

**Construcción del modelo lógico:** transforma el modelo conceptual hacia instrucciones que podemos implementar en una computadora.

**Tipos del modelo lógico:**

OO

RELACIONAL. (SE USA ACTUALMENTE). Se definen tablas. Cualquier entidad del modelo lógico es una tabla en el modelo físico.

JERÁRQUICO.

RED (LOS DOS ULTIMOS SON OBSOLETOS)

**Entradas para construir el modelo lógico:**

-esquema/modelo conceptual.

-Hay reglas en este modelo:

-Criterio de usos:

-Criterio de carga:

**No hay atributos compuestos, polivalentes y jerarquías en este modelo. (hay que reemplazarlos).**

**atributo derivado (si se usa mucho que se tenga al atributo y si se usa poco sacarlo total lo vas a podes calcular al mismo) y ciclos (dependiendo si se generan o no repetición de información en el modelo para el caso del ciclo me refiero)** atentan contra la minimalidad.

**Los atributos derivados** representan la primera causa que afecta la minimalidad. Un atributo derivado es un atributo que aparece en el modelo, pero cuya información, si no existiera almacenada en él, podría igualmente ser obtenida

**Los ciclos** de relaciones presentan la segunda causa que afecta la minimalidad. Se dice que existe un ciclo cuando una entidad A está relacionada con una entidad B, la cual está relacionada con una entidad C, la que a su vez se relaciona con la entidad A. No todos los ciclos afectan la minimalidad. Un ciclo afecta la minimalidad cuando una de las relaciones existentes puede ser quitada del esquema, y aun en esas condiciones, el modelo sigue representando la misma información.

**Cuando en un ciclo todas las relaciones son de muchos a muchos NO HAY NINGÚN PROBLEMA. (no hay ningún caso en que puedas sacar una relación y que dicha relación se pueda realizar por otro camino “mas largo”).**

El problema está cuando las relaciones no son de muchos a muchos. Por lo que se te pueden presentar casos en el cual la relación entre dos entidades puede sacarse, y de la misma manera seguir teniendo esa misma relación pero por otro camino “mas largo”. (osea que podes sacar esa relación o dejarla, depende el caso del problema (si vos usas mucho esa relación la dejas, sino la sacas y en el caso en el que la necesites tomas ese camino “mas largo”)).

### Soluciones frente a la falta de atributos compuestos:

1. Seria poner todos los atributos simples juntos en un string.
2. Creas una nueva entidad pero con los atributos simples que compone al atributo compuesto. (relacionando con la primera entidad).(LA INDICADA).
3. El atributo compuesto lo pones como una entidad aparte relacionada con la entidad donde antes estaba el atributo compuesto. (no recomendada).

(saque foto).

### DE LA DIAPOSITIVA:

· Atributos compuestos

· Generar un unico atributo que se convierta en la concatenación de todos los atributos simples que contiene el compuesto.

· Definir los atributos simples sin un atributo compuesto que resuma. La cantidad de atributos aumenta pero esta solución permite definir cada uno de los datos en forma independiente.

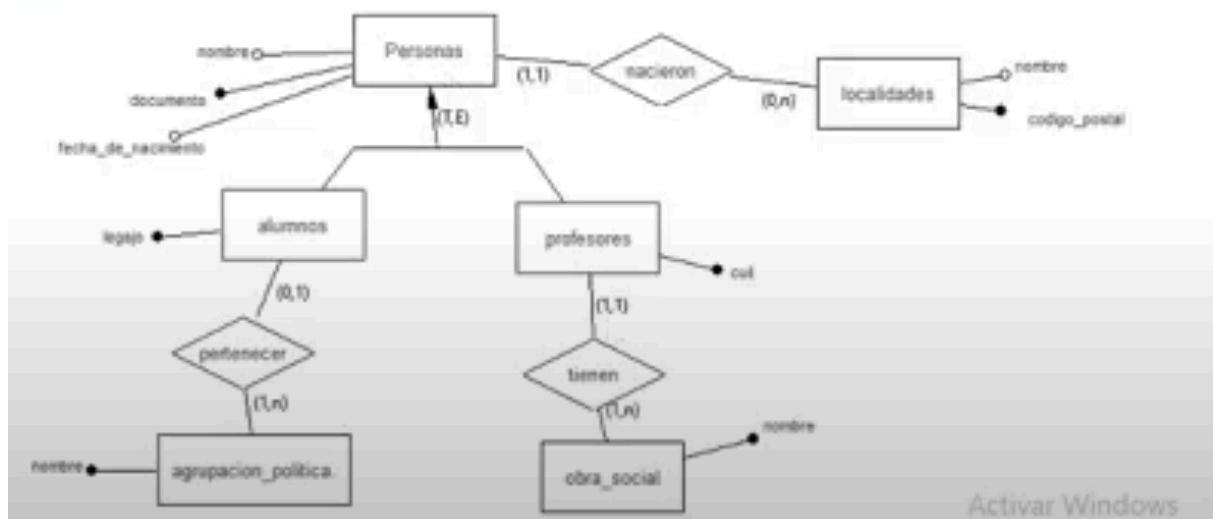
· Generar una nueva entidad, la que representa el atributo compuesto, conformada por cada uno de los atributos simples que contiene. Esta nueva entidad debe estar relacionada con la entidad a la cual pertenecía el atributo compuesto.

**Para el caso de los atributos polivalentes** creas una entidad (por ej titulos que es de (0,n) lo pones como una entidad que se relaciona con la entidad en donde estaba antes como atributo (le tenes que poner un identificador). (primer ejemplo en la carpeta).

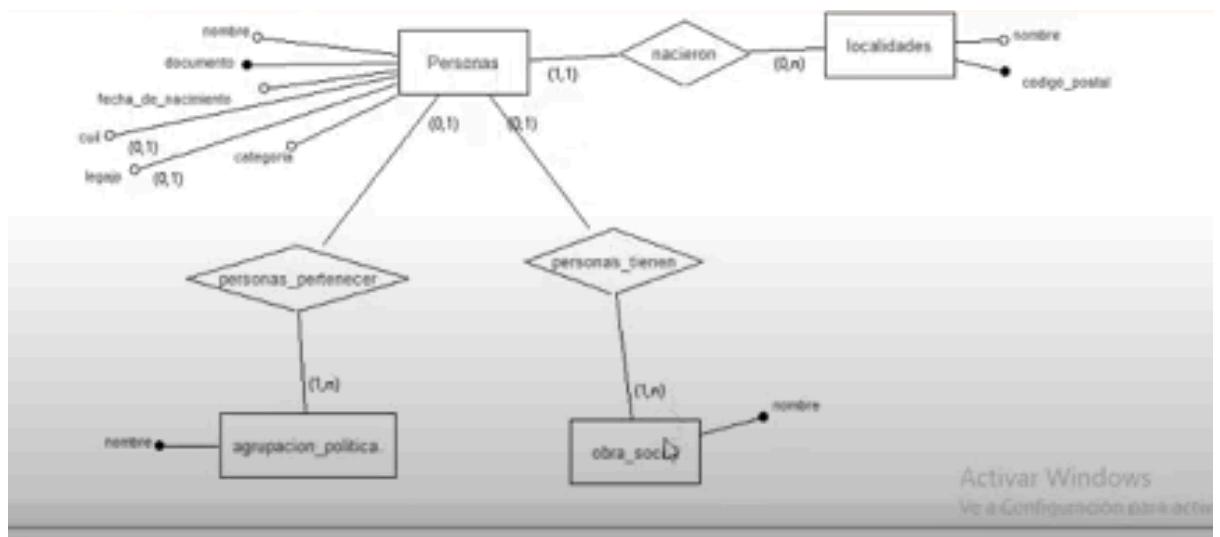
**Para el caso de jerarquías** no se pueden usar porque no hay el concepto de herencia.

Tres soluciones: Matar al padre, matar al hijo, o dejarlos básicamente.

