlines.productLine,
    textDescription
FROM products, productlines
WHERE
    (productlines.productLine = 'Trains' OR productlines.productLine = 'Planes')
    AND productlines.productLine = products.productLine;
```

En las bases de datos los AND son como multiplicación, los OR son como suma, entonces no es lo mismo $(a + b) * c$ o $a + bc$

Siguiendo con eso el sql anterior se pudo haber escrito como:

```
SELECT
    productCode, productName,
    productlines.productLine,
    textDescription
FROM products, productlines
WHERE
    (productlines.productLine IN ('Trains', 'Planes'))
    AND productlines.productLine = products.productLine;
```

A la hora de hacer el proyecto vamos a querer acumular de acuerdo a la característica, max, min, sum, count, etc

```
SELECT
    COUNT(1) as 'Number',
    AVG(buyPrice) as 'Buy Avg'
FROM products, productlines
WHERE
    productlines.productLine IN ('Trains', 'Planes')
    AND productlines.productLine = products.productLine
GROUP BY
    productlines.productLine;
```

Otro ejemplo:

```
SELECT
  COUNT(1) AS Numero, productLine
FROM products
GROUP BY productLine
ORDER BY `Numero` DESC
```

Transacción: Se ejecutan todos o ninguno, bloque de instrucciones

Amenazan la integridad: Delete, drop, update

Tipos de SQL

1. DDL: Data Definition Language, permite construir nuevos objetos, elementos de la base de datos.
2. DML: Todos los comandos de SQL que nos permiten administrar los datos en si
3. DCL: Para definir permisos, de control
4. DTC: Control de transacción, lo necesario para decir un conjunto de operaciones.

DDL

- CREATE

DML

- ALTER: Permite alterar un dato ya existente, si cambiamos un dato, hay que encargarnos de que en todos lados esté congruente
- DROP: Borra contenido y tabla
- TRUNCATE: Si queremos borrar todo el contenido de una tabla, no la tabla en si
- LOCK TABLE: Mientras esté bloqueado un elemento no se puede acceder mientras este bloqueado
 - Compartido: Permite solo lectura
 - Absoluto: No permite hacer nada
 - Se puede usar al hacer transacciones, al generar indices, etc.

DCL : Son de control

- GRANT: Da privilegios a un usuario para que pueda realizar ciertas acciones
- REVOKE: Elimina privilegios para un usuario

Alternativa a borrar en sí podemos usar banderas que indican si una cosa está oculta, en realidad borrar no hace nada, solo se borran los datos en si cuando se usa PACK, que elimina los bits vacios

TCL : Transacción

Las operaciones se hacen en un espacio de memoria aparte, que hace que los cambios no se efectuen hasta que demos la orden

- COMMIT: Efectua en la base de datos el conjunto de cambios, todas las operaciones que hicimos, materialízalas.
- ROLLBACK: Descarta los cambios en el espacio temporal de una transacción

Seguridad, Permisos, Privilegios

La seguridad es necesaria en toda la cadena que forma una organización, incluyendo en las bases de datos.

En los 90 la internet se hace publica/comercial. Con esto, las amenazas que se presentaban aumentaban, pues ahora hay comunicación con fuentes externas no confiables.

Recordemos que el DBMS garantiza:

- Integridad: Que la información sea verídica y correcta
- Disponibilidad: La información es accesible
- Confidencialidad: Solo quienes tienen que lo pueden ver

Estos principios son amenazados por:

- Ataque por fuerza bruta: Intentar determinar algo (como contraseña) por medio de prueba y error de forma automatizada. Por lo general se hace con un diccionario de valores, similar al de hash.
- Robo por sniffing: Así como Wireshark, que permite ver el tráfico que hay en un medio. Aunque no podemos prohibir los *sniffers*, podemos cifrar de punta a punta los datos que se transportan, como usando HTTPS. Hoy en día es muy fácil y está normalizado.
- Propagación por URL: Mandar información sensible como sesiones mandadas por URL, permite identificarse como necesario.

Contexto propagación por URL

HTTP sirve para la transferencia de archivos (información), de hecho, originalmente estaba pensando para ser del estilo *broadcast*, no estaba pensado para saber quién se está conectando, solo para enviar datos.

Otro ejemplo contrario es FTP, que es un servicio con *estado*, es decir que sabe a quién se conecta y que puede hacer. HTTP es entonces un servicio sin estado.

Para lograr automatización se invita el concepto de *sesión*, que es un número que se intercambia constantemente para hacer la identificación del usuario y servirle los datos.

Las sesiones se ocurren poner los datos del estado en una cookie. Una cookie es un archivo de texto, asociado a un dominio, que puede leer y modificarla. Cada que hay una solicitud se mandan/intercambian las cookies de forma constante.

