**Password: 6REvWsN};8\_YaYT`**

**Meet: owk-iuos-mfz**

**Invitación de Telegram:** <https://t.me/joinchat/lDmv_f3ckltiOWMx>

**Prof. Gabriel Arenas.**

**gabriel.arenas@um.edu.ar**

**Materia no promocional, se practica el examen final durante la materia haciendo los trabajos prácticos, para que cuando rindamos el final es lo mismo, agregando un poco de teoría, pero la practica lo vemos en todo el curso.**

**1ra Clase (19/03/2021):**

Antes había cintas magnéticas por lo cual mover archivos, era una tarea compleja por medio de programación, ya que todos los registros había que hacerlo todo manual. Todo esto se remplazó con Sistema de Gestión de Base de datos, es todo un sistema operativo orientado a administrar esos datos. Actualmente, Base de Datos no es solamente un archivo, sino que es un sistema operativo que administra los datos.

La computadora al inicio surgió como una máquina de cálculo rápido, al poder archivar lo que hacían y generar archivos se convirtieron en herramientas de múltiples funciones. En toda la historia ha existido la base de datos, pero eran conocidos como bibliotecas, ya que antes se guardaban información y registros sobre los ciudadanos. La búsqueda de datos era muy lenta, gracias a las máquinas se pudo remplazar esto y aumentó considerablemente la eficacia.

Herman Hollerith fue el que invento las tarjetas perforadas para almacenar datos, y fue la primera máquina IBM. Hollerith funda luego IBM, siendo los primeros fabricantes de computadoras.

Década 50 empieza a usarse la cinta magnética para almacenar la información y mantenerla en un medio que pueda persistir en el tiempo. Generando los primeros archivos secuenciales, es decir para encontrar los datos hay que hacerla rodar y leyendo en orden de guardado de datos.

Década 60, salen las primeras y segunda generación de computadoras, que son muy grandes, y solo algunas empresas podían tener una, y a medida que se optimizaba la creación iban bajando los precios. Las primeras bases de datos fueron con esquema de red y jerárquica, pudiendo almacenar datos en forma de lista y árboles. En esta época IBM y American Airlines se unen para hacer un sistema operativo que manejaba las reservas de vuelos, transacciones e informaciones sobre los pasajeros de la compañía.

Charles Bachman (formaba parte de CODASYL) creo un nuevo tipo de sistema de base de datos gracias a la creación de un nuevo lenguaje de sistemas de información. CODASYL, era un consorcia de industria que querían generar un nuevo lenguaje de programación que pudiera usarse en todas las computadoras. Con ayuda del estado logran generar un nuevo modelo

Decáda 70, Edgar Frank Codd, logre crear un nuevo sistema de programación que sientan las bases de segunda generación de los Sistemas Gestores de Base de Datos, esta idea no es complicada, el plantea organizar la base de datos en forma de tabla con filas y columnas, donde cada columna tiene un nombre y cada fila tiene un ancho fijo, es decir una tabla con elementos denominados campos y cada fila se le denomina registro. Entonces, un registro esta compuesto por una serie de campos que hacen referencia a un elemento. Luego desarrolló Relational Software System, hoy en día como Oracle Corporation, desarrollando un sistema de gestión de base de datos relacional con el mismo nombre de dicha compañía.

Decáda 80, ser desarrolla SQL, es un lenguaje de acceso a base de datos relacionales, con el fin de hacer consultas para obtener datos, buscar información, administración. Este lenguaje vamos a ver en esta carrera.

**Oracle** es la base de dato mas completa del mundo, dominando el mercado hasta hace poquito, como Sql de Microsoft, MySql, entre otras. Todas basados en el lenguaje principal SQL.

Decáda 80, a principios de este año se hizo muy popular SQL, y comenzó a ser estándar de la industria, por SQL se van dejando de lado la base de datos gerarquicas y en red, ya que este tipo de base de dato tiene mayor complejidad a la hora de programar. Por eso se hizo mas popular SQL, por ser mas sencillo y usar otro tipo de modelo por medio de lista y árbol.

Decáda 90, se empezó a desarrollar el base de datos orientadas a objetos, SQL por el tema del uso de listas no es muy bueno, por eso se empezó a desarrollar base de datos orientadas a objetos.

**Definición.**

Base de Datos es un sistema estructurado de forma lógica para la administración electrónica de datos que con ayuda de un sistema de gestión de base de datos (DBMS), regulan las pertenencias y los derechos de acceso y guardan la información, añadiéndola al repositorio que contienen. La mayoría de base de datos solo pueden abrirse, editarse y consultarse con aplicaciones específicas.

**¿Por qué son necesarias las bases de datos?**

Porque antes el almacenar datos era muy lento, Oracle y SQL revoluciono el mercado aumentando la eficacia en el manejo de datos, disminuyendo el tiempo de acceso a ellos.

**Sistema de Gestión de Base de Datos.**

Sistema ERP: Trabajar de modo automatizado, para que la aplicación avise cuando falte material, pedidos automáticos, etc.

Sistema EMRP: Organizar producción al trabajar con “Just In Time” y no tener almacenado insumo ocupando espacio y teniendo capital inmovilizado.

Sistema ACID (Atomicidad, consistencia, aislamiento, durabilidad):

Atomicidad: Leer **Apuntes01**.

Consistencia: Leer **Apuntes01**.

Aislamiento: Leer **Apuntes01**.

Durabilidad: Leer **Apuntes01**.

Requisitos:

* Almacenar datos: almacenar datos en forma electrónico, que se puede acceder mediante consultas.
* Editar datos: La mayoría de bases de datos permiten editar los datos guardados, según los permisos que tengamos.
* Borrar datos: Los registros de las bases de datos pueden borrarse. En algunos casos pueden salvarse datos o eliminar definitivamente
* Gestionar los metadatos: La información se guarda con metadatos o meta etiquetas (son información sobre esos datos), permiten ordenar la base de datos y facilitar la función de búsqueda.
* Seguridad de los datos: tiene cuatro operaciones fundamentales, también conocido como CRUD.
  + Crear (create)
  + Leer/recuperar (read/retrieve)
  + Actualizar (update)
  + Borrar (borrar)

La base de datos tiene que ser segura para evitar que sujetos no autorizados puedan acceder a la información que guardan. Además de un buen método de cifrado, para evitar la manipulación y pérdida de datos.

* Integridad de los datos: Leer archivo (**Apuntes01**), lo veremos más adelante.
* Función multiusuario: El sistema tiene que poder funcionar correctamente cuando varios usuarios escriben a la vez, además de tener permisos y seguridad de los datos, para que los datos se puedan acceder desde diferentes dispositivos
* Optimizar las consultas: Tiene que procesar consultas de la mejor manera posible, tiene que tener la ruta más corta, así se aumenta el rendimiento global de la aplicación.
* Triggers y stored procedures: Son dos miniaplicaciones guardades en el sistema de gestión de base de datos que se activan con eventos. Con estas aplicaciones mejora la integridad de los datos. La mayoría permiten tener programas almacenados internamente que admite a los usuarios almacenar programas dentro de la base de datos. Los Triggers están relacionados con las tablas (agregar, borrar y editar datos dentro de una lista) verifica si se implento un dato en una tabla y genera datos que sirven para hacer un seguimiento y los Stored Procedures pueden ser funciones de diferentes tipos, como una función que formatee la fecha.
* Transparencia del sistema: Importante en los sistemas distribuidos (base de datos en diferentes lugares), son procesos que corren en segundo plano, y el usuario final no vea todo el proceso que se realiza para mantener la base de datos al día.

**Modelos de base de datos en la historia:**

Todo depende de las evoluciones técnicas de la transmisión electrónica de datos, donde en la historia puede ganar el fabricante o la tecnología, tecnología como la eficacia y la manejabilidad, y por parte del fabricante más renombrados con alianzas o desarrollos.

Modelo jerárquico de base de datos:

Modelo más antiguo de base de datos, por ahora lo supera el modelo relacional en cuanto a implantación. Es un modelo arbolado y XML utiliza este sistema para guardar datos, entonces algunas compañías de seguro y banco han utilizado a pesar que el relacional se utiliza mucho más, el más conocido es el IMS/DB de IBM. Fácil de implementar cuando se trabaja con XML.

