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:

```
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 caracteristica, 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`: 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

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 SMBD 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 rapido.

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.

Parcial 3

Bases de datos distribuidas

Lo que logra es que los datos estén seguros en varios servidores, cuando se hacen consultas se buscará responder desde el servidor más cercano. El tiempo de respuesta es proporcional a el nodo donde está la información y donde se recibe.

Replicación en vivo para tener multiples nodos. Determinamos reglas con las cuales definimos qué datos se almacenan dónde.

Hay multiples servidores que se comportan como una sola base de datos.

- **Nodo**: Es na compputadora en la red
  - Hay modos especializados, aunque casi en todos serán SMBD
- Principios:
  - Autonomia local: Cada nodo tiene que garantizar el servicio global, usando sus recursos locales
  - Operacion continua: Deberia funcionar todo el tiempo, aun cuando falten nodos
  - Transacciones distribuidas: Que una transacción se distribuya entre nodos
  - Ind. Ubicacion
  - Ind. Fragmentación
  - Ind. Replicación
  - Ind. Hardware: No tienen por qué ser iguales en hardware
  - Ind. SO. red
  - Ind. de DBMS: Ni siquiera tienen por qué usar el mismo SMBD
- Nube: Se basa en la virtualización para poder generar tantas “maquinas” como sean necesarias, sea con maquinas virtuales o contenedores. Da flexibilidad porque puedes fácil levantar o tirar máquinas a voluntad y pagar por lo que usas. Puedes ejecutar datos o almacenar lo que quieras. Computo que da la posibilidad de usarlo segun la demanda.

En la nube podemos tener Bases de datos distribuidas con nodos flexibles, tambien podemos tenerlo en hosting pero los nodos son estáticos.

Data warehouse

Sistema de gestión, que almacena información, relacionado a business intelligence, para ayudar a tomar decisiones en una empresa, lo que implica el análisis de datos.

Una sola funete de verdad.

Extraction; Load; Transformation: La información para crear el data warehouse se extrae de las fuentes se carga en el warehouse. Ellos encontraron que se hace ELT en lugar de ETL, es decir, que se transforma en el warehouse mismo.

Grupos:
- Actualización
  - Tiempo real: Se actualiza cada que hay nueva información
  - Offline: Programadas cada tiempo
- Subgrupos
  - Operational data store: Ocupa tiempo real, almacena datos de forma rapida
  - Data Marts:
- Arquitectura:
  - Sencillos: Almacenan sin procesar
  - Sencilla con zona de preparación: Procesamiento (se prepara) antes de enviarse la respuesta

    - Radial: Se personaliza para cada área de una empresa
    - Sandbox: Para explorar los datos
- Arquitectura:
    - Sencillos: Almacenan sin procesar
    - Sencilla con zona de preparación: Procesamiento (se prepara) antes de enviarse la respuesta
    - Radial: Se personaliza para cada área de una empresa
    - Sandbox: Para explorar los datos, experimentar con ellos

SSS (SQL Server Integration Services): Por medio de SQL

Modelo estrella

Tenemos tablas de hechos y dimensiones

- Hechos: Información que relaciona las dimensiones, reciben muchos datos. Entonces la tabla de hechos es solo una tabla de llaves foráneas. Si puede tener datos pero principalmente es FK.
- Dimensiones: Los nodos en el *edge* de la estrella, tiene el contexto. Lo que surge a partir de ello, de forma que este tipo de nodos esta especializado por departamento

P. ej una tabla de hechos tiene las ventas, donde, cuando, cliente, monto y la de dimensión dice mas cosas sobre el contexto y detalles, como quien lo vendio.

Ventajas

- La información esta clasificada de forma optima para que su manejo sea más natural dependiendo del departamento
- Como tenemos multiples tablas pequeñas el rendimiento aumenta
    - Reduce redundancia
- Flexibilidad: En los sandbox podemos experimentar con las dimensiones
- Escalabilidad:
- Integración de herramientas: Como PowerBI, visualizaciones, etc.

Desventajas

- Esta complejo implementarlo de forma inicial porque requiere planificacion cuidadosa.
- Almacenamiento: Las tablas de hechos acumulan mucha información.
