Diseño de Bases de Datos

Docentes - JTPs

- Sobrado, Ariel
- Cipollone, Juan Manuel
- Nucilli, Emanuel
- Torres, Juan Ignacio

Turnos

- Tarde – martes 18 a 21 hs. - aula 5
- Mañana – martes de 11 a 14. - aula 5

Selección de turno

https://portalacademico.com.ar/

Consultas administrativas dirigirse al JTP de su turno.

Se tomará asistencia de forma conceptual.

Para cambios de turno **sin excepción** certificado laboral indicando horario de trabajo y referencias para contactar.
Se aceptarán cambios de turno hasta la semana de 02/09.

Temas y materiales

- asignaturas.info.unlp.edu.ar
- Modelo de entidad - relación
- Pasaje al modelo lógico y físico
- Álgebra relacional y SQL

Exámenes

- **Tres exámenes donde se evalúan los temas vistos y en cada uno se puede aprobar por tema.**
- **Para aprobar la cursada se deben aprobar los tres temas.**

Primer examen práctico 11/11/2025 18:00
Segundo examen práctico 02/12/2025 18:00
Tercer examen práctico 16/12/2025 18:00

¿PREGUNTAS?

Diseño de Bases de Datos

Esquema de la explicación

- Definición
- Simbología
- Ejercicio para resolver en clase
  - Identificar Entidades – Relaciones más evidentes.
  - Identificar Entidades – Relaciones menos evidentes.
  - Identificar relaciones
  - Definir atributos.
  - Atributos compuestos y polivalentes
  - Establecer cardinalidades
  - Determinar identificadores

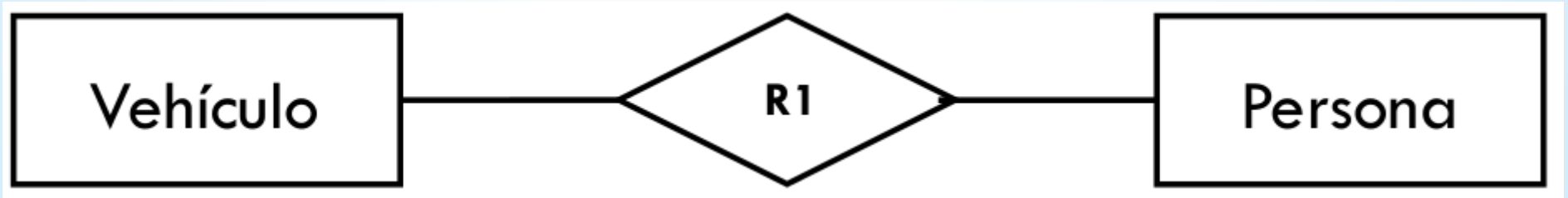
Diseño Conceptual - Definición

El diseño conceptual parte de la especificación de requerimientos y su resultado es el esquema conceptual de la base de datos. Un esquema conceptual es una descripción de alto nivel de la estructura de la base de datos. El propósito del diseño conceptual es describir el contenido de información de la base de datos. Herramienta utilizada: Modelo Entidad-Interrelaciones (ER).

Diseño Conceptual – MER – Simbología

Entidad: Una entidad representa un elemento u objeto del mundo real con identidad.

Relación: Las relaciones representan agregaciones entre dos (binaria) o más entidades. Describen las dependencias o asociaciones entre dichas entidades.