1. Dejar solo al padre y las relaciones de los hijos van al padre (atributos de los hijos).(los atributos de los hijos deben ser opcionales).Las relaciones que tenian los hijos con otras entidades se mantienen, pero seguramente cambie su cardinalidad.  
EJ:

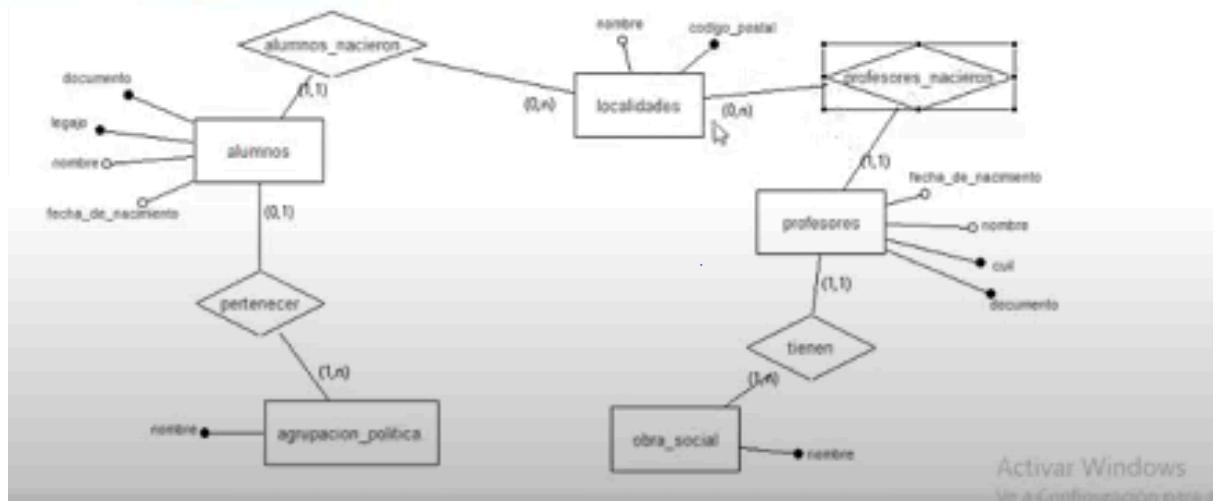


Activar Windows



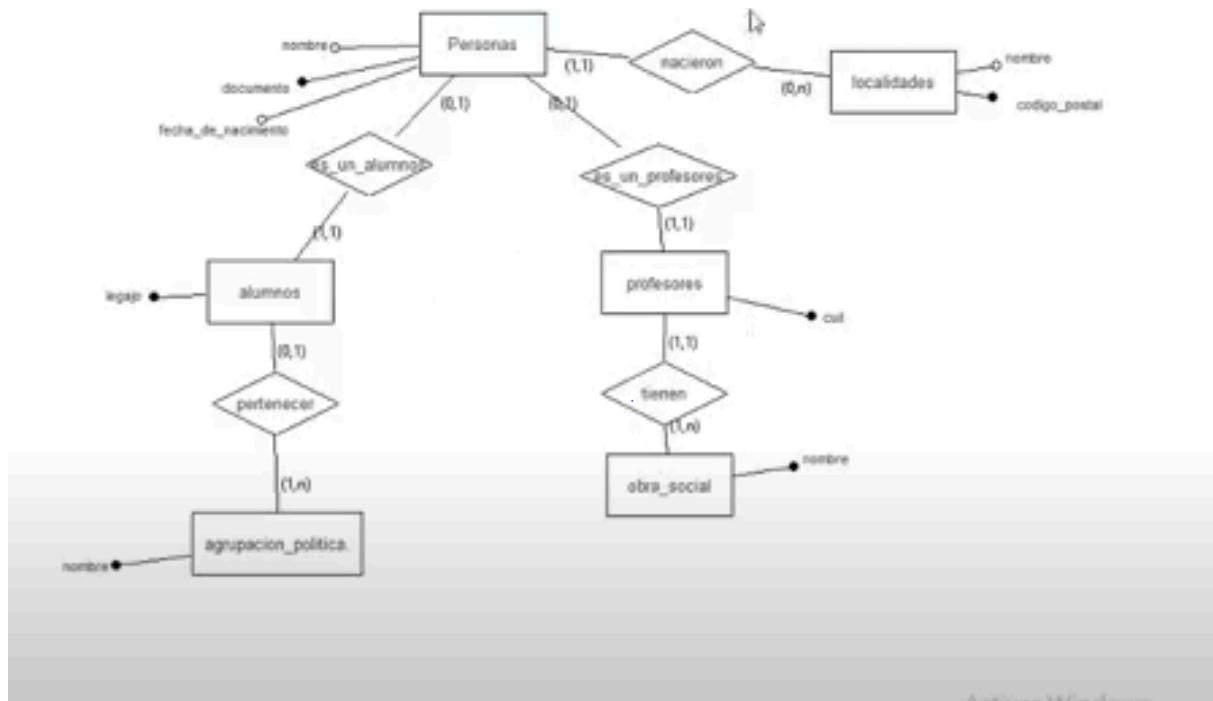
- Dejar a los hijos y los atributos del padre bajan a los hijos. (con cobertura parcial esta no sirve).

Ej: (mismo caso que el de arriba luego de aplicarle este metodo)



- Dejar al padre y a los hijos y convertir la jerarquía en relaciones uno a uno entre el padre y cada uno de los hijos. Esta solución permite que las entidades que conforman la jerarquía, mantengan sus atributos originales generando la relación explícita ES\_UN entre padre e hijo. Podes identificar qué va a ser la entidad padre (por ej tenes como padre persona y como hijos tenes empleado, administrador, etc) se va a dar cuenta mediante un nuevo atributo en la entidad padre llamado por ej "tipo" que indica qué sería (en el ejemplo que dije indicaría si es empleado o administrador). (solución mejor).

Ej: (basado en el ejemplo de los de arriba)



## MODELO FISICO:

Cada entidad se convierte en tablas. Cada tabla SI O SI debe tener una clave primaria.

Hay que definir las claves, primaria y univoca. (el concepto de identificador se va, se lo transforma en clave primaria y univoca). Las secundarias todavía no. No conviene elegir como clave primaria a un identificador.

Tabla= ID, atributo1,atributo2,atributo3.

tabla=relación.

**relaciones:** Relaciones n a n: se convierte en tabla.

Relaciones 1 a n: deberían convertirse en tabla. Si del lado del "1" del 1 a n es (1,1) no se convierte en tabla pero si es (0,1) hay que tomar una decisión.

Relaciones 1 a 1: si fuese (1,1) a (1,1) la regla es que se hace 1 sola entidad para todo. (todo en 1 tabla). (0,1) es un (1,1) (por jerarquía). Cualquier otra variante habría que analizarla.

## Clase teórica:

Las tablas tienen atributos y en los atributos debe tener dominio.

Las relaciones/tablas contienen tuplas.

Toda tabla debe tener clave primaria. Dejo el manejo de las claves primarias al dbms.

Pasos:

Eliminación de identificadores externos.

Selección de claves primarias y candidatas.

Conversión de entidades. (se convierten en tabla)

Relaciones (se refiere a las que tenias en los modelos conceptuales, la interrelación tamb se dice). Se resuelven en función de la cardinalidad: **De muchos a muchos** (todas se resuelven igual (ya sea (1,n) a (1,n) o (0,n) a (0,n) etc)) la relación se convierte en una tabla. Osea tenes una relación con 2 entidades, entonces vas a tener 3 tablas, una para la relación y 2 para las entidades. En la tabla de la relación vas a tener como atributos a las claves primarias de las 2 entidades (y si tiene atributos de la relación también lo pones). Después como clave primaria de la tabla de la relación tenes que fijarte que claves podes combinar para generar la clave primaria (o un atributo autoincremental y fue). A la tabla relación es mejor ponerle como nombre el nombre de las 2 entidades en vez del que tiene en el modelo lógico/conceptual (ej alumnos\_materias).

**Uno a muchos.** (1,1) (1,n) no se convierte tabla para la relación (la parte del 1,1 la tabla debe contener la clave primaria de la otra entidad donde se relaciona (explicado en el video de practica 3.1 minuto 17)). (1,1) a (0,n) pasa lo mismo. Si es (0,1) a (1,n)/(0,n) depende el caso, si se aceptan valores null no hace falta crear tabla, sino si.

**Uno a uno.** Se crea 1 sola tabla para todo, pones todo junto. Los dos últimos atributos deben aceptar valores nulos.

## Modelo Fisico



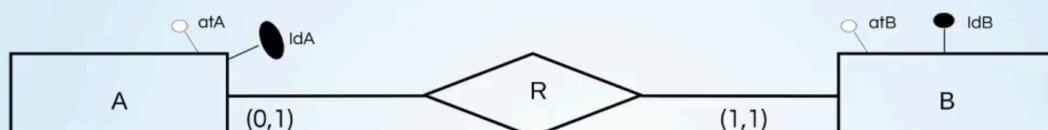
- Relaciones uno a uno
    - Un elemento de remito solo existe a partir de una factura

- ▶ Facturas = (id\_factura, tipo, numero, fecha, montototal, nro\_remito, fecharemito)
  - ▶ Los dos últimos atributos deben aceptar valores nulos
  - ▶ Único caso donde no se genera una tabla a partir de una entidad

DBD Clase 4

Remito no hace falta que se transforme en tabla.