- Mantenimiento: Hay cambios en la información que se maneja, se debe mantener actualizado.
- Habilidades: Puede requerir capacitaciones externas.

Si se necesita integrar todas las fuentes de información de una empresa, los usarios requieren hacer reportes, se gasta muchos recursos, se puede optar por un data warehouse.

Comentarios del profesor

El profe quiere definiciones exactas y claras con lo que podemos hacer con ello

- Es solo de lectura un data Warehouse? Normalmente se escribe una sola vez y se lee muchas, no se tiene la intención de que se borre nada, no es una base de datos operativa. Los datos de producción están en otro lado, no es para que nuestro negocio funcione.
- Impulsa la inteligencia empresarial, entendemos en negocio por medio de los datos y vemos cómo mejorarlo, no es una cuestión de operar.
- No es operativa, si un dato llega, no se borra. Se inserta y lee, y ya.
- No se trata de tener los exceles o el data warehouse, sino que es simplemente un agregador de información. Responde preguntas diferentes a las producciones.
- Es contextual:

    - Si hacemos un data warehouse de una empresa tenemos un diseño para esa empresa en especifico, pero no podemos usar el mismo modelo.
    - Data Mart: Es un data warehouse más pequeño, especializado a solo un área de la empresa (p. ej. ventas). Contextual cada uno, a una parte pequeña de la empresa.
- Una de las preguntas es cómo planearlo, si iniciar con Marts y después diseñar el Warehouse gigante, o qué hacer.
- Una dimensión: Aspectos del contexto que nos interesa analizar (p. ej. clientes, tiempo), solo usamos la información relevante para poder analizarlo.
    - Normalmente, si se hace un data warehouse de estrella se tiene una sola tabla de hechos.
    - En la presentacion vemos dos data marts funsionados porque comparten dimensiones.
    - Tenemos atomicidad de tiempo, puede que solo nos importe el año, o mes, no la fecha y segundo.
    - Si en un datamart tenemos resoluciones de tiempos medidos en distinto tipos no podemos fusionar data marts.
- Si en nuestro data warehouse queremos tener una ubicacion y otra de clientes, vamos a tener dos dimensiones distintas, estamos hablando sobre dimensiones, no entidades completas. Entonces no pensamos en objetos, sino en datos que nos interesan para entender nuestro negocio (p. ej. quien compra, desde donde).
- Dimension: Aspectos con los cuales quiero analizar los datos. Lo ponemos de forma que sea facil analizar los datos.
- ETL: Extraer, Transformar y luego subir:
    - Extracción: Siempre se tiene este paso, el tiempo de desarollo de los ETLs es lo más tardado. **Ubicar las funetes y obtener los datos, los que nos interesan**, tenemos que determinar que datos necesitamos.
        - Son solo los datos que tenemos, podemos contestar con lo que tenemos
        - Si no tenemos datos podemos ver cómo mejorar nuestro sistema para poderlo conocer.
        - Tenemos que entender qué es lo que quieren los jefes para entender los campos que necesitamos, porque nos ayudan a responder las preguntas.
    - Transform: Limpiar la información (p. ej. 'CDMX', 'Ciudad de Mexico'), o homologar los datos (p. ej. 'Enero', 'EN', 1 -> 1)
        - Para poder jugar con los datos, tienen que estar totalmente organizados y homologados
    - Carga: **Tener una carga inicial y después (mantenimiento)**:
        1. En tiempo real: El ETL es complejo y demanda recursos
        2. Cada cierto tiempo: Dependiendo de cada cuánto tiene sentido ver nuestro negocio
- ELT: Se usa por ejemplo en big data, cargamos todo y a ver qué nos sirve
- Es un analisis, nos interesa saber la edad exacta, no tener que calularla.
- Como analisis toca ver la definicion de analisis, supongo que implica tiempo y circunstancia.
- Se *rompe* con la normalización, y los esquemas de relación, en este tipo solo importan las dimensiones y cómo nos funciona el modelo para responder las preguntas.
- Si necesitamos conectar dimensiones, ya no estamos en estrella, es un copo de nieve.
- Si la estructura es muy compleja la estructura estrella no sirve.