A algunos desarrolladores se les ocurrió poder mandar la sesión por la URL, lo que abre una infinidad de posibilidades para ataques. ***Hay que asegurarnos de que estos datos solo se transfieren por cookies***

- Robo en servidor compartido: Ponemos una máquina grande y te rento un cachito de la misma, en este método original no había forma de virtualización ni de contener, lo que permite a los demás hacer acciones que permiten realizar operaciones como otros usuarios, usando sus recursos. Un ambiente compartido da la posibilidad de que pase.
- Robo por Cross-Site Scripting: En cuanto tenemos formularios, se implica que los datos serán procesados en el servidor. En este tipo de formularios se pueden enviar comandos, que buscan atacar el servidor inyectando código malicioso. Esto se evita sanitizando, entendiendo la gramática de los datos ideales y no aceptando datos con caracteres que puedan ingresar código.
- Inyección de SQL: Similar, se inyecta SQL que es procesado como llegó en el servidor, mostrando datos que no se deberían ver.
- Cross-Site Request Forgery: A los formularios le ponemos un número de folio, si no generamos nosotros ese identificador, no aceptamos los datos.

Incidencias evitables

- Passwords débiles, por defecto: Es totalmente evitable y debería hacerse a toda costa.

Julian Assange, fundador de WikiLeaks, logró entrar a servidores “ultra-protegidos” sin contraseña

- Preferencia de privilegios de usuario por privilegios de grupo: Tenemos un usuario al que le asignamos privilegios que son de acuerdo a un grupo, cuando quitamos al usuario del grupo nos aseguramos que pierde todo el privilegio del grupo de privilegios.
- Características de base de datos innecesarios: Poner todo los plugins de una base de datos solo agrega más piedras a la carga, es mejor dejar lo que se usa y se sabe como se usa.
- Configuración de seguridad ineficiente: Si a todos le damos permiso de lectura/escritura, entonces todo puede valer. Dale exactamente lo que necesita a cada usuario.
- Desbordamiento de Buffer: Con un sistema mal configurado, estamos metiendo más datos de los necesarios y de un tipo que no corresponde. Habría que ver la manera de evitar ese tipo de problemas, como restringir el tamaño.
 - Hacer que el sistema sea tolerante a fallos, y que verifique.
 - Que el error no salga en pantalla, que solo mande un error, cuando estemos en desarrollo habilitamos todas las alertas y mensajes para los desarrolladores, pero en producción que solo sea lo necesario pensando en el usuario final.
- Escalada de privilegios
- Ataque de denegación de servicio: Múltiples solicitudes por recursos que saturan la capacidad del servidor de dar servicio, este tipo de ataques pueden ser usados para generar logs de error hasta llenar el disco, por ejemplo, buscar archivos en disco (lento) y demás.
- Datos sin cifrar: Los sensibles son aquellos que nos permiten segmentar de la población, como gustos, religión, etc. Podemos tener esos datos con el consentimiento de la persona a la que pertenecen y debemos cifrarlo. Si llegan a hackear nuestra plataforma y se comprueba que los datos sensibles no estaban cifrados, conlleva a multas. El INAI es el órgano que verifica esto.

Protección

Para protegernos debemos:

- Verificar accesos, hacer la administración correcta de los permisos que tienen los miembros de una organización.
- Inferencias: No dar ningún tipo de información que permita al atacante saber el stack de tecnología que se usó para la plataforma, que no le permita *inferir*.
- Ingeniería social: Phishing, dejar USB con Malware en el sitio de trabajo, correos, etc. Para prevenir esto es necesario educar a los miembros de la organización.
- Flujo: En los sistemas que tengamos debemos saber el flujo de datos que hay y monitorear los puntos del camino para asegurarnos que se mantiene intacto y eficaz.
- Cifrar datos: Cifrar nunca está mal, mínimo lo sensible. Las contraseñas se almacenan directamente cifradas, no hay NINGUNA necesidad de almacenar la contraseña en sí, sino que solo guardar su versión cifrada, usar sistemas de digestión (que lo cifren/procesen de forma)
- Seguimiento del rastro: Debemos tener auditorías en el hardware.

Metodología

1. Identificar su sensibilidad: Analizar el sistema y determinar las partes que son más sensibles, como lo es la tabla con datos sensibles.
2. Evaluación de la vulnerabilidad/configuración: Teniendo en cuenta los componentes, su contexto, localización, acceso y demás variables.