Relación padre e hijos son inequívocas. Cada nodo hijo tiene un padre, y un padre puede tener varios hijos, conectar dos árboles diferentes no es fácil además de que los hijos están separados por sus padres, por lo cual es clara la relación entre sus datos, los registros tienen relación de padre e hijo, y es un modelo

Modelo de Base de datos en red:

Los registros no tienen relación padre e hijo, por lo que un registro puede tener múltiples procedentes. Tiene muchas formas de llegar de un punto a otro, ya que están conectados entre sí, pueden definirse dependencias, por lo que para pasar de un dato a otro debe pasar entre datos, por lo que puede ese dato bloquear el paso. Los registros pueden añadirse o eliminarse sin que la estructura global se vea afectada.

Este modelo se utiliza en grandes ordenadores como UDS de Siemens y DMS Sperry Univac. Con el tiempo, se desarrolló una mescla entre red y relacional, aunque no funciona del bien del todo.

Modelo de base de datos relacional:

El modelo más utilizado hoy en día. Se utiliza en el lenguaje SQL. Utiliza un atributo (Columnas) que identifica un elemento o registro (Fila o Tupla) dentro de la tabla y tienen una relación entre ellos.

Este modelo trabaja en tablas independientes, que utiliza la localización de los datos y sus conexiones. Los atributos no se pueden cambiar, son consideradas tabla primaria, ya que es único para cada elemento o registro.

Modelo de base de datos orientado a objetos:

Nacen en 1980 y hasta hoy, no han tenido un amplio desarrollo, hay desarrollos Open Source utilizan muy poco la memoria y su administración es bastante sencilla, trabajan con lenguaje OQL, muy similar SQL, pero orientado a objetos.

Los datos se guardan en un objeto junto con sus funciones(métodos) y atributos. Tienen un identificador que es único, el objeto se agrupa en clases dando como resultado a una jerarquía de clases, las ventajas son con problemas de tipo de datos complejos, trabajando de forma autónoma, almacenando objetos nuevos de forma sencilla. Lo negativo es que no es muy compatible con nuevos sistemas como lo es con los datos relacional.

Oracle tiene un modo de datos orientado a objetos relacional, que intenta mezclar lo mejor de estas dos.

Modelo de datos orientados a documentos:

Leer **Apuntes01**

Todas estas bases de datos se encargan de solucionar diferentes problemas, mejor orientadas para el trabajo de documentos.

Base de datos: Modelos y características:

**Cuadro que hace un resumen de cada uno.**

**Instalar la versión actual de MySQL y Python**

**2da Clase (26/03/2021):**

**Apuntes03.pdf**

**Ver videos:**

Diagrama Entidad-Relación:

<https://www.youtube.com/watch?v=TKuxYHb-Hvc>

Claves Primarias y Secundarias

<https://www.youtube.com/watch?v=jshi9VCTm7g>

**Claves y tipos de claves**

Entidad: Clase principal, que contiene atributos.

Claves: Atributo de una entidad, en una entidad puede haber varias claves y tienen características especiales según la utilidad de esta clave.

Claves Principales: Son la que evita que repitamos dos clientes usando el mismo código. Podríamos identificar clientes a través del Cuit o DNI, Identificando unívocamente al registro, siendo en teoría números únicos. Hay que seleccionar para toda la vida de nuestro sistema, utilice un campo único para el registro.

La clave principal es una clave que va a ser única para la tabla, entonces la idea es nuestra tabla debe permitir de alguna manera la selección de un campo que represente únicamente al registro.

Integridad referencial: El dato que se necesita es una sola y no puede haber más de una.

Clave Externas FORANEAS: Esto es cuando se accede a un dato de otra tabla.

Superclave: Cuando tenemos un conjunto de atributos y los podemos identificar con un atributo, siempre que podamos identificar ese conjunto de entidades con ese grupo de atributos y si le agregamos otro y sigue cumpliendo esa condición de hacer referencia al grupo de atributos entonces a eso le llamamos superclave.

Clave Candidata: Siendo esta una superclave, si le sacamos un componente deja de ser superclave, entonces pasa a ser clave candidata.

Clave Primaria: Siendo una clave Candidata, pero es elegido por el desarrollador para hacer algo en especifico dentro del registro. Normalmente en el diseño de base de datos las claves primarias para asegurarnos de que sea única, obligamos a que cumpla esa condición, generando un campo en la tabla enumerando de forma creciente, asegurándonos de que el numero no se repita y el identificador de numero va a ser único para una tabla.

**Normalización.pdf**

Cuando tenemos identificadas las identidades de nuestro sistema, es un proceso al que tenemos que someter ese modelo para obtener una serie de condiciones que hacen que nuestra base de dato trabaje mejor. Para evitar escribir datos en más de una tabla, siendo algo redundante.

Desnormalización: Se trabaja los datos para el análisis.

Redundancia: Repetición de datos en una lista. Siendo una demora, corregir la repetición, hacer cambios o pedir esos datos.

Anomalías de Actualización: Al tener datos en más de una tabla, puede que se actualice parcialmente el dato.

Anomalías de borrado: Al tener datos repetidos y borro un cliente, y me queda todos los datos de referencia a ese cliente, y si eso no es borrado, que sin saber a quién corresponde.

Anomalías de Inserción: No poder agregar datos a mi base de datos porque faltan otros datos. Ejemplo, trabajar en una tabla donde ponemos el País de un domicilio, siendo separado en una tabla, el domicilio del cliente es calle, número y país de código, si inserto datos, y no tengo dado de alta el país no voy a poder crear ese registro, si no creo primero el dato de país correspondiente para ese domicilio.

Reglas de Codd: Codd fue el que diseño una lista de reglas que tiene que tener una base de datos.

Cada campo tiene que ser del mismo tipo de dato, no puedo mezclar fechas con números.

Claves candidatas:

Clave foránea:

Clave alternativa:

Clave compuesta:

**Identificar un sistema, que tipo de tabla es:**

Tablas maestras: Guarda información única para el sistema. Teniendo una clave primaria y no tiene referencia cruzadas hacia otras tablas. Pueden haber referencia, pero solo para mejorar el modelo, ej: Si pongo en una tabla de cliente el domicilio y separo la dirección del país, provincia y localidad por separado. En la tabla de cliente voy a tener una referencia a través de código a una tabla correspondiente, teniendo claves foráneas de la tabla clientes

Tablas movimiento: Guarda información que no es única para el sistema, en donde se mezclan datos.

**Formas Normales de Reglas de Codd:**

Primera Forma Normal (1FN): Tiene que ver con los grupos repetitivos

Hay grupos de atributos que están referidos al cliente, todo lo que esta relacionado con cliente, producto, o identificar pedidos lo pongo todo separado, llamando esto grupos de referencia (No repetitivo.)

Atributos atómicos: Corresponde a un Cliente Indivisible.

Un atributo nulo significa que una columna no existe, por lo tanto, el punto 5 y 6 es lo mismo.

Por más que la tabla la ordene por diferentes columnas, los registros se deben mantener siempre unidos.

No tener grupos repetidos en varias columnas. Esto sirve para evitar tener problemas de valores nulos en las tablas.

Segunda (2FN): Tiene que ver con las relaciones.

Cada tabla se tiene que referir a una única cosa, siendo lo más atómica posible.

Todo campo que sea independiente de la clave principal, no tendría que existir en esta tabla. Evitando tener atributos que sean dependiente si o si de la clave primaria.

Dependencia funcional: se dibuja con una flecha, y significa que un atributo depende de otro atributo.

En el ejemplo se ve que el IDProducto se sabe tiene dependencia con el Nombre del producto, por lo cual esa relación, hace separar la tabla para evitar repeticiones.

Cuando trabajamos con datos que deben mostrarse de una determinada manera, y con el tiempo se tiene que modificar, ahí si se tiene que duplicar datos para no perder periodicidad de los datos. Es decir, no se puede perder los datos por normalizar una tabla.

Tercera (3FN): Se quita toda dependencia transitiva de la 2FN, ya que ningún campo que no sea parte de la clave candidata, puede depender de otro campo que no sea la Clave Candidata.

Tengo que asegurarme que no tenga ninguna dependencia funcional mi tabla.

Tener en cuenta estos tres formas normales principales para ayudar a hacer una tabla de base de datos, ya que las demás son demasiado, dando problemas a futuro por normalizar demasiado dando problemas.

**Conceptos de la Normalización:**

Dependencia Funcional: Cuando una clave depende de otra.

Claves o Llaves: atributo que diferencia cada tabla e impide que se nos repita la fila o cuplas.

Dependencia transitoria: Dependencia que nos relacionan tablas principales con tablas secundarias.