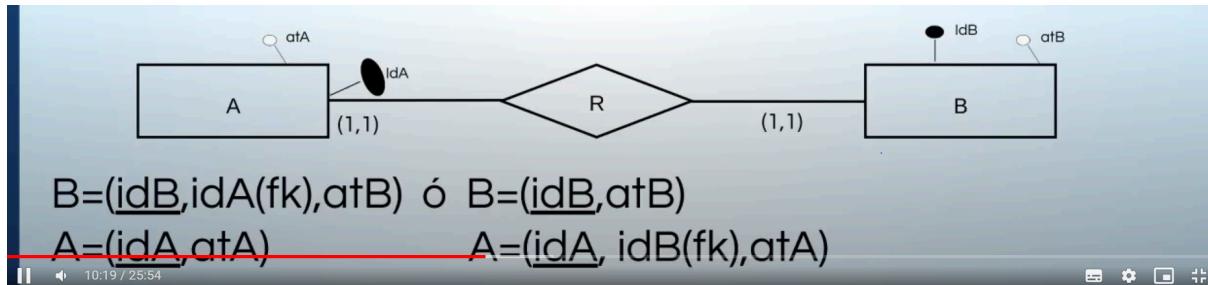
# Conversion de Relaciones



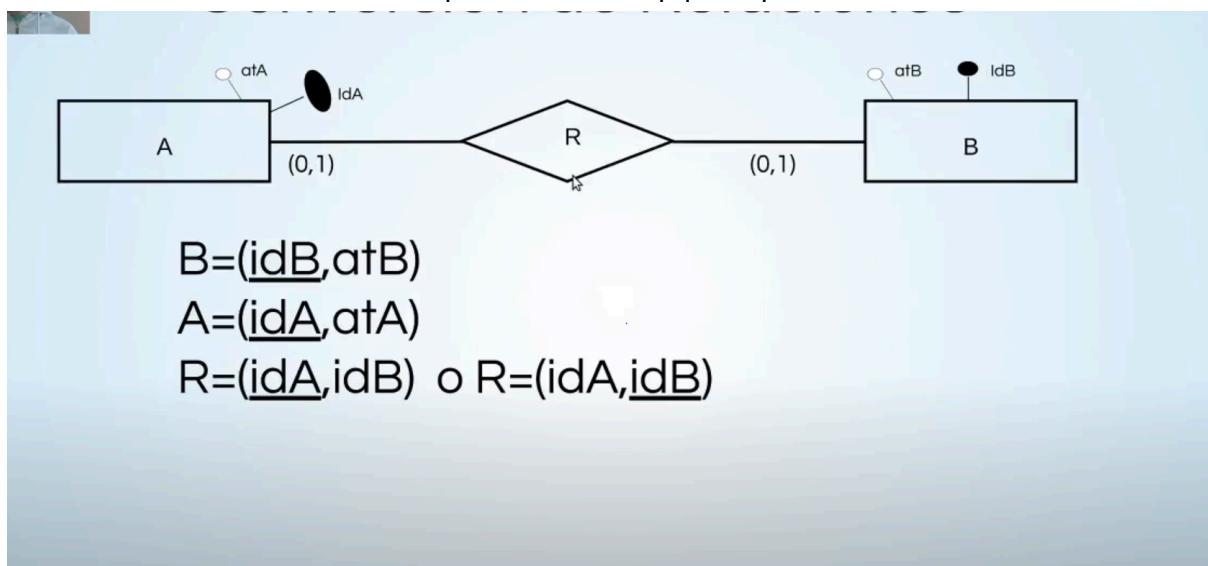
B=(idB,idA(fk), atB)

$A = (\text{id}_A, \text{gt}_A)$

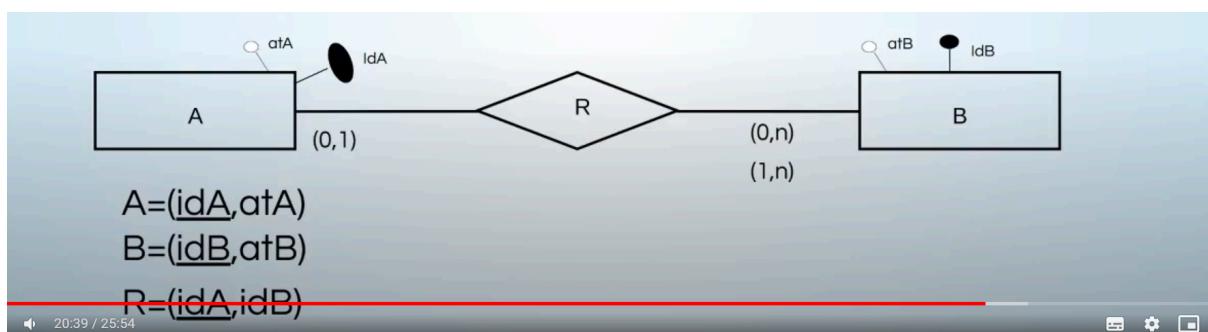
En este caso, la entidad A no siempre va a estar relacionada con la entidad B, por ende si le paso a A la id de B puede que sea nulo, y eso no podes tener nulo dijo el profe, por ende como B siempre se va a relacionar con A, a B le pasas el id de A.



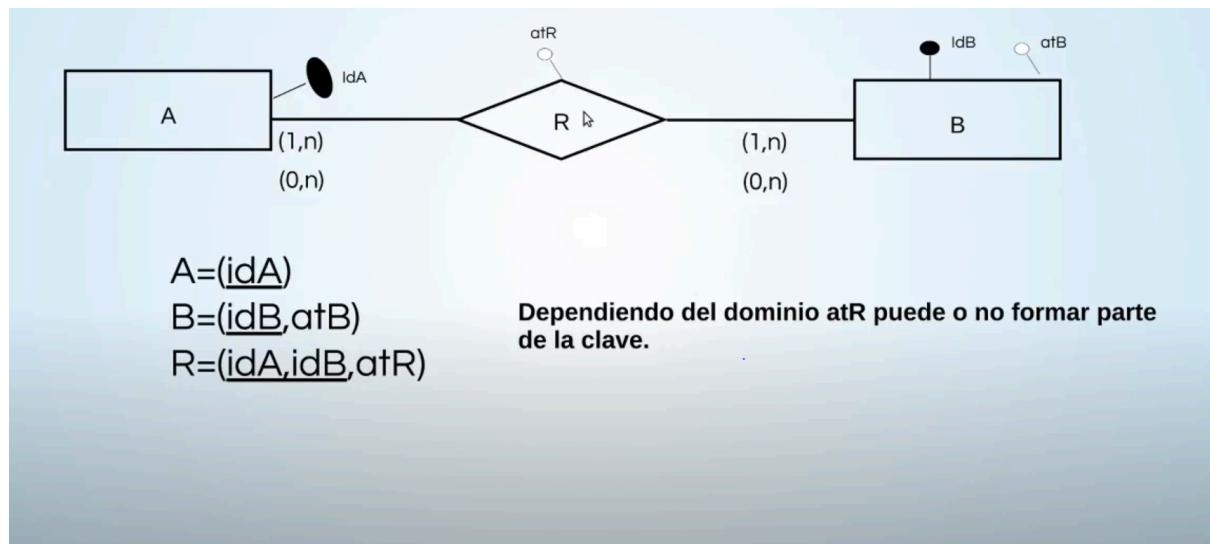
En el caso de arriba es indistinto que caso tomes ,pq siempre se van a relacionar.



En el caso de arriba tenes que crear si o si una tabla de la relación donde contenga el id de A y B ya que si pones los dos identificadores en la tabla de la relación, los dos identificadores van a tener valor porque si estan en la relación quiere decir que estan relacionados A con B (yo pondria a su vez como clave primaria la autoincremental pero bueno).



En el caso de arriba al momento de elegir la clave primaria de la tabla de la relación elegis a idA pq es la que no se va a repetir porque sabes que va a existir una sola en tu archivo conformado por filas y columnas. Ya que idA se relaciona con 1 de B solamente, en cambio B se puede relacionar con muchos de A, y se puede repetir y por ende no es clave primaria.



En el caso de arriba vos unis los identificadores de A y B en la tabla de la relación ya que si elegis cualquiera de ellas (dado el análisis de los otros ejemplos) se puede repetir la información, lo cual no sería correcto.

### Relaciones recursivas:

**Modelo Físico**

- Relaciones recursivas
  - Se tratan igual que las binarias
  - Parcial del lado de uno en este ejemplo
    - Opción 1 atributo nulo
      - Empleados (id\_empleado, nombre, dni, legajo, fecha\_nacimiento, id\_jefe)
    - Opción 2 sin atributo nulo
      - Empleados (id\_empleado, nombre, dni, legajo, fecha\_nacimiento)
      - Ejefe (id\_empleado, id\_jefe)

(0,1) tiene jefe  
 (0,n) tiene a cargo

Opción 2 sería para la práctica creo.

### Integridad referencial:

Asegura que un valor que aparece para un atributo en una tabla, aparezca además en otra tabla.

Se da entre dos tablas cuando un atributo es clave primaria en una de esas tablas y es clave foránea en la otra tabla. Clave foránea significa una clave que aparece en otra tabla y puede tener duplicados (en general estas claves son secundarias). **Esta IR maneja (lo que se puede hacer o no en altas,bajas y modificaciones (define restricciones))** y

**manipula las altas, bajas y modificaciones.** Ante una baja se define cuatro cosas (elegis solo 1):

Restringir. No dejar borrar por ej si queres borrar una clave de un alumno que tambien está en otra tabla. Pq podes perder la integridad de la información.