- No es barato, se necesita muchos recursos para un proyecto asi. Hay productos que lo hacen por fuerza bruta, por eso no todas las empresas lo tienen.
- Online Analitic Apps, como PowerBI, nos permiten modelar los datos. Nos permite formar *cubos* de información, usando las dimensiones que definimos.

Copo de nieve

Esquema dimensional: La dimensiòn es cualquier dato que se requiere analizar para el contexto que tenemos. Tabla de hehcos: Almacena informacion con muchas llaves foraneas, relaciona las tablas de

dimension. es, en el copo de nieve va normalizando las tablas de relacion para que se faciliten los queries. rtimos del esquema estrella por medio de normalizaciones, para obtener tablas de dimension no directamente relacionadas a la de hechos, pero si de acorde a la dimensión.

Esto lo hacen para facilitar los queries y su performance

- Se puede ahorrar en almacenamiento
- Es mas facil mantener las tablas normalizadas
- Se hace menos intuitivo para los usuarios
- Con mas joins se reduce el performance
- No se tiene que normalizar todo por que si, se toma en cuenta lo que se quiere lograr, lo que se tiene para lograrlo (p. ej. personal), y demás del contexto.
- Para quitar redundancia y ganar performance.
- La normalizacion no es natural para quien usa excel, por lo que el esquema de copo de nieve tampoco es natural para ellos. Aunque para eso esta PowerBI y demás.
- Hasta donde normalizar? La clave es, recordando que un datawarehouse esta motivado por quienes toman deciciones, debe ser capaz de responder sus preguntas.

  La guia para normalizar depende de las sub categorias que queremos obtener, normalizamos con la intencion de obtener las sub categorias que permiten tener consultas claras sobre lo que debemos responder.
- Tatuarse:
  - El data warehouse siempre tiene contexto y pregunta asociada, si no hay pregunta nos mandaron a hacer algo por diversion
  - Si no tenemos preguntas claras no podemos armar el data warehouse.
  - Tenemos que saber exactamente qué tenemos que hacer, de forma que podamos construir.
  - O se involucra de forma correcta la directiva o no va a salir.

Object Database

Es producto de la inteligencia artificial, habia lenguajes como SmallTalk que buscaban emular el comportamiento humano. Abstraer la realidad y darle las caracteristicas que tiene un objeto completo, como lo que es y para qué sirve.

Lo que se busca es generar un modelo de datos orientado a objectos qque tenga persistencia. SGBDOO (Sistema Gestor de Base de Datos Orientado a Objetos). En lugar de esquemas tenemos mapas de objetos, con herencia, por ejemplo.

Se necesita:
- Base de datos
- Modelo de datos:
- Gestor de base de datos: Se encarga de la persistencia: La fabrica de los objetos que se van a tener en el modelo, relacionado con la persistencia

Tenemos *fábricas* de objetos y las instancias del mismo. La fabrica se encarga de dar instancias de objetos.

Surgen en 1989 con el Manifiesto de A:::
- Objetos complejos: Componer objetos por agregación (uno se conforma de varios).

- Mecanismos de identidad de los objetos: Como identificar entre ambos objetos, aun cuando son identicos (p. ej. viven en distintos lugares en memoria)
- Soporte a encapsulación
- Tipos o clases
- Soportarse el enlace dinámico: Para ejecutar codigo externo, resolviendo lso simbolos de forma dinámica
- DML: Data Manipulation Language (SELECT, DELETE, UPDATE).
- El conjunto de todos los datos debe ser ampiables
- Debe ser capaz de servir para bases de datos grandes
- Concurrente
- Proporcionar una forma simple de consultar los datos (p. ej XPath de XML?)

Herencia, Polimorfismo, Interfaces/Protocolos, Categorias (herencia donde solo se pueden agregar metodos)

Caracteristicas

- Identidad de los objetos:
  - Puede ser con un identificador interno y con sus referencias (p. ej. con un garbage collector)
- Encapsulamiento: Public, Private, Protected
- Manejo de objetos complejos
- Polimorfismo
- Creación de versiones:
  - Por cada clase podemos tener versiones (p. ej agregamos un campo nuevo). La base de datos debe ser capaz de manejar distintas versiones.