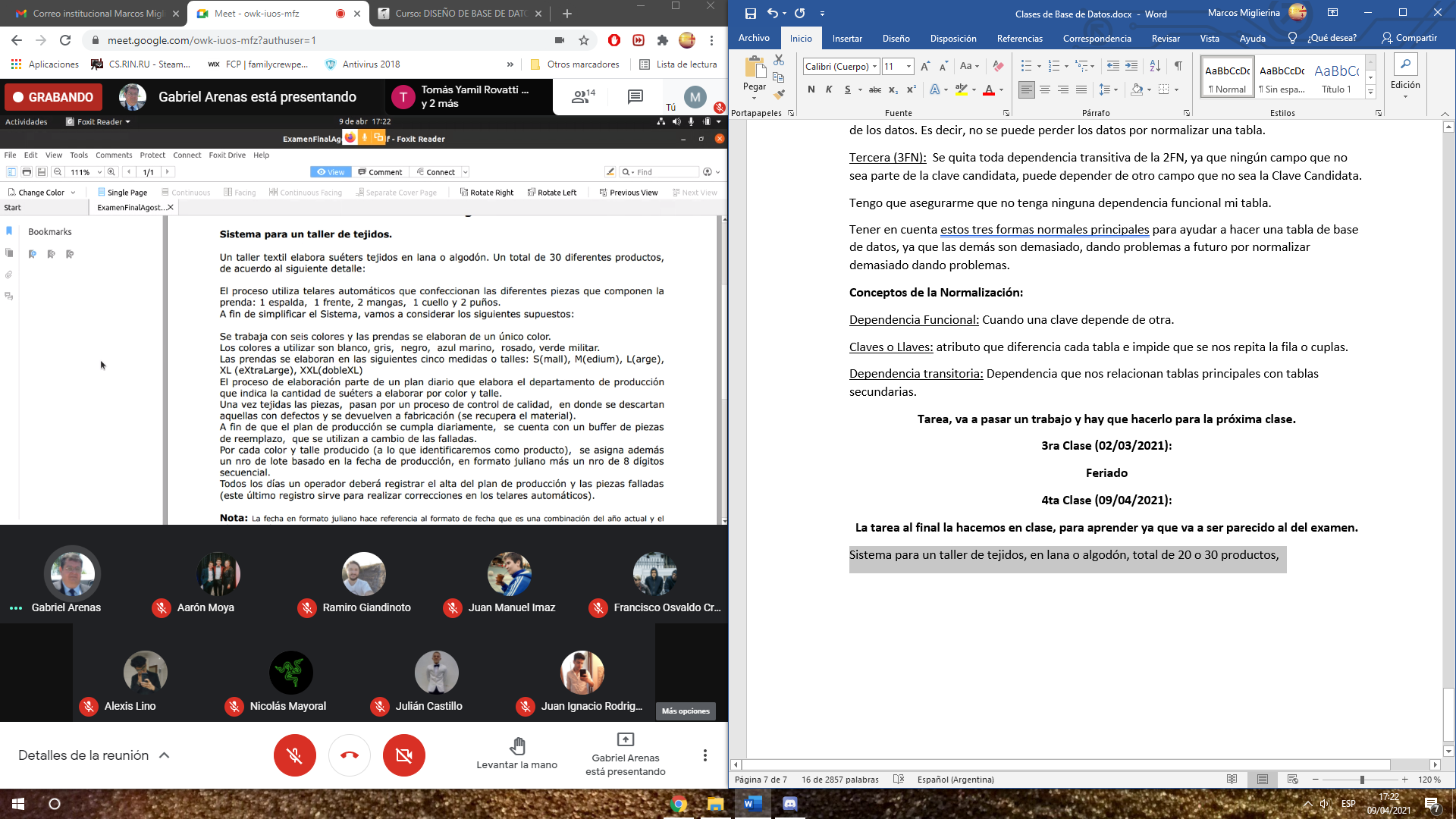
**Tarea, va a pasar un trabajo y hay que hacerlo para la próxima clase.**