Cascada. Borrar una clave de todas las tablas en la que esta misma se encuentre.

Null. borras información de por ej una tabla de alumnos ( el id de un alumno 14 borras) entonces en todas las tablas donde ese id aparece, lo pone en null.

No dar bola. No se controla nada. (Queres borrar algo? hazelo, pero no controlo una chota).

Con las modificaciones se puede aplicar lo mismo, salvo que quieras modificar una clave primaria autoincremental, ya que no lo puede modificar nada.

### **RE TOMA DE NOTA DEL MODELO FISICO EN CASA:**

El tipo de datos que describe los tipos de valores de un atributo se denomina dominio.

Identificador es sinonimo de clave univoca/candidata.

Clave primaria es con hash y clave univoca es con árboles.

Hay problemas a la hora de definir cual de los atributos de una entidad que tenía muchos identificadores va a hacer la clave primaria, por ende está el autoincremental.

**Autoincremental:** definir para cada tabla un nuevo atributo que no está presente en el modelo. Ese atributo es manipulado por el dbms, este le va a poner valor la cual NO ADMITE CAMBIOS (lo podes usar pero no modificar, solo lo conoce los programadores, el usuario no) y no se repite ese valor.. No esta propenso a error pq lo maneja el dbms y vas a poder hacer búsquedas re eficientes por ese valor y el usuario no sabe de su existencia, por lo que no está propenso a errores por parte del usuario. Vas a tener una clave primaria que se maneja automáticamente. Ese atributo lo pones como id\_nombreDeLaTablaEnSingular. (ej id\_alumno y la tabla se llama alumnos).

El atributo autoincremental **NO** es obligatorio.

El null no se puede comparar con un valor. te da NADA si comparas, ni falso ni verdadero. null es nada.

### **Restricciones:**

**Del dominio:** Si vos decis que vas a tener atributos integer, solo se va a permitir eso.

**De clave:** Evita que el valor del atributo clave (primaria, univoca, candidata) genere valores repetidos.

**Sobre nulos:** Evita que un atributo tome nulo en caso de no ingresarle valor.

**De integridad:** Ningún valor de la clave primaria puede ser nulo.

**De integridad referencial.**

Las operaciones de alta, baja y modificaciones pueden generar violaciones a las restricciones mencionadas. Si se viola alguna regla, la operación (alta, baja o modificación) no se hace, se rechaza.

## Restricciones

Las operaciones de Alta, Baja y Modificación (ABM) pueden generar violaciones a las restricciones anteriores.

- Alta
  - Puede violar: valor nulo para clave, repetición de la clave, integridad referencial, restricciones de dominio.
  - Si se viola la regla, la operación se rechaza
- Baja
  - Puede violar: integridad referencial (se procede como en el caso anterior)
- Modificación
  - Puede violar: cualquiera de las operaciones.

DBD Clase 4

En la de modificacion, para el caso de restriccion de dominio, queres modificar un atributo que es intiger, y le pones un string, ahí te salta esa restriccción y se rechaza la operacion.

La de la baja con respecto a la integridad referencial, seria por ej cuando queres borrar una información que esta en varias tablas, lo borras en 1 sola tabla, estas violando la integridad referencial y salta la restriccción cancelando la operación.

### **CLASE TEORICA 5.1 DEL VIDEO:**

Algebra relacional: Son operaciones entre uno o mas tablas de entrada que generan una nueva tabla (temporaria) como resultado.

#### **6 operaciones basicas:**

Las unitarias operan sobre 1 tabla, mientras que las otras operan sobre 2 tablas.

**Selección:** Selecciona tuplas que satisfacen un predicado dado.  $\sigma$

•  $\sigma$  Estadocivil = "casado" (asociados)

Seleccionas todas las tuplas de la tabla asociados donde estadocivil sea igual a casado.

$\pi$

**Proyección:** Devuelve la tabla que se va a operar con columnas omitidas.

Descartas los atributos que no son necesarios de una tabla basicamente (descartas columnas).

- Ejemplo 4: nombres de los asociados
  - $\pi_{\text{nombre}} (\text{asociados})$

**Producto cartesiano:**

**TABLA1 X TABLA 2.**

**T1 x T2**

Produce una tabla concatenando cada tupla de T1 con todas las tuplas de T2.

Tabla futbol Tabla tenis

Nombre	Apellido
Pedro	Troglio
Carlos	Griguol
Facu	Oreja

Nombre	Apellido
Rafael	Nadal
Gustavo	Lopez

**futbol x tenis**

Futbol.nombre	Futbol.apellido	Tenis.nombre	Tenis.apellido
Pedro	Troglio	Rafael	Nadal
Pedro	Troglio	Gustavo	Lopez
Carlos	Griguol	Rafael	Nadal
Carlos	Griguol	Gustavo	Lopez
Facu	Oreja	Rafael	Nadal
Facu	Oreja	Gustavo	Lopez

**Producto Cartesiano:**

- Ejemplo 6: mostrar las sedes de La Plata
  - $\pi_{\text{sede.nombre}} (\sigma_{\text{sede.idlocalidad} = \text{localidad.idlocalidad}} \wedge \sigma_{\text{localidad.nombre} = "La Plata"} (\text{sedes} \times \text{localidad}))$
  - $\pi_{\text{sede.nombre}} (\sigma_{\text{sede.idlocalidad} = \text{localidad.idlocalidad}} (\text{sedes} \times \sigma_{\text{localidad.nombre} = "La Plata"} (\text{localidad})))$
- Ejemplo 7: mostrar cada deporte y el nombre del asociado que lo practica.
  - $\pi_{\text{asociado.nombre}, \text{deporte.nombre}} (\sigma_{\text{asociado.idsocio} = \text{practica.idsocio} \text{ and } \text{deportes.iddeporte} = \text{practica.ideporte}} (\text{Asociados} \times \text{practica} \times \text{deportes}))$

**Renombrar (operación unitaria):**  Se usa ante una consulta donde vos necesitas usar la tabla mas de una vez.

## Renombrar:

- permite utilizar la misma tabla en, por ej., producto cartesiano. Operación **P**
- Ejemplo 7: mostrar todos los asociados que viven en la misma dirección que el socio con id 75
  - $\sigma_{\text{asociados}.idsocio=75}(\text{Asociados})$
  - 2.  $(\sigma_{\text{asociados}.idsocio=75}(\text{Asociados}) \times \text{asociados}) \rightarrow ???$
  - 3.  $\sigma_{\text{asociados}.direccion = ???}(\sigma_{\text{asociados}.idsocio=75}(\text{Asociados}) \times \text{asociados}) \rightarrow ???$
  - 4.  $\pi_{\text{aso.nombre}}(\sigma_{\text{asociados}.iddireccion = aso.iddireccion}(\sigma_{\text{asociado}.idsocio=75}(\text{Asociados}) \times \text{Paso}(\text{asociados}))$

## Unión:

### Operaciones - Unión T1 u T2

Produce una tabla que contiene todas las tuplas de T1 más todas las de T2, eliminando tuplas duplicadas. T1 y T2 deben ser compatibles (sus esquemas deben ser equivalentes en la cantidad, posición y dominio de los atributos, aunque sus nombres sí pueden ser distintos).

Tabla futbol

Nombre	Apellido
Pedro	Troglío
Carlos	Griguol
Gustavo	Lopez

Tabla tenis

Nombre	Apellido
Rafael	Nadal
Roger	Federer
Carlos	Griguol

futbol U tenis

Nombre	Apellido
Pedro	Troglío
Carlos	Griguol
Gustavo	Lopez
Rafael	Nadal
Roger	Federer

En la misma posición significa que deben estar en el mismo orden los atributos. Deben tener la misma cantidad de atributos, en el mismo orden, con un mismo dominio. (se compara el primer atributo de la primera tabla con el primer atributo de la segunda tabla y así siguiendo.)