3. Endurecimiento: Arreglar los posibles huecos, hacer las medidas necesarias para que el sistema como un todo sea más seguro.
4. Auditar: Evaluar las medidas, el resultado de los cambios en el ambiente/sistema/organización, utilizar los nuevos recursos/medidas para ampliar el conocimiento sobre el sistema.
5. Monitoreo: Ver en tiempo real la actividad del sistema, detectar señales/eventos, usar herramientas que analicen, hagan de forma automatizada/efectiva el monitoreo.

6. Pistas de auditoría: Actuar para resolver lo observado en el monitoreo.
7. Autenticación, control de acceso y gestión de derechos: IAM, gestion de permisos, acceso, etc.

Es un proceso que nunca termina, debe de hacerse de forma continua.

Si no mides, no sabes lo que está pasando. Hace unos años estaba pensada la seguridad como forma de ver que nadie entrara, pero esto no es así, el chiste es que si entran (y lo harán), reducir el daño posible que puedan hacer.

Operaciones por prioridad

1. Eliminar
2. Alterar, no cualquiera debe de poder modificar la estructura del sistema
3. Relacionar, agregar CONSTRAINTS, lo que puede deshabilitar operaciones de insertar/borrar/etc
4. Indizar, es un proceso caro y puede alentar el servicio, no cualquiera debe de poder hacerlo
5. Borrar (un registro)
6. Actualizar
7. Insertar
8. Leer

Un usuario máximo debe de poder llegar hasta actualizar/borrar.

Privilegios y roles

Los privilegios son aquello que puede hacer el usuario.

Repaso

1. Lo que se degrada es la confidencialidad, disponibilidad, integridad de la base de datos.
2. El ataque de fuerza es difícil pero funciona, porque seguimos poniendo contraseñas malas
3. El robo por *sniffing* requiere primero que puedas meter un *sniffer*, que se pudo meter a la red.
4. Cookie, archivo que va y viene y acompaña todo el tráfico del usuario
5. Hacer un ataque por sitio compartido hoy en día está difícil
6. XSS: Cross site scripting, que es inyección de código malicioso, lo que se previene sanitizando, haciendo que el código no se ejecute/cambiándolo.
7. Muchas solicitudes hasta saturar el servidor, solicitudes incorrectas. DDoS, es lo mismo que DoS pero Distribuido, de ahí la D extra, es decir muchos dispositivos.
8. Cifrar: Cifrar implica que se puede descifrar, que no es lo mismo a un digestor, que no tiene vuelta atrás.
9. Cifrados:
 - Simétrico: Cifras y descifras con la misma llave, es mucho más rápido, hay combinaciones de cifrados para optimizar el proceso, por ejemplo usar un cifrado simétrico pero para compartir la llave única se usa un mecanismo que usa cifrado asimétrico
 - Asimétrico: Llave pública y privada

Que no diga alarcon

Normalización

Busca reducir la redundancia, que los datos estén una sola vez presentes. Las formas normales conocidas son 5, aunque en la práctica llegamos hasta la 3era forma normal. La 4ta y 5ta normalización causa sobre-normalización, dando efectos absurdos, como una tabla por registro.

- 1era Forma Normal: *Una llave para cada registro*. Lo que buscamos es que en cada registro haya una llave única, lo que puede implicar llaves compuestas, la primera forma normal pide que cada registro tenga una llave que sirva para encontrar un solo registro.
- 2da Forma Normal: *Sacar todo lo que no depende de la llave compuesta*. Sin dependencias parciales de llaves concatenadas. Si una tabla tiene llaves compuestas, todos demás campos de dicha tabla tienen que depender de dicha llave forzosamente, por los n-campos de la llave compuesta. Es por eso que si un campo no depende de, por ejemplo, los 3 campos que forman la llave compuesta, sino que solo por 2, entonces no se cumple la regla.
- 3ra Forma Normal: *Sacar todo lo que no depende de la llave*. Lo mismo pero con llaves simples

Pone el ejemplo de una llave (num_factura, prod_num), el campo fecha no depende de ambos campos, depende exclusivamente de num_factura, de forma similar el precio depende exclusivamente del producto, no de la factura.

Los campos derivados no van, máximo una vista. Es necesaria la normalización en las bases de datos relacionales, es sencillo y nos permite un mejor manejo de los datos.

- Si hay una relación muchos a muchos, siempre hay una tabla en medio que resuelve la relación, aunque surge de forma natural con el simple hecho de normalizar.

Paginas web

Para poder hacer una pagina hay dos cosas fundamentales:

- Frontend: HTML, JS, CSS
- Backend:
 - WebServer: Apache, Nginx
 - App: JS, PHP, Python
 - Data: SQL

Vistas

Creó una vista para mostrar por cada orden la cantidad a pagar.