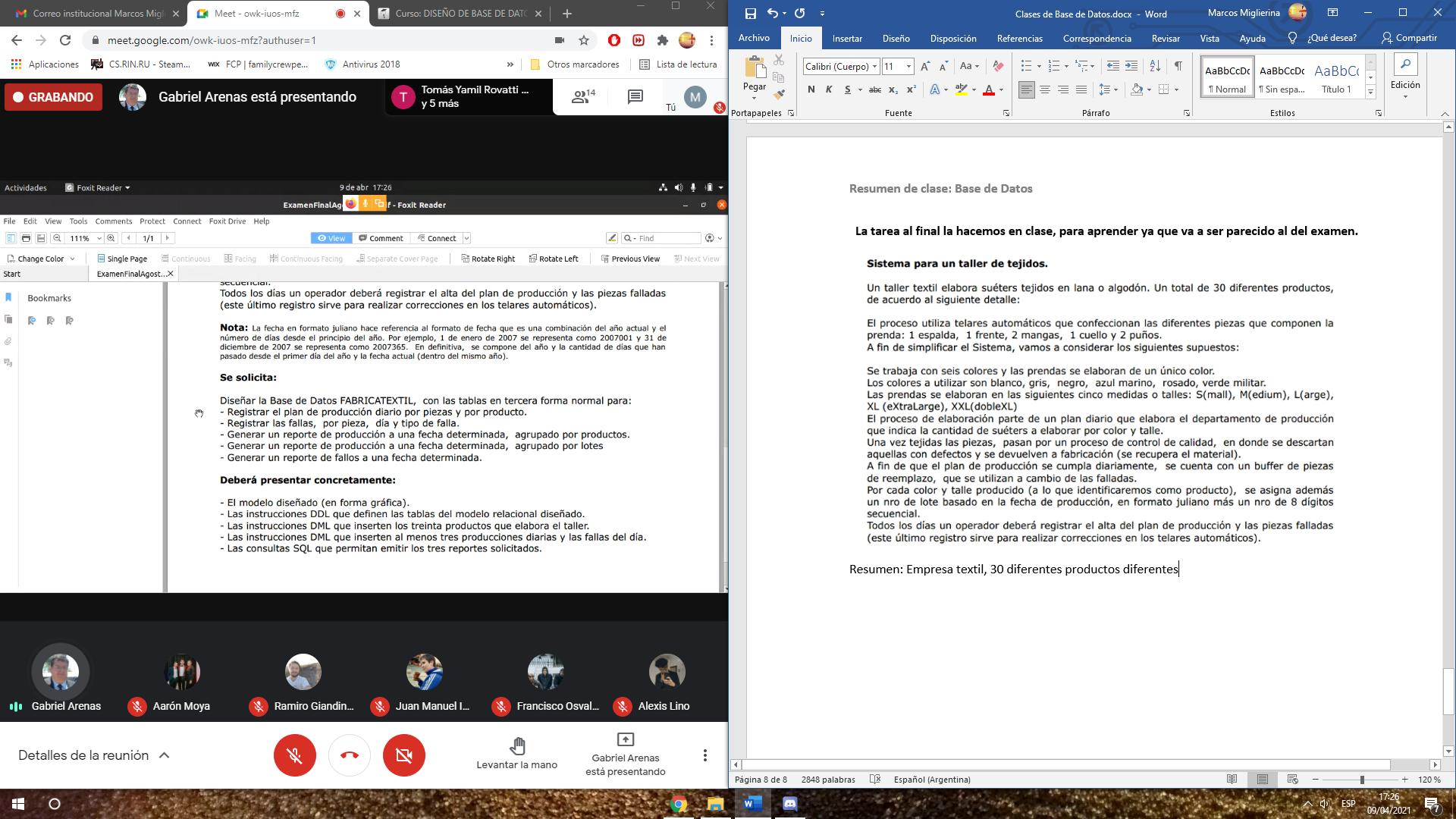
**3ra Clase (02/03/2021):**

**Feriado**

**4ta Clase (09/04/2021):**

**La tarea al final la hacemos en clase, para aprender ya que va a ser parecido al del examen.**



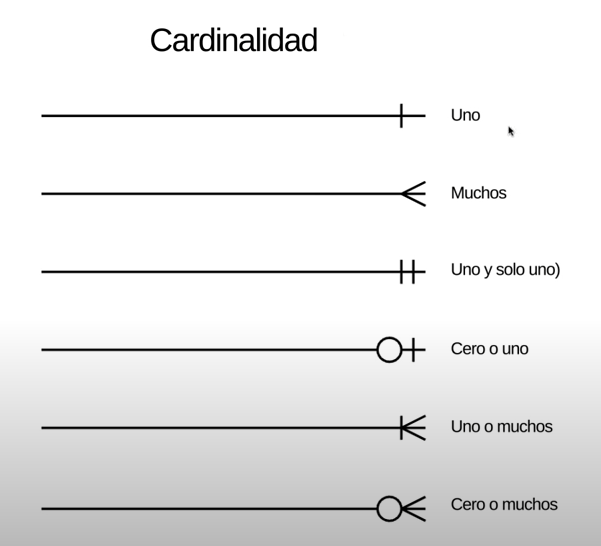
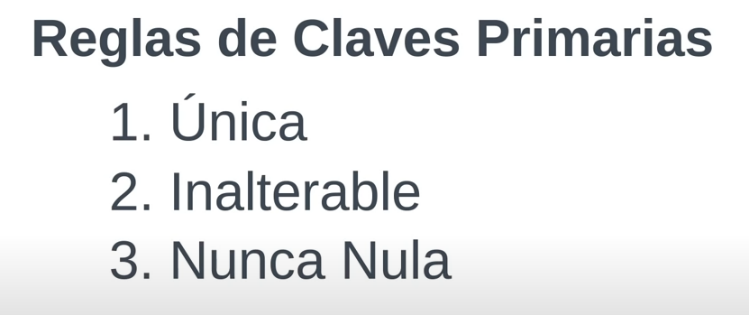
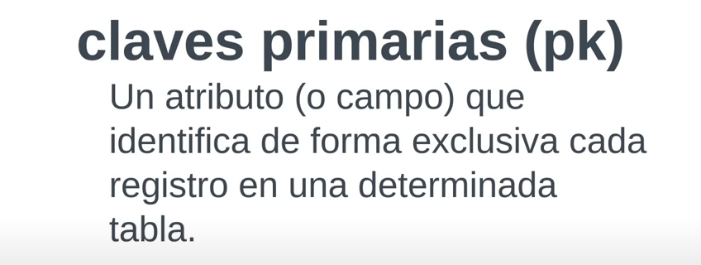


**Resumen:**

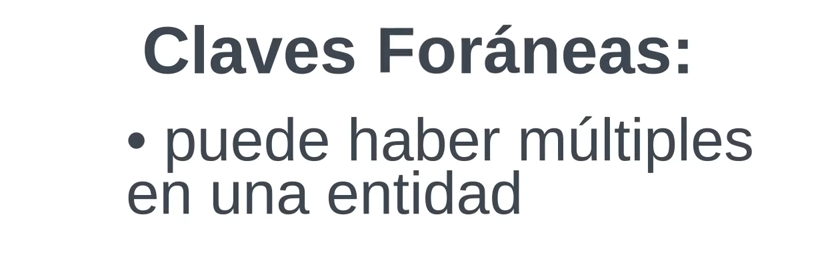
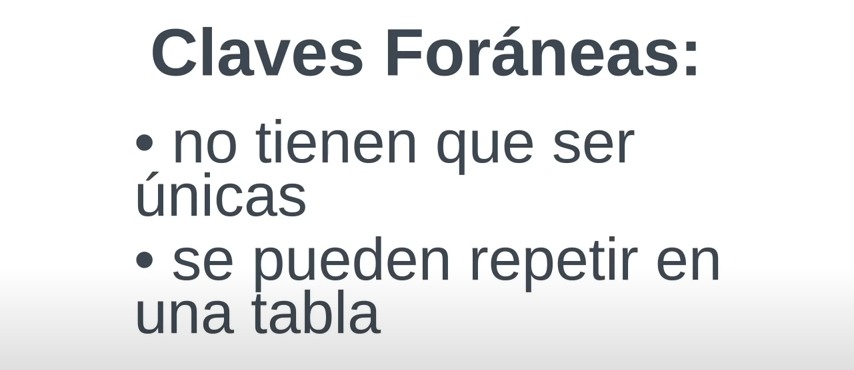
Empresa textil que desarrolla suéteres de lana y algodón (2 tipo), 30 diferentes productos, proceso para confeccionar es 1 espalda, 1 frente, 2 mangas, 1 cuello y 2 puños (Tiene 7 partes). Se trabaja con seis colores y se elaboran con un único color. Llevan 5 talles. Departamento de producción indica la cantidad de suéteres a elaborar por color y talle. Control de calidad donde se verifican piezas de recambio para suéteres fallados. A la tabla identificada como producto, donde se encuentra el color y talle, se le agrega un nro de lote, basado en la fecha de producción, en formato juliano más un Nro de ocho dígitos secuencial. Todos los días un operador debe registrar el alta del plan de producción y las piezas falladas (este último registro sirve para realizar correcciones en los telares automáticos).

Hacer trabajo en apuntes 3.

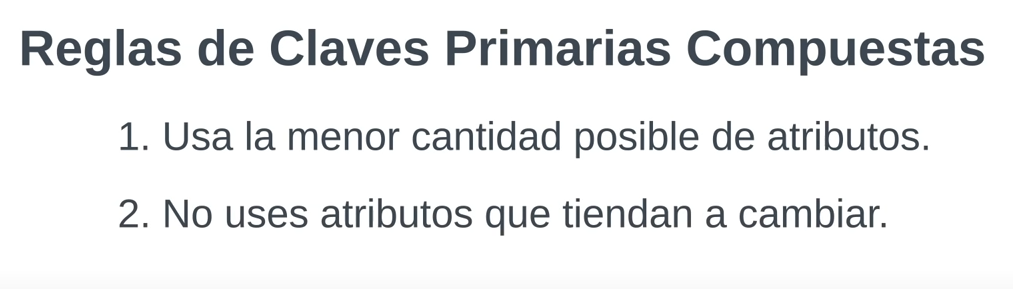
Se tiene que hacer el diagrama de identidad y atributos del problema. Lo visto en análisis de sistema.



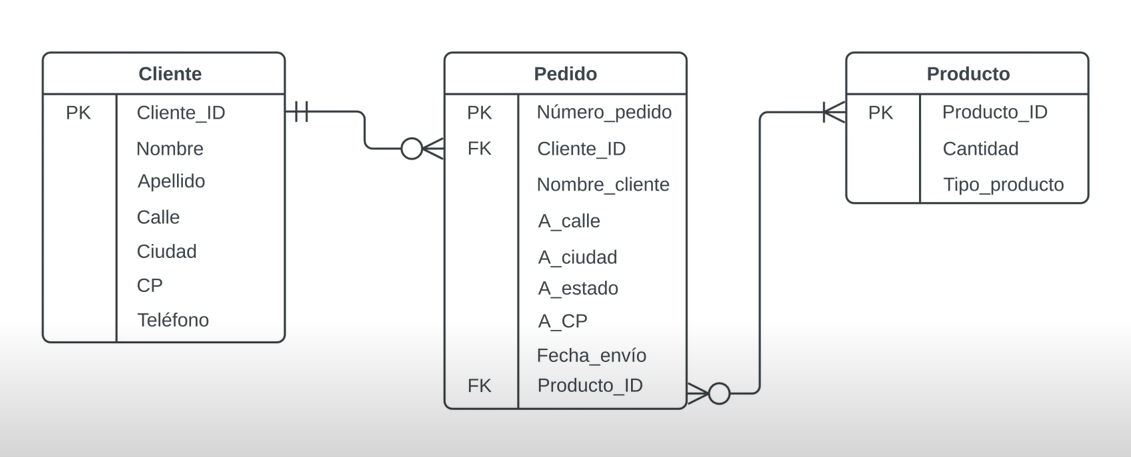
**Clave foránea:** Cuando una clave primaria en una lista, la utiliza otra lista, siendo en esta otra lista la clave foránea. Se escribe como (FK)