  - Implica el almacenamiento y carga de objetos de versiones pasadas sin problema.

Ventajas

- Podemos diseñar objetos complejos
- Manipularlos de forma rapida y ágil (aunque es más del lado del código)
- Ampliable: Se construyen nuevos tipos por medio de los que tenemos
- Modelar el mundo real
- Mejor control de concurrencia
- Menos codigo debido a la herencia
- Menos memoria por que podemos tener distintos hilos de forma cocurrente sobre la misma clase

Desventajas

- No hay estandares en la industria. Hacemos las clases de acuerdo a los requerimientos que tenemos.
- Para bases de datos pequeñas no es tan bueno, es mejor para problemas complejos

Se puede tomar una base de datos relacional como engine de persistencia para una base de datos orientada a objetos. No quiere decir que sea eficiente pero si se puede.

Modelo Inmon (o Top down)

- Intenta hacer el data warehouse de toda la empresa de una.
- Se dan resultados en mucho tiempo desde que inicia. Cuando eventualmente de los resultados sera de toda la empresa.
- Hasta el final podemos generar data marts, su fuente es el data warehouse completo.
- Busca responder que estrategia general usar para toda la empresa, busca el optimo global.
- Es muy grande, complejo,

- Modelo base: Copo de nieve. Por la complejidad no se puede solo tener estrella

Modelo Kimball (o bottom up)

Hay dos formas de ver la empresa:

- Kimball ve de arriba hacia abajo. Inicia por los procesos, se va a lo concreto. Las respuestas que busca responder es como hacer más eficaces los procesos. Empieza por el data mart.

  Inicia con secciones, haciendo data marts de secciones hasta que eventualmente los une.

  Se pueden unir los distintos data marts si comparten dimensiones. Eventualmente podemos llegar a tener data warehouse federados, que no se relacionan en nada, y eso no es un data warehouse.

  Se plantea con el hacer una estrella, tiene la ventaja de que es mas facil de contstruir, mas rapido porque no hay normalización
- Modelo base: estrella
- Top down.

Modular, escalable.

- ETL: Extraer, transformar, cargar

Se propone que se usa como una cocina, los que no son chefs no pueden entrar a la misma.

Diseño:

- Seleccionar un proceso en la organización: Las areas que se van a cubrir.
- Declarar el grano: Una fila de la tabla de hechos, cuales seran las especificaciones de las tablas. Ah ya, la granularidad de los datos, asi todos estan medidos con la misma regla.
- Identificar las dimensiones:
- Identificar los hechos:

Kimball:

- Centrarse en el negocio
- Construir infraestructura adecuada

Ventajas

- Implementación veloz
- No normalizar

Desventajas

- Pueden existir datos redundantes

Es una metdologia eficaz centrado a areas. Es rapido de construir, no tiene normalizacion, tenemos tablas de hechos y de dimensiones. Tendremos exito si analizamos exitosamente ventajas/desventajas. Lo que busca es ser lo mas preciso posible.

> Los data warehouse no se venden *per se*, sino que se vende como Inteligencia de Negocios. Para poder hacer el tipo de analisis se debe incluir un data warehouse

> Kimball: Tenemos un negocio familar, es una funeraria, tenemos un problema con la producción de los feretros, lo que queremos es mejorar un **proceso**. Inmon: Empresa, como tener una ofensiva adecuada ante la nueva ruta de la seda de china, es un problema que abarca toda la empresa.

Se pueden usar ambas estrategias: Queremos tener un proyecto de 60 meses que deberia terminar en 15 meses, en ese caso, por lo que no podemos solo usar Kimball.

Estamos en un barco de vapor, para hervir el agua necesitamos combustible. Nuestro barco es de madera, estamos en medio de la nada y nos quedamos sin combustible. Nuestra mision es que el barco que llegue. Vamos destruyendo el barco quitando lo menos importante y dandole como combustible eso al motor.

Esto se conoce como una ruta critica, para el problema de los 18 meses nos quedamos con solo lo critico. Todo el rato se piensa en inmon pero formulando las cosas para responder ya.