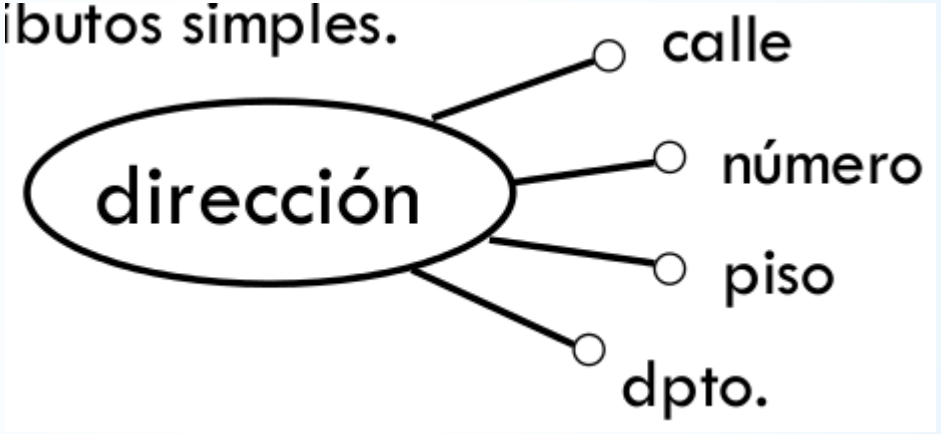
Diseño Conceptual – MER – Simbología

Relación Recursiva: Relación que une dos entidades particulares del mismo conjunto.

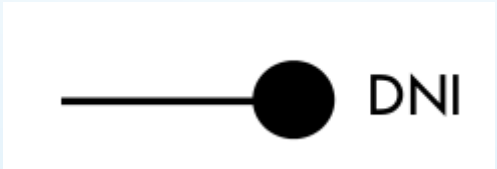
Atributo: Un atributo representa una propiedad básica de una entidad o relación. Un atributo es el equivalente a un campo de un registro.

Diseño Conceptual – MER – Simbología

Atributo Compuesto: Un atributo compuesto representa a un atributo generado a partir de la combinación de varios atributos simples.

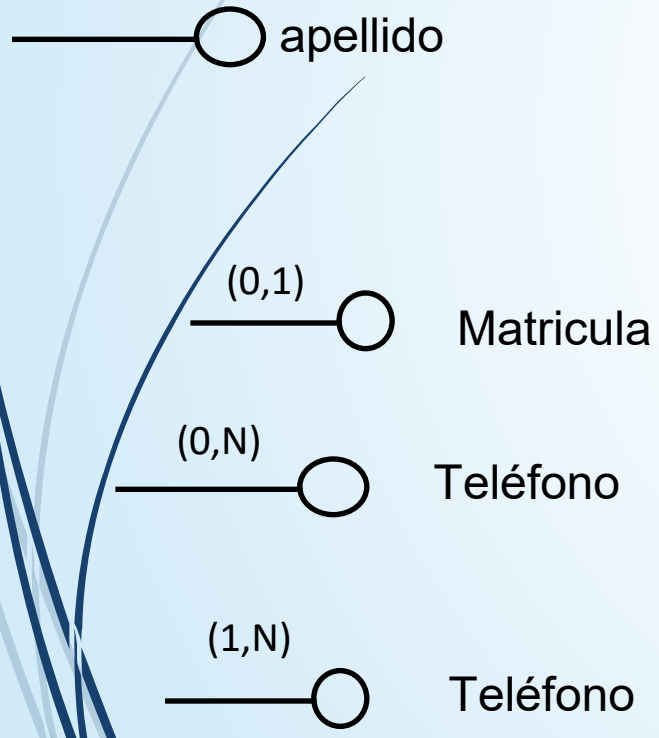


Identificador: Un identificador es un atributo o un conjunto de atributos que permite reconocer o distinguir a una entidad de manera unívoca dentro del conjunto de entidades.



Diseño Conceptual – MER – Simbología

Cardinalidad en los Atributos: Los atributos, tienen asociado el concepto de cardinalidad. Cuando se define un atributo se debe indicar si es o no obligatorio y si puede tomar más de un valor (polivalente).



Cardinalidad (1,1): Monovalente obligatorio. La cardinalidad existe y está presente, pero solamente en este caso no se indica en forma explícita.

Cardinalidad (0,1): Monovalente no obligatorio.

Cardinalidad (0,N): Polivalente no obligatorio.

Cardinalidad (1,N): Polivalente obligatorio.