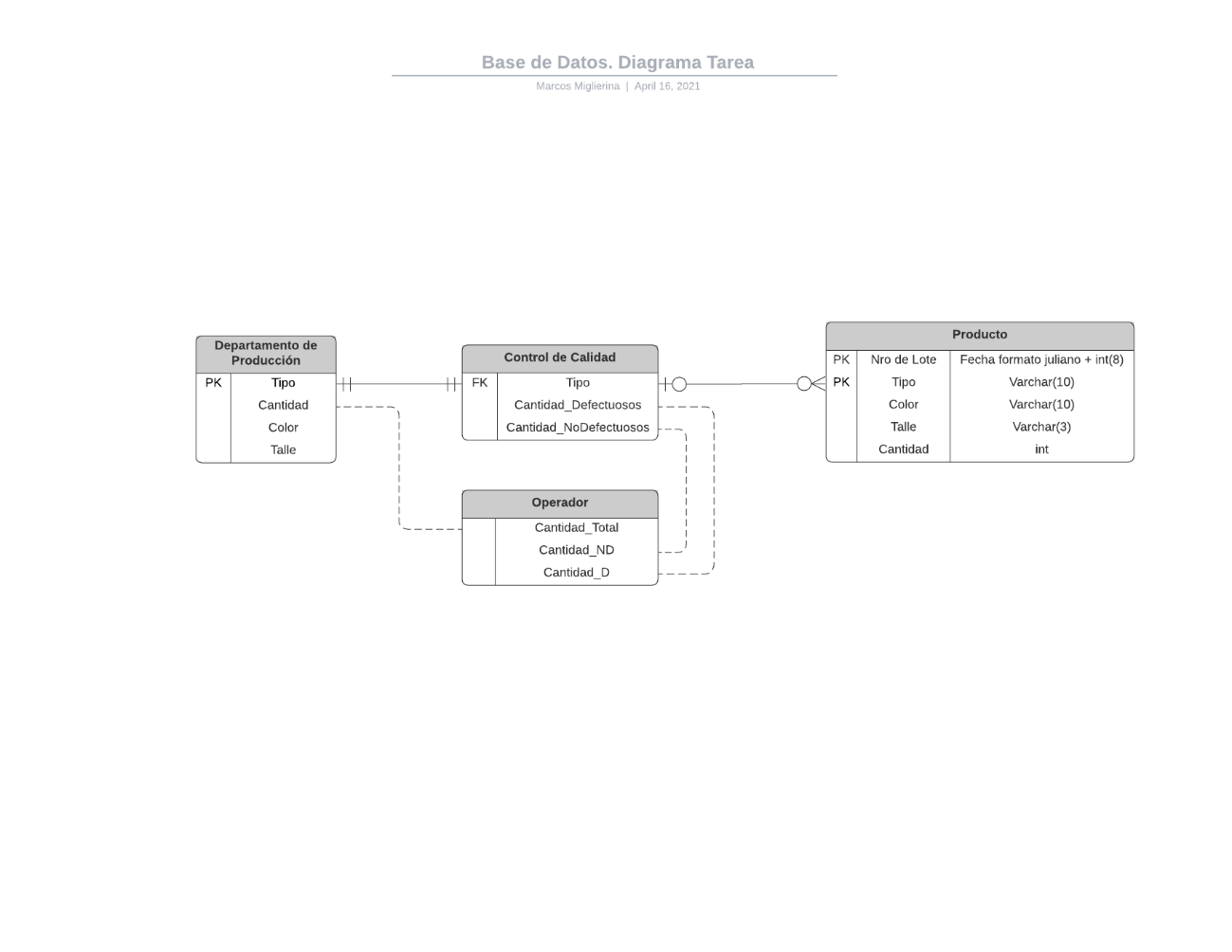
**Clave primaria compuesta:** Son cuando juntas dos claves primarias, para obtener una clave primaria única. Se escribe como (PK,FK) en los dos valores.



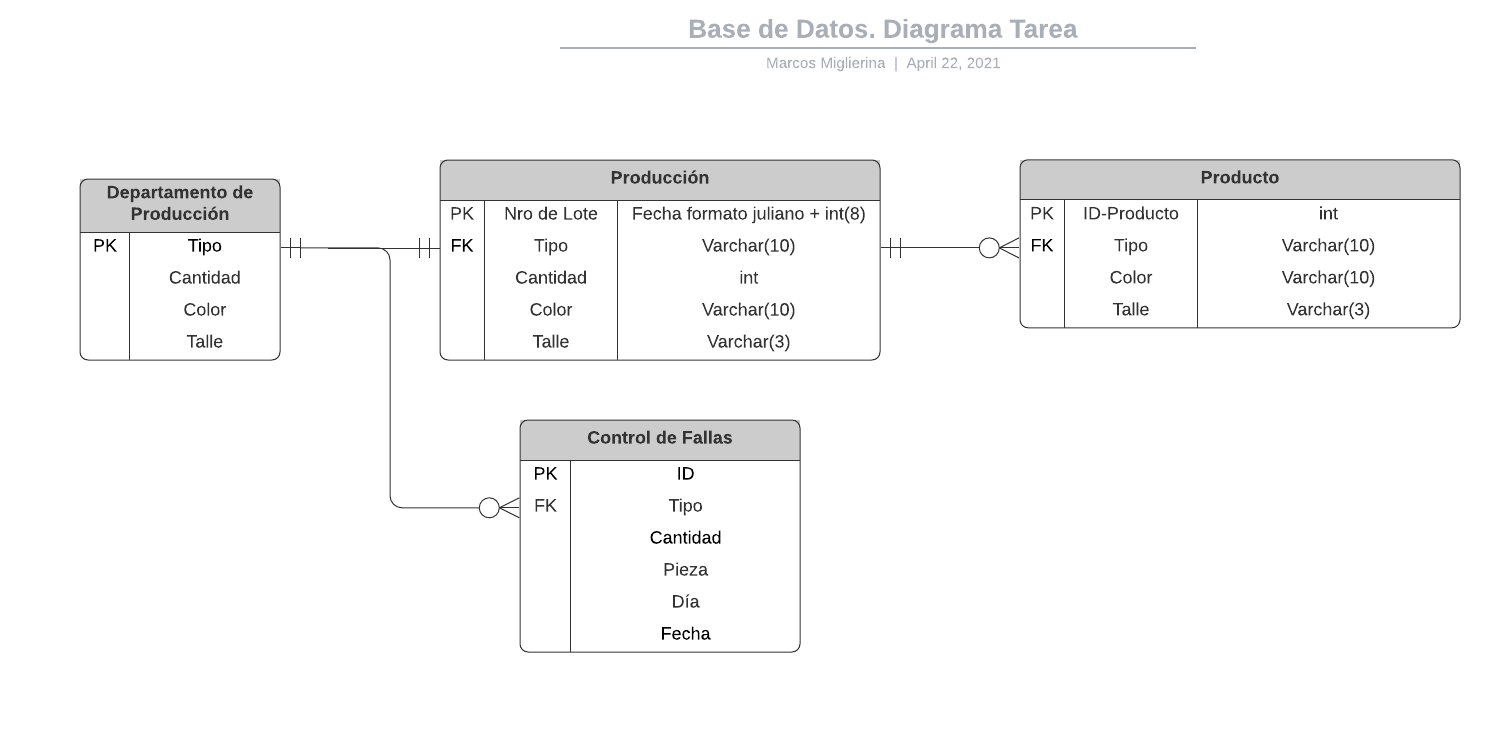
**Tabla puente:** Se usa cuando tenes una relación de muchos a muchos y da la información que hace falta entre ellos. Un ejemplo es tener una tabla de pedidos entre la interacción entre cliente y producto.



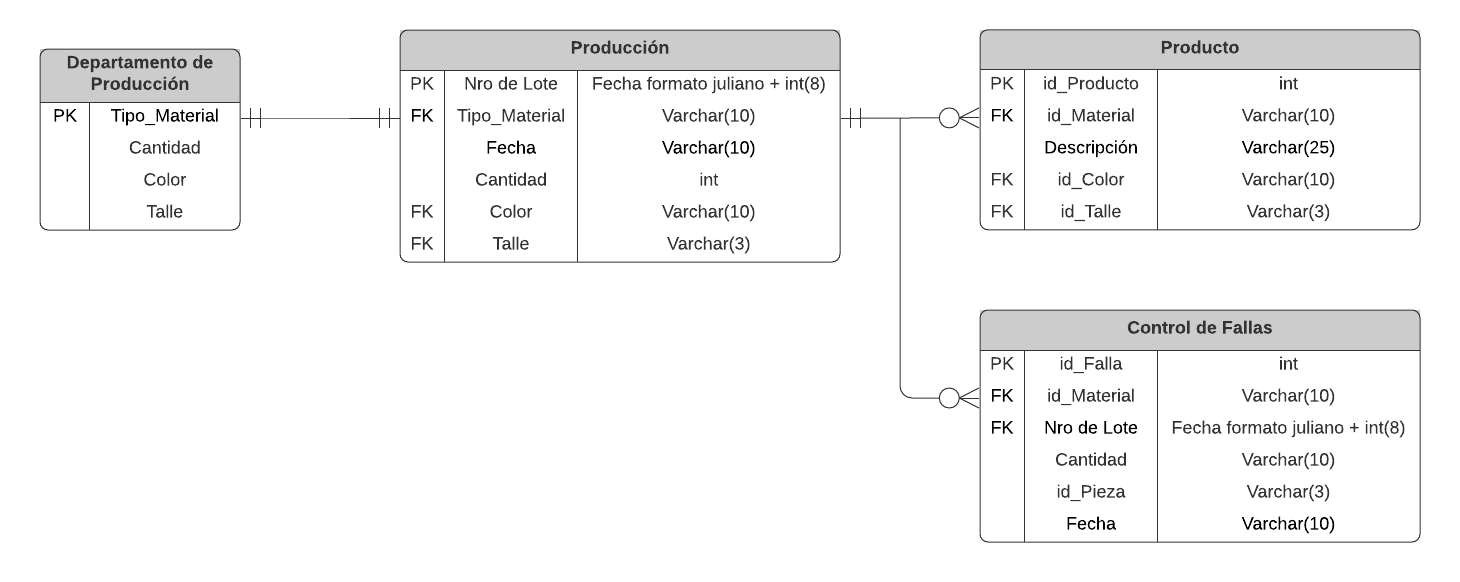
**Primera presentación**



**Arreglado un poco.**



**Arreglado con ayuda de la 6ta clase:**



Que fabrico y como lo fabrico lo hago en otra tabla. Id de producto que identifique. Nro de lote agrupa productos, por lo que hay que hacer un id para identificar unívocamente a cada uno. Agregando una tabla entre medio de producto y control de calidad. Crear tabla de lo que se genera diariamente (tabla de movimiento). No es necesario hacer tabla para operador.