## Unión:

- tupas comunes a dos relaciones, equivalente a la unión matemática. Debe efectuarse entre relaciones con sentido. Operación  $\cup$
- Ejemplo 8: asociados que practiquen vóley o futbol
  - $\sigma_{deporte.nombre = "futbol"}(\text{deportes})$
  - $\sigma_{deporte.nombre = "futbol"}(\text{deportes}) \times (\text{practica} \times \text{asociados})$
  - $\sigma_{asociado.idsocio = practica.idsocio \text{ and } deportes.iddeporte = practica.ideporte}(\sigma_{deporte.nombre = "futbol"}(\text{deportes}) \times (\text{practica} \times \text{asociados}))$
  - $\pi_{asociado.nombre}(\sigma_{asociado.idsocio = practica.idsocio \text{ and } deportes.iddeporte = practica.ideporte}(\sigma_{deporte.nombre = "futbol"}(\text{deportes}) \times (\text{practica} \times \text{asociados})))$
  - $\sigma_{deporte.nombre = "voley"}(\text{deportes})$
  - ....
  - $\pi_{asociado.nombre}(\sigma_{asociado.idsocio = practica.idsocio \text{ and } deportes.iddeporte = practica.ideporte}(\sigma_{deporte.nombre = "voley"}(\text{deportes}) \times (\text{practica} \times \text{asociados})))$
  - $\pi_{asociado.nombre}(\sigma_{asociado.idsocio = practica.idsocio \text{ and } deportes.iddeporte = practica.ideporte}(\sigma_{deporte.nombre = "futbol"}(\text{deportes}) \times \text{practica} \times \text{asociados})) \cup$
  - $\pi_{asociado.nombre}(\sigma_{asociado.idsocio = practica.idsocio \text{ and } deportes.iddeporte = practica.ideporte}(\sigma_{deporte.nombre = "voley"}(\text{deportes}) \times \text{practica} \times \text{asociados}))$

DBD - CLASE 5

**Diferencia:** Las tablas tienen que tener la misma estructura (dominio supongo), misma cantidad de atributos y en el mismo orden.

## Operaciones – Diferencia

### T1 – T2

Produce una tabla que contiene todas las tuplas de T1 que no se encuentran en T2. T1 y T2 deben tener esquemas compatibles.

Tabla futbol

Nombre	Apellido
Pedro	Troglio
Carlos	Griguol
Facu	Oreja

Tabla tenis

Nombre	Apellido
Pedro	Gonzalez
Pedro	Troglio

### futbol – tenis

Nombre	Apellido
Carlos	Griguol
Facu	Oreja

## HAY BUEN EJEMPLO EN LA TEORIA.

**Producto natural:** (Es lo mismo que el producto cartesiano + la selección que le hacías para que no se repita información o que no te quedase información inservible). Junta las filas de dos conjuntos (tablas) que tengan sentido. Deben tener atributo en común. Se puede hacer producto natural si tienen al menos 1 valor igual en un mismo atributo que tiene el mismo nombre en ambas tablas.

## Operaciones – producto natural T1 $\times$ T2

Produce una tabla concatenando tuplas de ambas tablas que tengan valores iguales en atributos con igual nombre (equicombinación). Se elimina uno de los ejemplares de cada atributo común.

Tabla tenis

Tabla futbol

Nombre	Apellido
Pedro	Troglio
Carlos	Griguol
Facu	Oreja

Nombre	Apellido
Pedro	Troglio
Pedro	Gonzalez

futbol  $\times$  tenis

Futbol.Nombre	Futbol.Apellido
Pedro	Troglio

Tabla tenis

Nombre	Apellido1
Pedro	Gonzalez
Pedro	Troglio

futbol  $\times$  tenis

Futbol.Nombre	Futbol.Apellido	Tenis.apellido1
Pedro	Troglio	Gonzalez
Pedro	Troglio	Troglio

## Intersección:

## Operaciones - intersección

T1  $\cap$  T2

Produce una tabla que contiene todas las tuplas que se encuentran tanto en T1 como en T2. T1 y T2 deben tener esquemas compatibles.

Tabla futbol

Nombre	Apellido
Pedro	Troglio
Carlos	Griguol
Gustavo	Lopez

Tabla tenis

Nombre	Apellido
Rafael	Nadal
Gustavo	Lopez

futbol  $\cap$  tenis

Nombre	Apellido
Gustavo	Lopez

**División:** Devuelve una tupla con los **campos** que están en la tabla 1 y no están en la tabla 2. Además, el valor de ese o esos campos se van a tener que corresponder con todos los valores de la segunda tabla. Es usada cuando se nos pide que devolvamos valores que cumplen condiciones para todos los casos.

Ej:



## Ejemplo práctico

**PRODUCTO** = (idproducto, nombre, códigobarra, preciocosto)

**CLIENTE** = (idcliente, nombre, dirección, idlocalidad(FK))

**FACTURA** = (idfactura, fecha, montofactura, idcliente(FK))

**RENGLON** = (idfactura, renglon, idproducto(FK), precioventa, cantidad)

**LOCALIDAD** = (idlocalidad, descripción)

- Nombre de clientes que compraron todos los productos

$$(\pi_{\text{idProducto}, \text{nombre}} (\text{Cliente} \sqcup \text{Factura} \sqcup \text{Renglón})) \% \pi_{\text{idproducto}} (\text{Producto})$$

En este caso se van a quedar atributos que están del lado izquierdo del % la cual no están del lado derecho del %, osea con todos los nombres en los cuales se correspondan con todos los productos de la tabla de la derecha del % a través del atributo idProducto que es el que tiene en común el atributo nombre, pq sino no lo podría hacer.

### T1 % T2

Produce una tabla con los campos de T1-T2 (están en T1 y no en T2), donde los valores en esos campos de T1 se corresponden con TODAS las tuplas en T2. El esquema de T2 debe estar incluido en T1.

Tabla futbol

Nombre	Apellido
Pedro	Troglio
Carlos	Griguol
Pedro	Gonzalez

Tabla tenis

Apellido
Gonzalez
Troglio

**futbol ÷ tenis**

Nombre
Pedro

**Asignación:** Son variables básicamente, le asignas a una variable.

## Operaciones – Asignación

A ← Consulta

Vuelca a A los resultados de CONSULTA. Luego puedo utilizar A.

Después lo de acá abajo es sobre las tablas.

# Álgebra Relacional

## Operaciones de Updates:

- Agregar tuplas
  - $r \Leftarrow r \cup E$  ( $r$  relación  nueva tupla)
  - Deportes  $\Leftarrow$  deportes  ("golf", 5000, 21 )
- Eliminar tuplas
  - $r \Leftarrow r - E$
  - Deportes  $\Leftarrow$  deportes - ("bochas", 500, 1 )
- Actualización de datos
  - $\delta_A \Leftarrow_E (r)$
  - Ej:  $\delta_{\text{saldo}} \Leftarrow_{\text{saldo} * 1.05}$  ( depósito )

En la actualización de datos, deposito es una tabla y saldo es un atributo.



## Actualización de tablas

Producto =  $(\text{codProd}, \text{desc}, \text{existAct}, \text{existMin}, \text{pVAct})$

❑ Incorporar el producto (1235, "tuerca de 9 mm", 10, 50,\$10):  
Producto  $\Leftarrow$  Producto  $\cup \{(1235, \text{"tuerca de 9 mm"}, 10, 50, \$10)\}$

❑ Eliminar el producto 893:  
Producto  $\Leftarrow$  Producto -  $\sigma_{\text{codProd}=893}(\text{Producto})$

❑ Aumentar el 1% el precio de venta actual de todos los productos:  
 $\delta \text{ pVAct} \Leftarrow \text{pVAct} * 1,01(\text{Producto})$

### **CLASE PRACTICA TOMA DE NOTA:**

. En la practica solo se centra en modificar los atributos polivalentes, compuestos y jerarquias al momento de realizar el modelo logico.

En la jerarquia, cuando aplicas la tercera forma que anote en el modelo logico, y te queda una entidad que no tiene identificador, le bajas uno de la entidad "padre", **SE LO PODES**

**BAJAR DE UNA Y SIN UNIRLO CON NINGUN OTRO ATRIBUTO DE ESA ENTIDAD “HIJA”.**

**CLASE TEORICA:**

Normalización: técnica de diseño de bd que comienza examinando las relaciones que existen entre los atributos. Te permite generar tablas mas seguras que no permitan repetición.

dependencia multivaluada: Se da entre atributos A B y C en una relación de modo que para cada valor de A hay un conjunto de valores de b y c sin embargo los conjuntos b y c no tienen nada entre si. (A determina a B y C, pero B y C no tienen nada entre sí). Además el par A con B determina a C, y que A y C determina a B.  $A \rightarrow \rightarrow B$  (A multidetermina a B).

Dependencia multivaluadas trivial es una dependencia que vos no la podes descomponer más y sigue valiendo. ej (  $A \rightarrow \rightarrow C$  (trivial) y  $(A,B) \rightarrow \rightarrow C$  (NO ES TRIVIAL PQ SE PUEDE DESCOMPONER)). Solución a esto, tener dos tablas.

Algebra relacional:

Lenguaje de consulta:

Procedurales: decis “que” quiero obtener de la bd y como se va a operar”

No procedurales: me limito a decir “que” quiero y el “como” es un problema del dbms.

Lenguajes mas eficientes son los procedurales.

Vamos a ver el procedural.

Se agrega, modifica, y se borra de una sola forma.

tabla=relación.

Operaciones de una o 2 tablas que genera una tabla como resultado (en álgebra relacional).

Operaciones fundamentales (SOLO PARA CONSULTAS): unitarias y binarias. Las de las unitarias se aplican a las tablas.

Selección: Se anota con el “sigma” es como una T. (carpeta)

Proyección: devuelve la relación argumento (carpeta).