```
CREATE VIEW salesPerOrder AS
SELECT orderNumber, (quantityOrdered * priceEach) AS Total
FROM orderdetails
GROUP BY orderNumber
ORDER BY total DESC;
```

Podemos editar el código de la vista si le damos click en el menú

Procedure

El stored procedure no tiene por qué devolver un valor

Subió el thread stack al doble de mysql

```
DELIMITER //
```

```
CREATE PROCEDURE GetAllProducts()
BEGIN
    SELECT * FROM products
END //
```

```
DELIMITER;
```

Triggers

Ejemplo, ahora va a crear una tabla nueva que va a registrar todos los eventos en cambios sobre la tabla employees.

Primero crea la tabla de employees:

```
CREATE TABLE employees_audit(  
  id INT AUTO_INCREMENT PRIMARY KEY,  
  employeeNumber INT NOT NULL,  
  lastname VARCHAR(50) NOT NULL, /* ? cambio de nombre */  
  changedate DATETIME DEFAULT NULL, /* Fecha que se cambio */  
  action VARCHAR(50) DEFAULT NULL /* Que operación se hizo */  
);
```

Y ahora si va a crear el TRIGGER para cuando se integre la integridad de las tablas:

```
CREATE TRIGGER before_employee_update  
BEFORE UPDATE ON employees -- Ejecutar antes de que se haga el cambio en si  
FOR EACH ROW -- Para toda la tabla  
INSERT INTO employees_update  
SET action = 'update',  
    employeeNumber = OLD.employeeNumber, -- Valor antes de actualizarse  
    lastname = OLD.lastname,  
    changedate = NOW()  
;
```

Ahora usará un procedure

```
DELIMITER //
```

```
CREATE PROCEDURE InsLog(  
  IN pEnum INT, -- IN porque entra  
  IN PLName VARCHAR(50),  
  IN pDate DATETIME,  
  IN action VARCHAR(50)  
)  
BEGIN  
  -- Aqui va el insert/acción  
  INSERT INTO employees_audit  
  (employeeNumber, lastname, changedate, action)  
  VALUES  
  (pEnum, pLNAME, pDate, pAct);  
END //
```

```
DELIMITER ;
```

Y puede probar la macro:

```
CALL InsLog(1, 'test'. NOW(), 'test');
```

Y modifica el trigger para que quede:

```
CREATE TRIGGER before_employee_update  
BEFORE UPDATE ON employees -- Ejecutar antes de que se haga el cambio en si  
FOR EACH ROW -- Para toda la tabla  
  CALL InsLog(OLD.employeeNumber, OLD.lastname, NOW(), 'update')  
;
```

Entonces:

- Validar para el tipo de dato específico que va a leer

- Usar Expresiones regulares
- Protección contra XSS

Transacciones

Paralelo: Al mismo tiempo Concurrente: Puede que se de el caso, donde las transacciones parciales pueden influir en el resultado final cuando se están procesando usando concurrencia

Características:

- Atomicidad: Busca que la transaccion sea mas corta, que antes y despues la base de datos tenga un estado consistente
- Consistencia:
- Aislamiento: Se queda cada transaccion en su propio ambiente aislado
- Durabilidad: Se hacen permanentes los cambios

Etapas:

- Inicia las transacciones
- Si hay error entonces hace rollback

La concurrencia se pone dificil cuando los recursos son limitados, y los clientes son muchos. Hacer más transacciones por minuto aumenta la productividad.

Para poder usar concurrencia se tiene que controlar, buscando que los flujos puedan viajar sin colisiones.

Estrategias:

- Pesimista: Pasamos uno por uno, porque asumimos que habrá problemas, entonces metemos a todos a una cola.
- Optimista: Nadie choca nunca, si lo hace solo lo repetimos. La unica forma de que se pudiera dar es que el cpu fuera infinitamente más rápido que la red, asume que no habrá cola.
- Mixto: Combina ambas,

Problemas

- Con dato temporal: Una transaccion T_1 realiza updates que después de otros pasos de micro-transacciones resulta en error y hace rollback. La transacción T_2 ontinua *sabiendo* que su valor leído de T_1 es correcto, aún cuando ya se ha des-hecho.
- Dirty read: T_2 realiza datos con un registro x , T_1 actualiza el valor de x , pero T_2 nunca se entera de este hecho. Por lo que no está tomando en cuenta que hubo operaciones que modificaron los valores, lo que genera en operaciones erroneas porque se están haciendo con valores sucios, invalidos, porque ya no reflejan el estado actual de los datos.
 - Si vamos a realizar estadísticas con valores reales, no deberíamos de hacer operaciones de este tipo cuando hay transacciones en el momento, hay que dejar que se terminen todas las transacciones para poder hacer la estadística.
 - O podemos hacer una copia de la base productiva, porque la copia sirve como fuente de verdad hasta un tiempo específico.
 - Si hacemos estadística sobre el servidor en producción también afectamos el desempeño del servidor en producción, lo que afecta a los usuarios
- No repeatable read: La transacción T_1 modifica x , T_2 lee el valor antes de modificarse, después de un rato vuelve a leer x , lo que resulta en dos valores leídos diferentes
- La concurrencia provoca errores

Solución

Para estas soluciones no tomamos un esquema optimista, ni tampoco pesimista, de forma que podamos encontrar mecanismos que disminuyan los errores.

- Semáforos: Controlan el flujo en los cruces que son importantes.
- Sellos de tiempo: Impiden acciones sobre los datos. El servidor, cada que actualiza un dato en la tabla (modificar/borrar) pone en un campo invisible un timestamp, al realizar otras operaciones verificamos que el último timestamp que nosotros conocemos es el mismo que el sello en la fuente de verdad. Si resulta que no tenemos lo último, entonces descartamos lo nuestro y tomamos los datos de la base de datos
- Multiversión: Exactamente el mismo concepto pero ahora con un contador que permite identificar de entre los cambios el más verdadero/actual.

Bloqueos

- Bloqueo compartido (*RWLock*): Todos pueden leer pero nadie (o solo uno) puede escribir.
- Bloqueo exclusivo (*Mutex*): Solo uno puede leer y escribir a la vez.
- Interbloqueo: Un recurso que se quedó bloqueado pero es compartido. Esto suele pasar cuando un proceso con acceso a un recurso lo deja bloqueado (p. ej. porque murió el proceso).
 - Los sistemas manejadores de bases de datos pueden matar procesos que están bloqueando el acceso a recursos.
 - Solo mata procesos que están a su cargo.

Recuperación

Cuando el servidor de la base de datos no puede realizar sus operaciones de forma correcta (p. ej. la degradación de un disco duro permite la lectura pero no escritura). Con recuperación se busca que el SMBD pueda solucionar los errores que encuentra (p. ej. marcando el sector a nivel de sistema operativo como no-escribible)

Hay algunas herramientas que ayudan a **prevenir** la pérdida de datos:

1. A nivel de software:

- Respaldos
- Anti-virus: quiere dañar