**5ta Clase (09/04/2021):**

Vemos todo en el programa de MySQL Workbench.

Vemos ejemplos sobre las tablas que ya vienen en MySQL.

Con el dibujo con + crea una ventana, en la que podemos intercambiar información con la base de datos.

Primero hay que elegir que base de datos vamos a trabajar, vamos a ver comandos:

Control + enter: para cargar una línea de código.

**use sakila;** Tener en cuenta de poner **punto y coma al colocar comandos.** En este caso use, sirve para referenciar la base de datos que vamos a interactuar.

**Select “\*” from “list”:** Busca una lista, o podés usar asterisco (\*) para verificar todo y from es de donde quieres obtener los datos, en este caso colocas el nombre de lista.

**Select firstName, lastName from actor**: al colocar coma, podes traer varias columnas de la lista.

**Select \* from address where address\_id = 6:** Where lo usamos para obtener datos más precisos, para traer un dato de una sola fila donde el id es 6.

**Select address\_id,**

**address**

**from address:** Esta forma se llama adentada, porque es una forma más detallada y legible.

**Select Dom.address\_id,**

**Dom.address**

**from address Dom:** Se puede poner un alias a las tablas, entonces para llamar valores dentro de la tabla, se puede buscar directamente como **domi.address** para obtener el valor.

**Select Dom.address\_id “ID\_Domicilio”:** De esta forma se puede leer mejor con el id que le pusimos entre comilla al leer la tabla.

**Select count(\*) from address:** Esto sirve para contar cuantas filas tiene una tabla.

**Select Dom.address\_id,**

**Dom.address “Lugar”,**

**Dom.city\_id “ID Ciudad”,**

**Cit.city “Ciudad”,**

**from address Dom,**

**city Cit**

**where d.city\_id = C.city\_id;** Esto sirve para unir (Join), dos tablas diferentes y se muestre todo unido en uno mismo. Tener en cuenta que las dos tablas tienen que tener una columna en común, no tiene por qué ser las mismas, pero tiene que tener la cantidad en común.

**Arreglar el trabajo anterior y mandarlo por correo**

**6ta Clase (23/04/2021):**

**Vemos el trabajo de entidad-relación que había que hacer del sistema de taller de tejidos**

**La pregunta para resolver el problema es,** primero hay que saber qué es lo **que se produce (salida)**, **con que es lo que lo produzco (entradas)** y **como es** **el proceso**. Con eso te da la posibilidad de realizar el diagrama

**Vimos lo de la 6ta Clase:**

**Select Dom.address\_id,**

**Dom.address “Lugar”,**

**Dom.city\_id “ID Ciudad”,**

**Cit.city “Ciudad”,**

**from address Dom,**

**city Cit**

**Producto Cartesiano:** Hacemos que se unan dos tablas. Es algo no querido en una instrucción select. Ya que se tiene que agregar una instrucción que una las dos tablas, por eso es importante el Where.

**Instrucciones de agrupación**

**Agrupador Suma:**

**Select sum(amount) from payment:** Esto sirve para obtener el resultado total de la columna.

Select **P.customer\_id “Id\_Cliente”,**

**C.first\_name “Nombre”,**

**C.last\_name “Apellido”,**

sum(**P.**amount) “Consumo”

from payment P,

customer C

**Where P.customer\_id = C.customer\_id**

**Group by customer\_id, C.first\_name, C.last\_name:** Este caso te da el cliente, nombre y apelldio y cuanto consumio cada cliente por medio de la suma. Recordar que el **Group By** se coloca todo lo que no esté en la función suma, para que lo agrupe todo junto, y el where, es para hacer relación entre las dos tablas, y no te de la suma de toda la tabla, sin que una los valores que son iguales en las dos tablas.

**Group BY:** El **Group By** tiene la función de agrupador, y te da siempre el valor según el objeto que le pongamos antes, un par de ejemplos.

Select customer\_id, **avg(amount)/count(amount)/sum(amount)** from payment

Group by customer\_id;

En este caso va a dar el **promedio/contar/sumar/valor minimo** de la cantidad, según el customer id. Si no ponemos el Group By, pone el primer id del customer, y en avg(amount) coloca el total.

Select customer\_id, amount from payment where customer\_id = 1;

Acá te da la cantidad de pagos que realiza el customer\_id, en este caso son 32, después para sacar el promedio, lo que hace es tomar esos 32 y dividirlo por el total y te da el resultado.

**Funciones**

En este caso vamos a hacer la función para obtener el año, mes o día, de una fecha almacenada en la tabla

**Select customer\_id,**

**year(payment\_date),**

**month(payment\_date),**

**sum(amount)**

**from payment**

**group by customer\_id, year(payment\_date), month(payment\_date);**

Aca en esta function se puede saber que persona compro en que año y mes y cuanta fue la cantidad total, si quitamos el customer\_id, nos da el monto total en cada mes, y si ponemos un **where year(payment\_date) = 2005 (año)** nos da el total en cada mes, cortado por año.

Where year(payment\_date) **in (2005, 2006):** Sirve para obtener valores dentro de todos los posibles valores ingresados dentro.

**Mínimo(min()) y Máximo(max()):**

**Select customer\_id, year(payment\_date), max(amount), min(amount)**

**From payment**

**Where year(payment\_date) = 2006**

**Group by customer\_id, year(payment\_date);**

**Hacer todos los puntos del Trabajo Práctico 1**

**7ma Clase (30/04/2021):**

**Repasamos lo visto en la clase anterior.**

**Íbamos a ver Subquary y justo lo llamaron al teléfono al profe, entonces…**

**//////////Esto lo saque de internet, mientras el profe movía su camioneta/////////**

**Subquary:** Es una consulta dentro de otra, es posible usar subquery dentro de **Select, Insert, Update y Delete**. La subquery, es una sentencia **Select** y debe ir **encerrada entre paréntesis.**

Hay tres tipos de consulta, de una sola fila, de varias filas o correlacionadas.

Esto lo podemos utilizar para evitar colocar el group by de Nombre e Id\_Departamento, al realizar una función de mínimo, para esto se usa las subquery, un ejemplo de esto es:

**SELECT** Nombre, Id\_Departamento, Salario

**FROM** Empleado

**WHERE** salary = (**SELECT** **MIN**(salario) **FROM** Empleados);

Si queremos añadir una condición en una consulta con cláusula GRUP BY, debemos utilizar la cláusula **HAVING**. Así podemos seleccionar el salario mínimo de un departamento, siempre que dicho salario esté por debajo de la media de todos los empleados:

**SELECT** id\_dep, **MIN** (salario)

**FROM** Empleado

**GROUP** **BY** id\_dep

**HAVING** **MIN** (salario) < (**SELECT** AVG (salario)

**FROM** Empleado)

**Subqueary de multiples filas:**

El tipo subquery devuelve mas de una fila como un conjunto de resultados. Generalmente se utiliza el **Where o Having**. Dado que esto devuelve muchas filas, hay que usar algún tipo de operador de comparación de conjuntos, estos son:

**IN:** Devuelve true si el valor es igual que un elemento de la lista

**ANY:** Compara el valor con cada elemento de la lista.

**ALL:** Devuelve true si el valor es igual a todos los valores de la lista.

Los comparadores **ANY y ALL** se utilizan combinados con los operadores de comparación habituales:

**= ANY :** Igual a alguno de los valores de la lista (equivalente a IN).

**!= ANY :** Distinto de alguno de los valores de la lista.

**< ANY :** Menor que el mayor de los valores de la lista.

**> ANY :** Mayor que el menor de los valores de la lista.

**= ALL**: Igual a con todos los valores de la lista.

**!= ALL** : No igual a ningún valor de la lista (equivalente a NOT IN);

**> ALL** : Mayor que el mayor de todos los valores de la lista.

**< ALL :** Menor que el menor de los valores de la lista.

Ejemplos:

Seleccionar los empleados cuyo departamento está en la locaclización 2:

**SELECT** nombre, id\_dep

**FROM** Empleado

**WHERE** id\_dep IN (**SELECT** id

**FROM** Departamento

**WHERE** localizacion = 2);

Seleccionar todos los nombres de los empleados y su salario siempre que tengan menor salario que los empleados del departamento 5.