Dependiendo de lo que se necesita resolver hay distintas herramientas. Los ETL no cambian. Lo mas importante es responder a las preguntas del cliente, lo que quiere decir que él debe tener preguntas, saber lo que quiere. Si hoy no sabe lo que quiere te pedirá cualquier cosa.

Si no tiene preguntas podemos:
- Si tuvieramos que quitar todo menos 1 cosa, que seria? Si responde que la gente no nos resuelve nada.
- No podemos cambiar nada más que una cosa.

Con eso podemos medio saber qué es lo que quiere. Este tipo de procesos no dependen del numero de gente, no hara las cosas mas rapido necesariamente. Si metes mas gente incluso pueden chocar entre si.

Nos preguntara mas o menos que necesitamos en el examen.

CRIP-DM

Enfocado a personas que no son data scientists, que no son expertos en el tema.

- Comprender el negocio: Comienza con objetivos del negocio, como optimizar procesos, encontrar embotellamientos
- Comprender los datos: Explorar y entender los datos. Que tenemos y que necesitamos. Que calidad, disponibilidad, granularidad y frecuencia.
- Preparar los datos: Selecciona, limpiar
- Modelado: Integrar disitntas tecnicas para ver si podemos lograr el objetivo. Para simular lo que pasa en la vida real.
- Evaluacion: Los resultados obtenidos del modelo. Rendimiento, sensibilidad, precision y eficacia. Esta listo para desplegarse?
- Despliegue: Se aplica el modelo en el negocio completo.

Esta metodologia no sigue los pasos de forma lineal, sino que es un proceso de prueba y error. Entre modelado y preparacion de datos puede ser un va y viene hasta que se logre.

Video:

Ventajas:

- Ayuda a mejorar la empresa
- Plazos definidos
-

Desventajas

- Inconvenientes
- Requiere conocimientos en el tema

- La implementación como el limpiado y modelado requiere conocimiento.

Ejemplo de uso

- Tenemos una cadena de restaurantes, queremos aumentar nuestras ventas.

1. Comprender el negocio: Especificamos, queremos aumentar las ventas 10% en tanto tiempo.
2. Comprender los datos: Recopilamos los datos de los puntos de venta y de redes sociales o demás lugares.
3. Preparar los datos: Eliminar datos duplicados, var categoricas
4. Modelado: Se identifican patrones y se emplean para entender como esta funcionando el negocio y por lo tanto como mejorarlo
5. Evaluacion: De todo lo que ya hicimos, que es lo que mejorres resultados dio. Eficiencia y rendimiento, por medio de proyecciones y estadisticas que nos pueden dar un poco de conocimiento sobre su eficacia. Solo se despliega si estamos algo segurs de que puede funcionar.
6. Despliegue: Evaluamos y obtuvimos resultados, ahora se pone en platica sobre el negocio de verdad. T

Le esta preguntando sobre las regreciones lineales, habla de los minimos cuadrados. Poniendo el ejemplo de que tengamos un modelo que predice y. Para ver el error obtenemos la diferencia entre lo que tenemos y lo que debe ser. Puede que no se despliegue el modelo porque la evaluacion no dio para sacarlo, lo que hace que tengamos que volver a empezar En excel podemos graficar fast regresiones lineales Rellenar datos sus

- KDD y CRISP no compiten como pasa con Inmon y Kimball.
- CRISP: La metodologia es esta
- Si es en particular sobre bases de datos entonces usamos KDD
- KDD es parte de CRISP

CA

Lo que CA hacia es distribuir mensajes de acuerdo a la persona, haciendo que cada quien escuchara lo que queria oir. Logrando que Trump ganara.

- Obtenian datos de aplicaciones de Facebook, porque dabas permiso para ver tus datos y de tus contactos.
- Michal desarrolla el methodo OCEAN de acuerdo a lo que opinas en la red, miden tu personalidad
- Wylie: Denuncio
- FB: Pago
- OCEAN: O big 5, mide 5 caracteristicas de una persona que la permite. Con 70 likes te puede conocer como amigo, 150 como padres, 300 likes como pareja, +300 lo que piensas de ti mismo
- Tambien puede saber que orientacion sexual, tendencia politica, etc.

Giovanni Santo Omni Videns