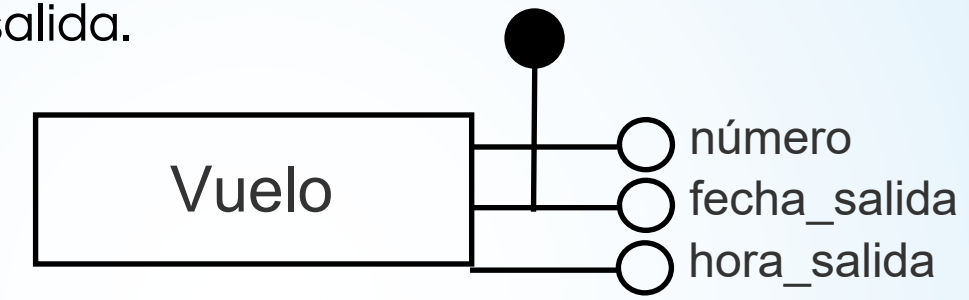
Diseño Conceptual – MER – Simbología

Cardinalidad en las relaciones: Es el nivel de correspondencia entre las entidades que se relacionan. Se debe definir el nivel mínimo de correspondencia (cardinalidad mínima) y el nivel máximo de correspondencia (cardinalidad máxima).

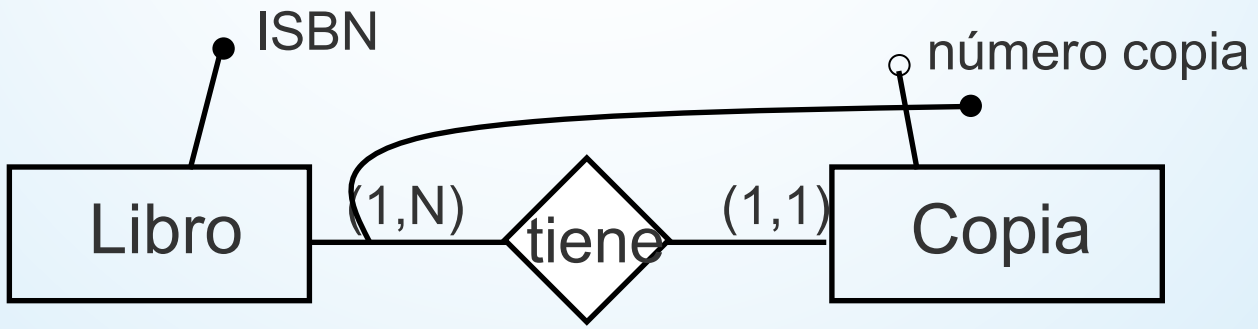
Esto muestra que un alumno debe cursar al menos una materia (obligatoriamente), pero puede cursar varias. Además, una materia puede no ser cursada (opcional) por ningún alumno o ser cursada por varios.

Diseño Conceptual – MER – Simbología

Identificador Compuesto: Identificador conformado por más de un atributo. El número de vuelo de una Aerolínea puede repetirse para diferentes fechas de salida.