- Anti-malware: quiere usar el recurso para obtener beneficios, es más difícil de detectar (p. ej. ser usado como bot u obtener información)
- Detector y prevención de intrusos (p. ej. zuricata) leyendo logs y comportamientos anormales

2. A nivel de prodecimiento:

- Probar las pruebas de recuperación para verificar que en caso de un incidente si sean de confiar

3. A nivel de infraestructura: Prevenir involucra todo lo físico

- Localización, dependiendo de donde estamos y que riesgos hay
- Suministro de energía ininterrumpido
- Prevención de incendios
- Conexión siempre

Semaforo

Un proceso puede tener hilos de ejecución que se realizan de forma concurrente.

Antes de los hilos se clonaba el proceso entero, lo que quiere decir que también sus recursos. Con los hilos lo que se permite es que se puedan usar recursos compartidos de forma concurrente.

Los procesos tienen un cursor (un *thread*) que indica en qué punto de ejecución se encuentra el programa. Lo que pasa con los hilos es que se crea un stack para el hilo nuevo y se usa el mismo código y heap.

Si tiene su propio stack tiene su propia pila de llamadas, y pueden ejecutar su propio código. Eso sí, todos usan la misma memoria Heap y Código.

Cada uno de los threads se ven como un proceso diferente a nivel del sistema operativo, aunque se trate de distintos seguidores dentro de la aplicación.

Las secciones críticas son aquellas donde pueden existir colisiones, para proteger estas zonas críticas usamos algo conocido como Mutex (o RWLock).

El semáforo frena el flujo de todos de forma que solo *entre* uno a la vez.

Ejercicio: Implemente un semáforo para que dos o más páginas distintas no puedan leer o acceder a una página si otro está usando el recurso. Buscamos resolver el problema. Bloqueo compartido.

Socket: Unión de IP y puerto

Fragmentación

Normalmente se tienen dos tipos de fragmentaciones, se puede hacer para eliminar datos que no interesan (p. ej. para hacer que una vista no tenga información sensible). Es decir lo hacemos cuando conviene, por cuestiones geográficas, o de seguridad.

- Horizontal: Estamos llevando todas las columnas, es decir, quitamos por filas.

Si queremos hacer la operación inversa, deben ser las mismas columnas con el mismo tipo.

- Vertical: Movemos por columnas, nos llevamos las columnas seleccionadas de todas las filas.

Siempre tiene que tener un id para poder identificar cada fila, en caso de que no tomemos el id existente, entonces es nuestra responsabilidad generar un id

La operación inversa es un join

- Mixta: Ambas xd

Replicación

Hacemos respaldos para tener disponibilidad, fiabilidad. Hacemos un respaldo porque sabemos que los datos son valiosos y queremos asegurar que no los podamos perder.

Cuando los datos son muy valiosos y cambian a cada segundo, entonces toca hacerlo cada segundo. Queremos tener una copia en vivo de los datos.

Tener replicación nos permite tener disponibilidad, fiabilidad (alguno de los servidores encontrará errores), rendimiento, reducción de carga.

Si el servidor está al 70-80 de carga, tenemos que ver ayudarlo para que, en caso de carga extrema, se pueda mantener vivo el servicio.

Load balancer: Direcciona las solicitudes a los servidores de forma que se pueda optimizar la carga para ambos. Hay load balancers que pueden hacer que si solo es escritura, se mande la solicitud al servidor copia, y que lo que sea de escritura se mande al servidor primario.

El primario se encarga de avisar a los secundarios solo sobre los cambios en la información. Es decir, lo que hay en los logs (que son operaciones que modifican la información/estructura)

Reparación

Los SGBD tienen fallas, a nivel de hardware o software. La reparación se refiere a detalles pequeños que pueda resolver, que estén en su control.

Compactación

Ahora si el equivalente a la fragmentación de memoria. El comando PACK de una tabla se encarga de re-organizar los registros de forma que queden de forma continua todos los registros vivos.

Para hacerlo no deben existir operaciones a la vez sobre la tabla, el espacio en disco debe ser mínimo el mismo tamaño de la tabla que se está compactando.

Cuando el administrador realiza la operación de PACK (asumiendo que lo hace manualmente) debe primero realizar una de-fragmentación del disco, para que los archivos queden juntos y el proceso de compactación sea más rápido.

Hacer la compactación se debe realizar pero no de forma tan seguida que afecte uno de los principios de la base de datos (disponibilidad)

Las bases de datos, como toda la estructura, se puede optimizar basado en el análisis de las operaciones que se realizan sobre una base de datos.