Renombrar: Renombra una tabla cuando estas trabajando en un producto cartesiano con tu misma tabla ( $t_1 \times t_1$ ). Se usa la “P”.

Producto cartesiano: Se anota con una X. Conecta dos entidades de acuerdo a la definición matemática de la operación. ( $TABLA1 \times TABLA2$ ) (te las junta básicamente). (carpeta).

El producto cartesiano es la operación más ineficiente de todas, te genera tuplas que no las vas a usar pq no te sirven.

No importa la eficiencia de la consulta.

Union: Debo practicar la union entre dos tablas que tengan sentido. Deben tener la misma cantidad de atributos y cada uno en el mismo orden deben ser del mismo tipo (osea integer con integer, string con string, etc). El iésimo atributo de la tabla 1 debe ser compatible con el iésimo atributo de la tabla2.

Diferencia: A-B te da lo de A que no está en B. Las dos tablas deben ser de unión compatibles.

Podes asignarle lo que haces a una variable temporal. (asignación temporal).

Operaciones adicionales:

Producto natural: No juntas todo como en producto cartesiano, sino lo que tiene sentido que se junte entre 2 tablas por ej. (carpeta ej). Entre las 2 tablas debe haber dos atributos que se llamen igual.

### Teoria 4.2 apuntes:

#### Dependencia funcional:

.Se da cuando hay una dependencia entre 2 atributos de una misma tabla. Es otra restricción a las antes mencionadas. En la diapositiva "r" seria una tabla. Si existe una dependencia funcional es una restricción sobre las posibles tuplas que yo podria tener.

.Se cumple "un atributo X entonces un atributo Y" si dado los valores de X puedo saber los valores de Y. Si yo conozco el valor del atributo X estoy seguro que puedo conocer el valor del atributo Y. Si el atributo X en una tupla vale 10, el atributo Y vale 20, entonces en cualquier tupla en donde el atributo X valga 10, el atributo Y va a valer 20.

Todos los atributos de una tabla dependen de la clave primaria, sabiendo la clave primaria podes saber el resto de los valores. PUEDE ocurrir que un atributo que no es clave primaria determine a la clave primaria SI Y SOLO SI ese atributo es CLAVE UNÍVOCA. (por ej tenes como clave primaria id y dsp como atributo clave univoca a DNI, sabiendo el DNI podes determinar y conocer su clave primaria, pero dsp el nombre del alumno no determina cual es su ID porque hay muchos con ese nombre). **En una tabla, los atributos que son clave primaria y univoca DETERMINAN al resto de los atributos (vas a poder conocer el resto de los atributos).**

Te tenes q preguntar "dado este atributo, puedo conocer el valor de este otro atributo?", por ej esto:

### Ejemplo 3

- Empl\_proyecto=(nro\_empl, nro\_proy, horasTrabajadas, nombre\_empleado, nombre\_proyecto)
  - (Nro\_empl, nro\_proy) → horasTrabajadas
  - Nro\_empl → nombre\_empleado
  - Nro\_proy → nombre\_proyecto

los 3 son dependencias funcionales pq (por ej el segundo) dado el nro de empleado puedo saber/determinar el nombre del empleado.

El de la clave primaria tmb, pq yo se q dada esa clave primaria (con valores para el nro de empleado y nro de proyecto, ej 23,54) puedo determinar las horas trabajadas para ese valor indicado, pq no se repite para ese conjunto.

### **Dependencia funcional completa: ( tambien son dependencias multievaluadas triviales (def mas abajo)).**

Cuando yo a partir de una dependencia funcional puedo obtener un subconjunto de la cual se siga cumpliendo dicha dependencia, entonces eso genera una dependencia funcional completa.

- Si A y B son atributos de una relación r, B depende funcionalmente de manera completa de A, si B depende de A pero de ningún subconjunto de A.
- En la transparencia anterior
  - (nro\_empl, nro\_proy)  nombre\_empleado
  - Nro\_empl → nombre\_empleado
  - Ambas funcionales, cual completa?
- (nro\_empl, nro\_proy) → nombre\_proyecto
- Nro\_proy → nombre\_proyecto
- Idem anterior

En el primer ejemplo la segunda seria completa. (El nombre del empleado depende de manera funcional y completa del nro de empleado, pero el nombre del empleado no siempre depende de manera funcional y completa del nro de empleado Y el nro de proyecto a la vez, por ende ese seria dependencia funcional parcial).

En el segundo ejemplo pasa lo mismo.

### **Dependencia funcional parcial:**

A entonces B es una dependencia funcional parcial si existe algún atributo que puede eliminarse de A y la dependencia continua verificándose. Básicamente si tenes una dependencia funcional de la cual le podés sacar un subconjunto y construir una dependencia completa, entonces la de arriba (la primera) va a ser dependencia parcial.

### **Dependencia funcional transitiva:**

Si A determina B y B determina C, entonces A determina C. En el video te explica un ejemplo donde se puede generar repetición de información.

### **Normalización:**

Es una técnica formal. Estudia las dependencias funcionales. Se revisan las tablas y que dichas tablas tengan exclusivamente dependencias funcionales completas para asegurarnos que no hay repetición innecesaria de datos.

**Primera forma normal:** Un modelo estará en 1NF si para toda relación r del modelo (tabla) cada uno de los atributos que la forman son si y solo si monovalentes. **Se aplica siempre**

**Segunda forma normal:** Un modelo está en 2NF si y solo si está en 1NF (por esto es un modelo incremental) y para toda relación r del mismo no existen dependencias parciales. Todas las dependencias deben ser completas. **Normalmente se aplican siempre**

**Tercera forma normal:** Un modelo está en 3NF si y solo si está en 2NF y para toda relación r del mismo (tabla) no existen dependencias transitivas. **Normalmente se aplican siempre.**

**Boyce Codd forma normal:** Un modelo está en BCNF si y solo si está en 3NF y para toda tabla el determinante (sería el X del X entonces Y) es una clave candidata o primaria.

**Cuarta forma normal:** Un modelo está en 4FN si y solo si está en BCNF y para toda relación r del mismo (tabla) sólo existen dependencia multivaluadas triviales.

**Dependencia multivaluadas trivial** es una dependencia que vos no la podes descomponer más y sigue valiendo. ej ( $A \rightarrow\!\!> C$  (trivial) y  $(A,B) \rightarrow\!\!> C$  (NO ES TRIVIAL PQ SE PUEDE DESCOMPONER). Solución a esto, tener dos tablas.

**Quinta forma normal:** Un modelo está en 5FN si está en 4FN y no existen relaciones (tablas) con dependencias de combinación. Esto ultimo significa que no se generen tuplas innecesarias (espurias, tipo q no tienen sentido los datos q tienen) a la hora de combinar tablas con una operacion de algebra relacional.

#### **Dependencias Multivaluadas:**

A multidetermina B cuando dado un valor para el atributo A yo puedo obtener múltiples valores para el atributo B. Ej una fecha de nacimiento puede multideterminar nombres de personas ya que en una fecha pudieron haber nacido muchas personas. Esto no tiene nada de malo, el problema es que cuando esa multideterminación se te va de las manos

#### **Clase teorica sql:**

Alter table modifica la tabla.

Con tipo text casi podes escribir de manera ilimitada y el varchar indica limitación de caracteres.

Unique index es clave univoca.

#### **DML:**

el where es opcional (sería el filtro)

Select \* te muestra todo el contenido de una tabla.

Select Distinct x, no repite una fila que ya apareció (no te repite info) El opuesto de este es el Select All, (el all esta por defecto) y te muestra todas las repeticiones (si las hay).

Concat(Rtrim(apellido) „ „ Rtrim(nombre)) te muestra la info así: Wagner,Luciano. (el Rtrim te borra los espacios en blanco ya q si tenes string de 40 y ocupas 4, te quedan espacios al pedo). (va en el select)

INNER JOIN es el producto natural.

Atributo as x, renombra el atributo en la columna que se te va a mostrar. Select atributo as x por ej.

Operador Like: No compara igualdad (que tengan el mismo tamaño) . Where nombre like “perez%” te compara con aquellos nombres que empiezan con perez y dsp si otra tiene otros caracteres no la tiene en cuenta.

Upper y lower (nombre) pone mayus y minusculas.

%casa% cualquier cadena que tenga casa en su interior.

“\_\_\_” cualquier cadena con 3 caracteres

"\_\_\_% cualquier cadena con al menos 3 caracteres.

'P\_rez%' por si tiene acentos.

Ordenar se hace con **ORDER BY**. (iria debajo del where)

Si no se le indica nada (en cuanto a ascendente y descendente) va a ser en orden ascendente por defecto, sino tenes que usar Desc. Cada atributo le podes indicar si es ascendente o descendente.

**Union:** agrupa las tuplas resultantes de dos subconsultas. Los atributos de ambas tablas deben tener el mismo dominio y deben tener la misma cantidad de atributos ambas tablas. (el primer atributo de una tabla con el primer atributo de la segunda tabla y asi sucesivamente se compara).