**SELECT** nombre, salario

**FROM** Empleado

**WHERE** salario < ALL (**SELECT** salario

**FROM** Empleado

**WHERE** id\_dep = 5);

Conocer el nombre y salario de todos los empleados que cobren lo mismo o más que cualquiera de los empleados del departamento 3:

**SELECT** nombre, salario, id\_dep

**FROM** Empleado

**WHERE** salario >= ANY (**SELECT** salario

**FROM** Empleado

**WHERE** id\_dep = 3);

**Subquery con multiples filas:**

Las subconsultas que hemos visto devuelven un único valor por fila. Sin embargo, es posible encontrar subconsultas que tengan más de un valor por fila, estas pueden ser colocadas en las cláusulas **From, Where o Having.**

Ejemplos:

**SELECT** Salario.salario, Salario.id\_dep

**FROM** (**SELECT** salario, id\_dep

**FROM** Empleado

**WHERE** salario BETWEEN 1000 and 1500) Salario;

Cuando las consultas de varias columnas están colocadas en el **Where** o el **Having**, la comparación debe hacerse de todas las columnas de cada fila. Como en el siguiente ejempo:

**SELECT** nombre, salario

**FROM** Empleado

**WHERE** (salario, id\_dep) in (**SELECT** salario, id\_dep

**FROM** Empleado

**WHERE** salario BETWEEN 1000 and 1500)

**ORDER** **BY** nombre;

**Subquery correlacionada:**

También es posible hacer que los resultados de la consulta interna dependan de valores proporcionados por la consulta externo, esto quiere decir que la consulta interna, se ejecuta una vez por cada fila seleccionada por la consulta principal.

**SELECT** Nombre, Salario, Id\_Departamento

**FROM** Empleado E

**WHERE** Salario > (**SELECT** AVG(Salario)

**FROM** Empleado T

**WHERE** E.Id\_Departamento = T.Id\_Departamento);

**////////// Esto lo saque de internet, mientras el profe movía su camioneta /////////**

**Desde acá empieza con lo temas visto en clase.**

Se agrega comentarios con tres guiones **(###)** y con el **/\* \*/**

**Consulta anidada se agrega en paréntesis**.

En este caso lo que hacemos es agregar un campo con un select para traer una sola línea, en este caso que traiga el nombre de language.

**Use** Sakila (**Recordar siempre que lista queremos utilizar.**)

**Select** F.film.id;

F.tittle;

F.release\_year;

(**Select** L.name **from** language L **where** L.language\_id = F.language\_id) “Idioma”;

**From** film F;

--- **where** F.language\_id != 1; //Para verificar si hay películas sin id 1 que es ingles, y en este caso no existen.

**Diferencia entre Subquery y Join:** Con la consulta anidada me aseguro siempre que me va a traer una fila, y devuelve nulo en caso de no encontrar nada. Mientras el Join, es obligatorio el unir dos tablas y no hay manera de que te traiga nulos, ya que, si una tabla no tiene los valores, no los muestra, y te trae menos registros, por esto es más sencillo y rápido hacerlo desde consulta anidada. Con el Join hay que hacer una consulta especial, para que traiga el dato que me está faltando.

**Otro ejemplo:**

**Select** I.inventory\_id,

I.film\_id,

(**Select** F.title **from** film F **Where** I.film\_id = F.film\_id) “Pelicula”,

I.store\_id,

(**Select** S.manager\_staff\_id **from** store S **where** S.store\_id = I.store\_id) “ID\_Gerente”,

(S**elect** St.first\_name **from** staff St **Where** St.staff\_id = ( **Select** S.manager\_staff\_id **from** store S **where** S.store\_id = I.store\_id) ) “Nombre\_Gerente”,

I.last\_update

**From** inventory I;

**Creación de una base de datos // Creación y Llenado de tabla.**

**Lenguaje DDL** (**Data Definition Language**)**,** es lo que nos permite la creación y administración de la base de datos, no es muy diferente a SQL.

**Lenguaje DML (Data Modification Language)**, este lenguaje es el que estamos usando dentro del SQL, lo que nos permite mostrar la base de datos, por medio de consultas.

**Instrucciones para crear una base de datos:**

**Create Database** prueba2021;

//Esto debe crear una nueva database con ese nombre. Pero para poder verlo, hay que tocar en el botón de actualizar arriba derecha en **Schemas**

**Use** prueba2021;

//Esto es para usar la base de datos.

**Describe** agenda;

//Para obtener información sobre la tabla colocada, mostrando la estructura de la tabla, diciendo que tipo(int, str, etc.), permite nulos, tiene claves, que muestra cuando es nulo y un extra.

**Create Table** agenda

//Sirve para crear una tabla dentro de la base de datos que tengamos en use.

**Create Table** agenda **(**

agenda\_id **integer(8),**

apellido **varchar(20),**

nombre **varchar(30),**

domicilio **varchar(50),**

teléfono **varchar(12)** **)**;

//Solamente usar el tipo de dato númerico cuando se pide calcular algo.

**Insert into** agenda **(**agenda\_id, apellido, nombre, domicilio, teléfono**)**

**Values** (1, “Perez”, “Rigoberto”, “BetaTester”, “+54 (2604)353123”);

//Esto sirve para meterle valores dentro de la tabla

**Dos tareas para antes de la próxima clase y enviarlo al correo del profe:**

**De Tarea, hay que crear una base de datos, y crear dos tablas, además colocar 5 registros dentro, y hay que enviarle código por correo donde que escribimos para crear la base de datos, crear la tabla y los cinco datos.**

**Ademas armar un select en la base de datos de sakila para traer el nombre del gerente utilizando el store\_id en la tabla Cliente (Customer) y utilizando el addres\_id de customer, traer el domicilio y nombre de la ciudad de la tabla de direcciones (Adress)**

**8va Clase (07/05/2021):**

**Vamos a ver un video sobre el uso de WorkBench y como encarar el diagrama de entidad y relación.**

**//Crear DIAGRAMA Entidad Relación en MySQL WorkBench//**

**//**<https://www.youtube.com/watch?v=hfE0_Mme32k>**//**

**//Lista de Reproducción//**

**//**<https://www.youtube.com/watch?v=FQ7XAygh0qA&list=PL1vDASG8ZwjBZMurvLLVjowpw2FyWFQNt&index=1>**//**

**El problema de Join:** Es que te trae toda la tabla para comparar con otra tabla, el subquery te trae solo una columna.

**Lo soluciona el Subquery.**

**No usar el use BaseDeDatos:** Para evitar el uso de esto al iniciar el programa, se puede referenciar a la base de datos antes de colocar la tabla después del **from**.

**from** sakila.payment;

Practicamos consultas anidadas, vista en la clase anterior.

****

//**concat(“Texto1”, “Texto2”):** Sirve para unir texto todo junto.

//Se puede hacer infinitos select anidados, el tema es el performance al realizar la búsqueda.

//**and** “Demora” > “Prestamo Permitido”: Me da todos los valores donde la demara es mayor y lo muestra en la lista.



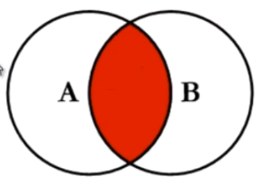
**//Create View “Nombre” (Lista de Campos // Esto no es necesario) AS:** Esto almacena tablas en **VIEWS**, para mostrar cuando llamamos **select \* from Nombre.** Tener en cuenta que esto es una vista nomas, no se puede editar.

