Bases de datos avanzadas

Contents

—

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 puden 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 autografia 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 eralida 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 manajador es el que se encarga de manejar los datos, asi 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 ponian ponedas en los ojos para Caronte, quien no tenia para pagar se queda en el lado de los vivos, pero muerto, por lo que los vivos no lo ven pero tampoco esta muerto. Por eso un servicio se le llama daemon, porque los *vivos* no lo ven

Dato
El dato es algo que podemos cororborar con la realidad. Puede ser:
- Hecho: *hoy es X*
- Antecedente
- Fundamento

Los datos estan 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 estan 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 deciciones: Si no hay decision entonces no es información, si se escucha y no se actua 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, esta rodeado 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 actua 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 ningun 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, calculos, 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 puden 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 especificos 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 estándard de SQL 92 y *esto más*. A estas modificaciones se les llama dialectos, haciendo a la vez más dificil 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 especifica.

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 valida 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 confia en el nativo
- Implantación se seguridad: Lo mismo de las operaciones peligrosas
- Seguridad: Verifica que los accesos a la base de datos estn 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 foranes/primarias si esten relacionadas.
- **Operacionesi fundamentales**: Leer, insertar, modificar, borrar

Proyecto
Mostrar controles, todo facil de digerir, chance simular. Lo importante es extrar 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 imporación, 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 exiosamente 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, asi 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 antiguedad 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, pude ser parte de la RAM.
- El ***Gestor de recuperación*** garantiza la consistencia de la base de datos.

Los SMBD hoy en dia 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 petmite almacenar copias entre multiples 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:

```sql
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:

```sql
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 caracteristica, max, min, sum, count, etc

```sql
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`: Permites solo lectura
  - `Absoluto`: No permites 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 oredn

- `COMMIT`: Efectua en la base de datos el conjunto de cambios, todas las operaciones que hicimos, materiaízalas.
- `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 veridica y correcta
- Disponibilidad: La información es accessible
- 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: Asi como wireshark, que permite ver el tráfico que hay en un medio. Aunque no podemos prohbir los *sniffers*, podemos cifrar de punta a punta los datos que se transportan, como usando `HTTPS`. Hoy en día es muy facil y esta 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 conectano, 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 ocurrio 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 ocurrio 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 maquina grande y te rento un cacho de la misma, en este método original no habia 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 codigo malicioso. Esto se evita sanitizando, entendiendo la gramática de los datos ideales y no aceptando datos con caracteres que puedan ingresar codigo.
- 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 numero de folio, si no generamos nosotros ese identificador, no aceptamos los datos.

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

Julian Asagne, 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.
- Caracteristicas 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.
- Configuracion 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. Habria 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: Multiples 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 organo que verifica esto.

Protección

Para protegernos debemos:

- Verificar accessos, hacer la administración correcta de los permisos que tienen los miembros de una organización.
- Inferencias: No dar ningun tipo de información que permita al atacante saber el stack de tecnología que se usó para la plataforma, que no le permita *inferir*.
- Ingenieria 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 està mal, mínimo lo sensible. Las contraseñas se almacanan 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 vulnerabilida/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 asi, 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 dificil 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 dia está dificil
6. XSS: Cross site scripting, que es inyección de codigo malicioso, lo que se previene sanitizando, haciendo que el codigo no se ejecute/cambiandolo.
7. Muchas solicitudes hasta saturar el servidor, solicitudes incorrectas. DDoS, es lo mismo que DoS pero Distribuido, de ahi 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:
   - Simetrico: 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 unica se usa un mecanismo que usa cifrado asimétrico
   - Asimetrico: Llave publica y privado

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 resgistro.* 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 unta 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 realación, 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 codigo de la vista si le damos click en el menu

Procedure

El stored procedure no tiene por qué devolver un valor

Subio 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 especifico que va a leer
- Usar Expresiones regulares

- Protección contra XXS

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

Caracteristicas:
- 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 isolado
- 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 leido 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 estadistica 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 leidos 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 a fuente de verdad. Si resulta que no tenemos lo ultimo, 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 perdida de datos:

1. **A nivel de software**:
   - Respaldos
   - Anti-virus: quiere dañar
   - Anti-malware: quiere usar el recurso para obtener beneficios, es más dificil 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 intraestructura**: Prevenir involucra todo lo físico
   - Localización, dependiendo de donde estamos y que riesgos hay
   - Suministro de energia 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 si, 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 criticas usamos algo conocido como Mutex (o RWLock).

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

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

Socket: Union de IP y puerto