Funciones de agregación son funciones que operan sobre conjuntos por lo que jamas van a poder estar en un where, ya que este ultimo es sobre tuplas, por lo que son incompatibles. Son 5. Pueden aparecer en el select y en el having. Hacen algo relativo a lo que tiene definido como funcionalidad.

ej: select MAX(monocuota)

from deportes.

El count te cuenta filas. Generalmente se usa count (\*)

### Clase teorica 29-9:

create view le pone nombre a una consulta, es como guardar toda la consulta en una variable para que luego pueda ser referenciada nuevamente (por ej lo queres unir con otra consulta). Es una tabla temporaria.

ej:

```
CREATE VIEW consulta (
    SELECT a.idsocio, COUNT( * ) AS Cantidad
    FROM asociados a INNER JOIN practica p ON ( a.idsocio = p.idsocio )
    GROUP BY a.idsocio )

    SELECT a.nombre, c.cantidad
    FROM asociados a INNER JOIN consultas c ON (a.idsocio = c.idsocio )
```

Group by: permite agrupar un conjunto de tuplas por algun criterio.

Having: permite aplicar condiciones a los grupos.

La función de agregación puede ir en el having. El having solo se puede usar si hay group by, pq sino no te sirve, es una condicion de filtro para el group by.

**valores nulos:** No se pueden comparar contra valores existentes, es NADA. La unica forma de comparar es usar el **IS NOT NULL O IS NULL**.

subconsultas anidadas:

Pertenencia a conjuntos: IN. Solamente va en el where, lo que hace el in es comparar 2 cosas y da true y false. puedes usar **NOT in**. where x in (select ....) te dice básicamente que te quedas con una información de una tabla en particular “donde x pertenece al conjunto del select....”). o puedes poner ahí **not in** indicando que te quedas una información donde x no pertenece al conjunto (select...).

Comparación de conjuntos: > (**some**) (>= < = <>). y despues tenes el > **all** ((>= < = <>)..

**Cláusula exist:** Va en el where, devuelve verdadero si la subconsulta no es vacía (devuelve alguna tupla) y devuelve falso si la subconsulta es vacía. Admite poner un **not exist**.

**variantes del producto natural:**

#### **TOMA DE NOTAS DEL VIDEO DE EXPLICACION SOBRE SQL:**

**Distinct:** te elimina tuplas repetidas. Se aplica a campos del select.

## Ejemplo:

Alumno=(dni, nombre, apellido)

Select DISTINCT(nombre)

From alumno

Para filtrar una tupla usas where. Su condición debe ser verdadera para que te devuelva esa tupla, si es falso no devuelve ninguna tupla.

Operador	Significado	Ejemplo
=	es igual a	Select nombre From alumno Where (nombre="Luciana")
>	es mayor a	Select nombre From alumno Where (dni>24564321)
<	es menor a	Select nombre From alumno Where (dni<24564321)
>=	mayor o igual a	Select nombre From alumno Where (dni>=24564321)
<=	menor o igual a	Select nombre From alumno Where (dni<=24564321)
$\neq$	distinto a	Select nombre From alumno Where (dni<>24564321)
BETWEEN	entre (se incluyen extremos)	Select nombre From alumno Where (dni between 24564321 and 27456789)
LIKE	como	ejemplo próxima diapositiva

**Like:** Te permite trabajar con strings.

Puede usar:

En el primer ejemplo te devuelve todas las tuplas donde apellido tenga algo y termina con ez. Por ej: asdasdasdaez o sadadxsez. Podes tener tipo %ez% en el where tmb.

En el segundo ejemplo te devuelve todas las tuplas donde los primeros 2 caracteres (hay 2 guiones bajos) pueden ser cualquier cosa y termine con ciana. por ej: ticiana.

Podes hacer operaciones sobre el **Select** siempre y cuando corresponda con su dominio.

```
ej: SELECT apellido, dni -20000000
     FROM alumno
     LIMIT 0 , 30
```

**IS NULL** (su negacion **IS NOT NULL**): Verifica si un atributo contiene valor de NULL. Valor que se almacena por defecto si el usuario no define otro.

ej:

```
SELECT nombre, apellido  
FROM alumno  
WHERE (nombre IS NOT NULL)
```

Producto cartesiano:

**Producto Cartesiano (,)**: para realizar un producto cartesiano, basta con poner en la cláusula FROM dos o más tablas separadas por coma.

```
SELECT *  
FROM alumno,materia
```

---

```
SELECT a.nombre as 'Nombre alumno' ,m.nombre as 'Nombre materia'
```

**AS**: Renombre de atributos.

```
FROM alumno a,materia m
```

Alias definido para una tabla.

**LO PRIMERO QUE SE EJECUTA EN una consulta de SQL ES EL FROM.**

En el from pones Tabla x para **renombrar la tabla**. y con el **AS renombras atributos**

```
SELECT a.nombre AS 'Nombre alumno', m.nombre AS 'Nombre materia'  
FROM alumno a, materia m  
LIMIT 0 , 30
```

Mostrar :

30

fila(s) iniciando en la fila #

0

en modo

horizontal

+ Opciones

Nombre alumno	Nombre materia
Pablo	Diseño de bases de datos
Pablo	Fundamentos de organización de datos
Ricardo	Diseño de bases de datos
Ricardo	Fundamentos de organización de datos
Carlos	Diseño de bases de datos
Carlos	Fundamentos de organización de datos
Carlos	Diseño de bases de datos
Carlos	Fundamentos de organización de datos
Luciana	Diseño de bases de datos
Luciana	Fundamentos de organización de datos
Luciana	Diseño de bases de datos
Luciana	Fundamentos de organización de datos

#### Producto natural: Inner join.

.Usas el inner join para indicarle al SQL por qué atributos queres que compare. Sino esta el **natural join** que se fija de manera automatica los atributos que tengan mismo nombre en ambas tablas y si tienen iguales valores en todas esas, entonces se queda con la tupla.

**INNER JOIN:** producto natural, reúne las tuplas de las relaciones que tienen sentido. El producto natural se realiza en la cláusula FROM indicando las tablas involucradas en dicho producto, y luego de la sentencia ON la condición que debe cumplirse.

**SELECT** a.nombre,a.apellido, e.nota  
FROM alumno a INNER JOIN examen e ON  
(a.dni=e.dni)   **¿Otra forma? ¿Cuál es mejor?**

**NATURAL JOIN:** análogo al producto natural de AR, trabaja por equicombinación.

Lo de que dice de realizar otra manera, es hacer en el from un producto cartesiano y despues en el where haces el filtrado.

**otro ejemplo:**

```
SELECT a.nombre,a.apellido,m.nombre, e.nota  
FROM alumno a INNER JOIN examen e ON (a.dni=e.dni) INNER JOIN materia m ON (e.codigo=m.codigo)
```

Examen tiene clave foranea con alumno (por el dni) y con la materia (con el codigo).

**Forma equivalente:**

```
SELECT a.nombre,a.apellido,m.nombre, e.nota  
FROM alumno a, examen e, materia m  
WHERE (a.dni=e.dni)AND(e.codigo=m.codigo)
```



**Left join:**

```
SELECT *  
FROM alumno a LEFT JOIN examen e ON (a.dni=e.dni)
```

**Este ejemplo te serviría para mostrar los alumnos que rindieron o no algún examen.**  
Lo que hace es que SIEMPRE en una tupla vas a tener la información del alumno (de la

izquierda) y vas a tener la informacion de la derecha si cumple que los atributos entre ambos que tengan igual nombre tengan igual contenido, caso contrario lo de la derecha te lo pone en null.

+ Opciones							
dni	nombre	apellido	codigo_examen	dni	codigo	nota	
24564321	Pablo	Perez		1 24564321	1	2	
24564321	Pablo	Perez		2 24564321	1	6	
27456789	Ricardo	Suarez		3 27456789	2	10	
28670845	Carlos	Gandolfi	NULL	NULL	NULL	NULL	
30678987	Carlos	Gonzalez		4 30678987	1	5	
36987654	Luciana	Amendola		NULL	NULL	NULL	
38987543	Luciana	Iun		NULL	NULL	NULL	

Right join es lo mismo pero al revés.

.Union (igual que antes): no retorna tuplas duplicadas.

. Union all: Es lo mismo que union pero retorna tuplas duplicadas.