**9na Clase (14/05/2021):**

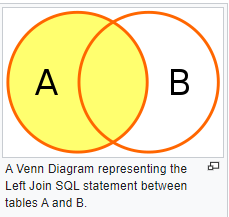
**18:30 se hace una clase de consulta, de media hora.**

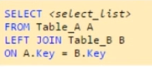
Hoy vamos a ver lo que son los Join, siempre cuando unías dos tablas y tenían un valor de referencia para las dos tablas. Lo vamos a ver en teoría de conjuntos. Vemos un video, el cual es: Inner Join, Left Join, Right Join, full Join / <https://www.youtube.com/watch?v=eNikUI0Y8y8> y vemos un ejemplo <https://www.youtube.com/watch?v=93J8SYhhOoU&list=PLg9145ptuAigFBMVBYEX6YQllCeL9lA8M&index=12>

**Inner Join:** son dos tablas y tienen que tener una tabla en común, lo que se hace es comparar estas dos columnas y encontrar coincidencias. No aparece ningunos valores donde el ID no coincide en las dos tablas. En el código se hace un join normal.

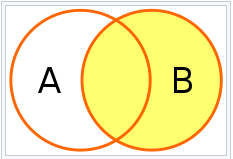


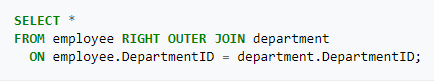
**Left Join (Agarra la primera tabla):** Se parte de todos los elementos de la tabla A y se compara con la tabla B. Es decir, siempre se muestran todos los datos de la fila A y no los de la B.



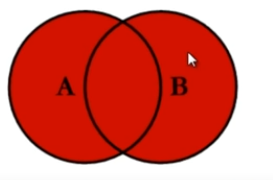


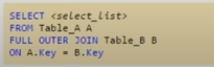
**Right Join (Agarra la segunda tabla):** Se parte de todos los elementos de la tabla B y se compara con la tabla A. Es decir, siempre se muestran todos los datos de la fila B y no los de la A.



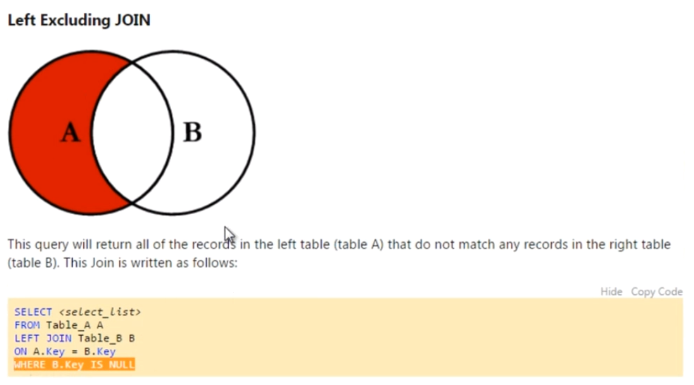


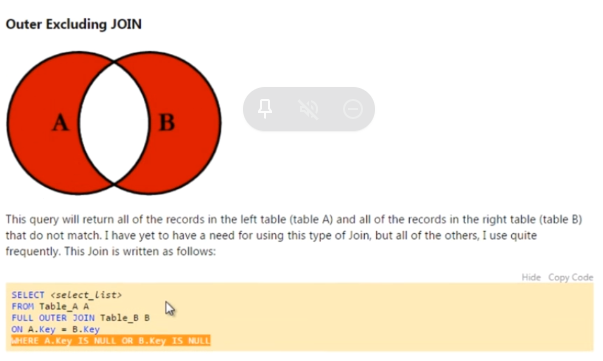
**Full Join:** Muestra todos los elementos de la tabla A con todos los elementos de la tabla B. Mostrando tanto los elementos que no comparten entre ellos. Esto en el sql no existe, ya que te produce un producto cartesiano, y hay posibilidad de romper el servidor.





**Left / Right Excluding Join:** En este caso se muestran todos los datos que no tengan relación entre ellos y que no tengan relación.





Ejemplos en mysql

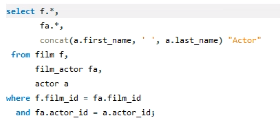
**Probamos juntar tablas con ejemplos de join**



**Probamos una unión de joins.**

****

**Formato inner join en Oracle, y se coloca un (+) según si es right o left:**



**10ma Clase (28/05/2021):**

**Vemos formato de ejemplo, como el cuit. XX-XXXXXXXX-X / 99999999999**

Entonces lo ideal es usar una función que tome la cadena de 11 números, y los transforme con guiones.

**En este caso vamos a probar una función ya creada en sakila:**

**get\_customer\_balance(customer\_id, date) //** En este caso la función lo que hace es mostrar cuanto dinero deben las personas al entregar las películas tarde.

Diferencia entre función y procedimiento:

* **Procedimiento:** No devuelve un valor, sino que puede o no devolver valor. Pero la función tiene que resolver algún problema nuestro.
* **Función:** Devuelve un valor, en este caso lo que hace es metemos datos, realiza la función y devuelve un datos.

**Para llamar a un procedimiento, hay que utilizar el comando Call procedimiento (). En caso de la función**, con solo poner **la función escrito dentro de un select, funciona devolviendo valores**.

Tarea hacer un select que utilice rewards\_report y inventory\_held\_by\_customer.

****

**Codificación dentro de la función:** Tener en cuenta que después de un select, va un **INTO** para devolver ese valor obtenido en el select lo guarde en una variable, también esa variable tiene que ser creada antes.

**Select** X **INTO** devolver\_Valor

**11va Clase (28/05/2021):**

Hacemos un programa en Python que nos deje meter registros en la tabla ya creada en la base de datos de Python.

**12va Clase (11/06/2021):**

**Entregamos el trabajo práctico y vemos lo que es un trigger.**

**https://manuales.guebs.com/mysql-5.0/triggers.html**

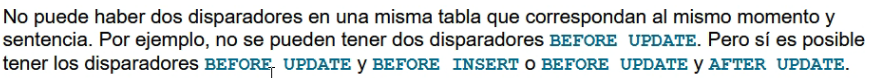
**Trigger:** Es una ejecución a través de una tabla y va siempre acompañando a una tabla, un trigger se dispara siempre que realizamos un evento. Estos eventos son cuando modificamos datos almacenados, es decir, **INSERT, Update** o **Delete.**

A diferencia de una función o un procedimiento, un trigger no puede existir sin una tabla asociada.

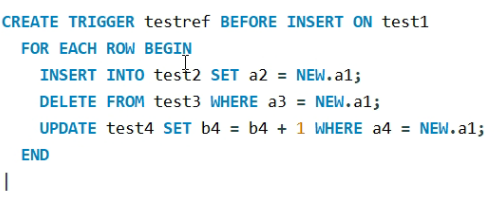


No podemos definir trigger para tabla temporaria (Tabla para hacer una función o un procedimiento o un programa que se destruyen al finalizar el uso) o una vista.

**Momento\_disp:** puede ser en **After** o **Before**

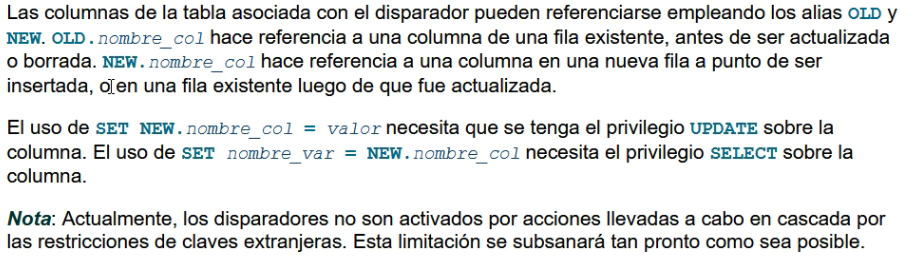
**Evento\_disp:** puede ser **Insert, Update** o **Delete**

**Sentencia\_disp:** Se ejecuta cuando se activa el disparados debe colocarse **BEGIN … END**.



En este caso lo que el trigger hace que al insertar un valor en la tabla test1, hace lo siguiente:

* Inserta en la tabla 2 en la columna a2 le mete un nuevo valor que tenia en la tabla test1 a1.
* Borra en la tabla 3 donde el a3 es igual al valor insertado en el a1.
* Actualiza la tabla 4 donde en la columna b4 es igual al mismo valor de b4 más uno donde a4 es igual al nuevo valor de a1.



**13va Clase (11/06/2021):**

Vemos como se hacen las diferentes tablas, de sueldos, de facturación, etc. Hacemos diagramas y luego transformamos esos diagramas en tablas. Y todo lo que hemos visto en clase.

El final va a ser hacer un sistema partiendo de una explicación textual. Hacer modelo relacional y hacer las tablas en workbench. Puede también tomar esto, es o lo anterior o esto: También se va a preguntar para acceder a consultas y realizar vistas para mostrar datos. Y también se tiene que preparar una lección oral con algún tema y si hay alguna duda con la lección se hacen unas preguntas.

Entregar el trabajo final para el jueves 24.