**LAS CONSULTAS A UNIR DEBEN TENER ESQUEMAS COMPATIBLES.**

```
SELECT nombre
FROM alumno
WHERE (dni > 25000000)
UNION
SELECT nombre
FROM materia
```

EJ:

La diferencia se realiza con el except: Se debe tener esquemas compatibles. Es la misma lógica que en A.R

ej:

```
SELECT dni  
FROM alumno  
WHERE (dni > 25000000 )  
EXCEPT  
(SELECT dni  
FROM examen  
)
```

esquemas

**La intersección:** Se realiza con INTERSECT. Se debe tener esquemas compatibles.

**ORDER BY:** Ordenar el resultado de una consulta determinada. Ordenar las tuplas resultantes por el atributo que se indique. Por defecto el orden es ascendente. Para que sea descendente, al final tenes que escribir **DESC**.

```
SELECT nombre, apellido,dni  
FROM alumno  
WHERE (dni>23000000) and (nombre='Luciana')  
ORDER BY apellido, dni DESC
```

Aca te ordena por apellido de manera ascendente, y cuando haya una tupla con igual apellido que otra, anda al segundo criterio de ordenación la cual es por DNI de manera descendente y asi sucesivamente.

**FUNCIONES DE AGREGACIÓN: VAN EN EL SELECT.** Se recibe un conjunto de tuplas de entrada produciendo un único valor de salida. No siempre hay q usar group by para estos casos, si lo tenes q usar group by hay q combinarlo con esas funciones, pero estas funciones no dependen del group by. ej punto 2.5 de la practica.

Se utilizan en el select y operan sobre un conjunto de tuplas de entrada produciendo un único valor de salida.

- ◆ **AVG**: promedio del atributo indicado para todas las tuplas del conjunto.
- ◆ **COUNT**: cantidad de tuplas involucradas en el conjunto de entrada.
- ◆ **MAX**: valor más grande dentro del conjunto de tuplas para el atributo indicado.
- ◆ **MIN**: valor más pequeño dentro del conjunto de tuplas para el atributo indicado.
- ◆ **SUM**: suma del valor del atributo indicado para todas las tuplas del conjunto.

ej clave e importante del max:

► Ejemplo 22: mostrar el deporte que cobra la cuota mas alta  
**SELECT MAX (montocuota) FROM deportes**      **SELECT nombre, MAX(montocuota) FROM deportes**      **SELECT nombre FROM deportes WHERE montocuota = (SELECT MAX(montocuota) FROM deportes )**

Los que estan en rojo no andan bien pq podes tener 3 maximos, pq esos 3 tienen valores iguales y son los mas altos.

**SIEMPRE HAY QUE AGRUPAR POR LOS CAMPOS QUE APARECEN EN EL SELECT y POR LA CLAVE PRIMARIA DIJO EL PROFE, exceptuando funciones de agregacion como AVG, MAX,ETC.**

ej:

❖ **GROUP BY**: agrupa las tuplas de una consulta por algún criterio con el objetivo de aplicar alguna función de agregación.

```
SELECT nombre, apellido, AVG(notas) as promedio  
FROM alumno a INNER JOIN examen e ON  
    (a.dni=e.dni)  
GROUP BY e.dni, nombre, apellido
```

Qué información se puede mostrar cuando se realizó un agrupamiento?

Porque es importante agrupar además por PK?

Agrupas un conjunto de tuplas, en este caso agrupas las tuplas correspondientes a un mismo dni (donde a.dni = e.dni) y ese conjunto de tuplas va a hacer la entrada a esa función de agregación la cual generará un único valor de salida.

Pones e.dni en el group by pq queres sacar promedio, si vos pones a.dni no se repite ese dni, aparece 1 sola vez, pq es clave primaria, no te sirve.

Pones en el group by el dni, pq sabes q hace referencia a 1 sola persona, si vos pones nombre y apellido solamente, suponete que tenes 2 personas distintas llamadas Luciano Wagner, cuando agrupes te lo va a tomar como una única persona y en realidad son personas distintas.

#### EL COUNT, A TENER EN CUENTA:

Dentro de la función de agregación COUNT en SQL, generalmente se coloca el nombre de la columna que deseas contar. La función COUNT cuenta el número de filas que contienen valores no nulos en la columna especificada. Aquí tienes un ejemplo de cómo se utiliza:

```
```sql  
SELECT COUNT(NombreColumna) FROM NombreTabla;  
```
```

En este ejemplo, "NombreColumna" es el nombre de la columna que deseas contar, y "NombreTabla" es el nombre de la tabla en la que se encuentra esa columna. La función COUNT cuenta el número de filas en la tabla donde la columna "NombreColumna" no contiene valores nulos.

Si deseas contar todas las filas de la tabla, independientemente de los valores nulos, simplemente puedes usar `COUNT(\*)` en lugar de una columna específica:

```
```sql
```

```
SELECT COUNT(*) FROM NombreTabla;
```

...

Esto contará todas las filas de la tabla "NombreTabla".

**HAVING:** Se usa con el group by **SOLAMENTE**. Se usa para filtrar/mostrar grupos que cumplan una determinada condición. Sería el where pero con grupos.

La cláusula **HAVING** se usa con la cláusula GROUP BY para restringir los grupos que aparecen en la tabla de resultados mediante alguna condición que deben cumplir los grupos

```
SELECT nombre, apellido, AVG(nota) as promedio  
FROM alumno a INNER JOIN examen e ON  
(a.dni=e.dni)  
GROUP BY e.dni, nombre, apellido  
HAVING promedio > 5
```

**Subconsultas:** Comparamas valores de la consulta principal con valores devueltos de una subconsulta.

Consiste en ubicar una consulta SQL dentro de otra. SQL define operadores de comparación para subconsultas:

- ❖ **= (igualdad):** cuando una subconsulta retorna un único resultado, es posible compararlo contra un valor simple.
- ❖ **IN (pertenencia):** comprueba si un elemento es parte o no de un conjunto. Negación (**NOT IN**).
- ❖ **=SOME:** igual a alguno.
- ❖ **>ALL:** mayor que todos.
- ❖ **<=SOME:** menor o igual que alguno

El **in** determina si un valor está incluido en un conjunto de valores que te devuelve la subconsulta.

=some es si un valor de la consulta principal es igual a alguno de los valores devueltos de la subconsulta.

>all, es si un valor de la consulta principal es mayor que todos los valores devueltos de la subconsulta.

<=some ....

ej:

```
SELECT nombre, apellido  
FROM alumno  
WHERE dni IN (SELECT dni FROM examen  
WHERE nota = 10)
```

Te va a devolver nombre y apellido de la tabla alumno donde el dni este incluido en la subconsulta indicada. Basicamente te devuelve todos los nombres y apellidos de los alumnos que se sacaron 10. **PRIMERO SE REALIZA LA SUBCONSULTA.**

ej:

- ▶ Subconsultas anidadas
- ▶ Comparación de Conjuntos
  - ▶ > **some** ( $<$ ,  $=$ ,  $\geq$ ,  $\leq$ ,  $\neq$ )
  - ▶ Ejemplo 31: mostrar todos los deportes, menos el mas económico

```
SELECT nombre
FROM deportes
WHERE montocuota > SOME ( SELECT montocuota
FROM deportes )
```

- ▶ 00:00,00
- ▶ > **all** ( $<$ ,  $=$ ,  $\geq$ ,  $\leq$ ,  $\neq$ )
  - ▶ Ejemplo 32: presentar el deporte mas oneroso

```
SELECT nombre
FROM deportes
WHERE montocuota >= ALL ( SELECT montocuota
FROM deportes )
```

DBD - CLASE 6

**Clausula exist** (se usa para los casos en donde te pide “mostrar TODOS los deportes que practica un asociado por ej): Te devuelve verdadero si la subconsulta devolvió al menos un valor. Caso contrario devuelve falso. Negacion : **NOT EXIST**.

Permite comprobar si una subconsulta generó o no alguna tupla como respuesta. El resultado de la cláusula **EXIST** es verdadero si la subconsulta tiene al menos una tupla, y falso en caso contrario.  
Negación (**NOT EXIST**)

```
SELECT nombre, apellido
FROM alumno a
WHERE EXISTS (SELECT * FROM examen e WHERE
a.dni=e.dni and e.nota<=4 )
```

Condición de la consulta principal

Si es verdadero, se muestra el nombre y apellido del alumno, CASO CONTRARIO NO SE MUESTRA ESE ALUMNO. Muestra los alumnos que se sacaron nota  $\leq$  a 4.

## INSERTAR, ELIMINAR Y MODIFICAR UNA TUPLA:

- ❖ **INSERT INTO**: agrega tuplas a una tabla.

```
INSERT INTO alumno (dni, nombre, apellido) VALUES (22670845,  
Juan,'Del Piero');
```

- ❖ **DELETE FROM**: borra una tupla o un conjunto de tuplas de una tabla.

```
DELETE FROM alumno WHERE dni=28670845;
```

- ❖ **UPDATE ... SET**: modifica el contenido de uno o varios atributos de una tabla.

```
UPDATE alumno SET nombre='Luciana' WHERE dni=24564321
```

En el update pones que queres modificar el atributo nombre a Luciana donde el dni sea 24.... Si no indicas el where, se te cambia todos los nombres por LUCIANA.

En el notexist va siempre en el select el asterisco. No te importa q devuelva, pq te devuelve verdadero o falso.