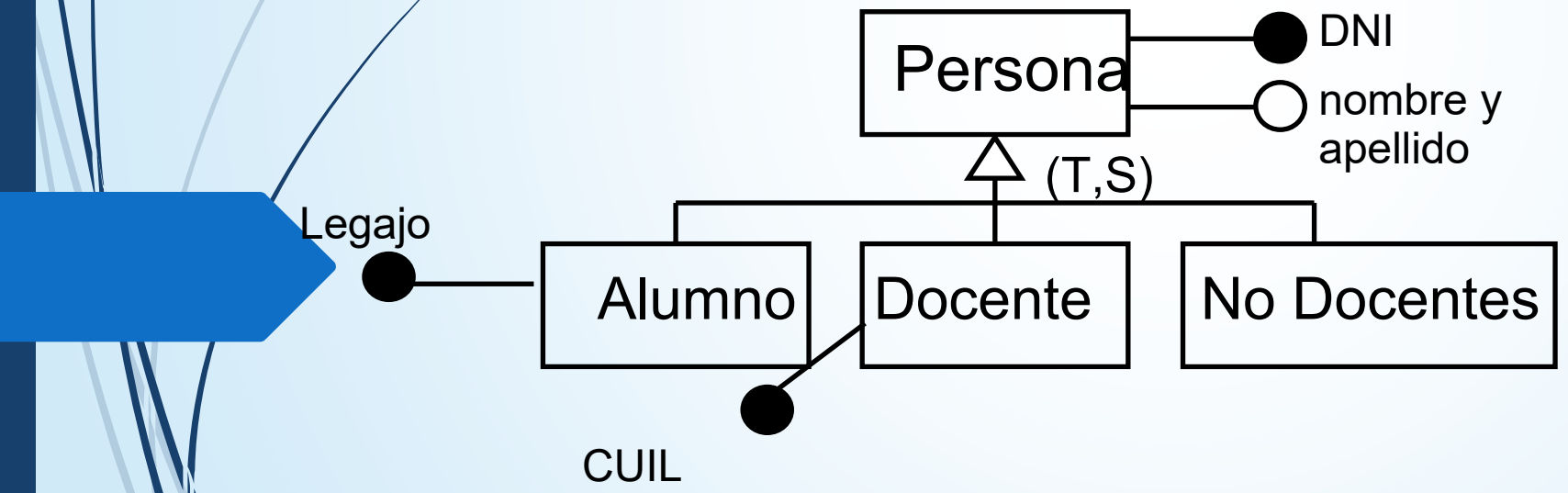


Identificador Externo: Identificador conformado por atributos que pertenecen a otra entidad. El número de copia puede repetirse para diferentes ISBN.



Diseño Conceptual – MER – Simbología

Jerarquías de generalización: Permiten extraer propiedades comunes de varias entidades, y generar con ellas una super-entidad que las aglutine. Así, las características compartidas son expresadas una única vez en el modelo, y los rasgos específicos de cada entidad quedan definidos en su sub-entidad.



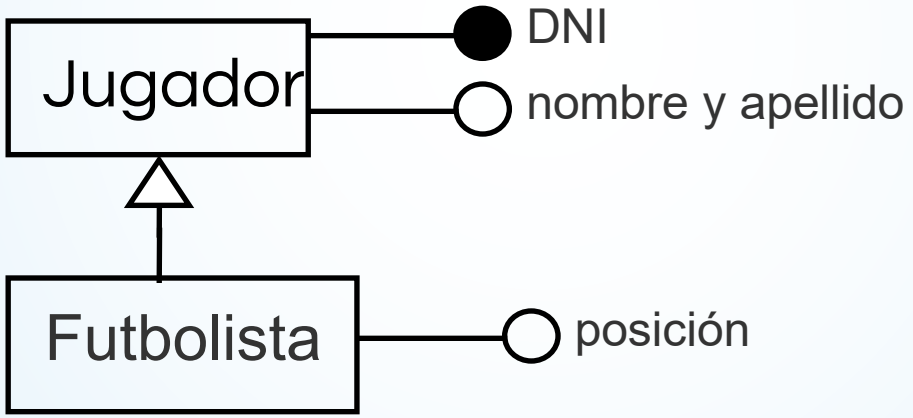
Coberturas posibles:

(T,E) (T,S)

(P,E) (P,S)

Diseño Conceptual – MER – Simbología

Subconjuntos: Caso especial de las jerarquías de generalización, donde se tiene una generalización de la que se desprende solamente una especialización. No es necesario indicar la cobertura para los subconjuntos.



Ejercicio Integrador

Se desea modelar la información referente a los alumnos de la Facultad de Informática. De los alumnos se conoce su DNI, Legajo, nombre y apellido, dirección detallada y teléfono/s.

Además se conoce información sobre las materias que se dictan en la facultad. De las materias se conoce código de materia (único), nombre y descripción.

Es importante representar las cursadas de cada materia. De cada cursada se sabe el año en que se dicta y a que materia corresponde. Una materia se cursa una única vez por año. Un alumno se puede inscribir a una o muchas materias.

Además es necesario modelar a los empleados de la facultad. De los empleados se conoce DNI, nombre, apellido y legajo. De los empleados docentes además se conoce el título (puede no tener título o tener más de uno) y las materias que dicta. El docente puede rotar de materia, por lo que es necesario representar el historial de materias por las que pasó. Por otro lado, de los empleados no docentes es necesario representar cuit y antigüedad.

Entidades más relevantes

Alumno-Materia-Empleado

Entidades menos relevantes

NoDocente-Docente-Cursada

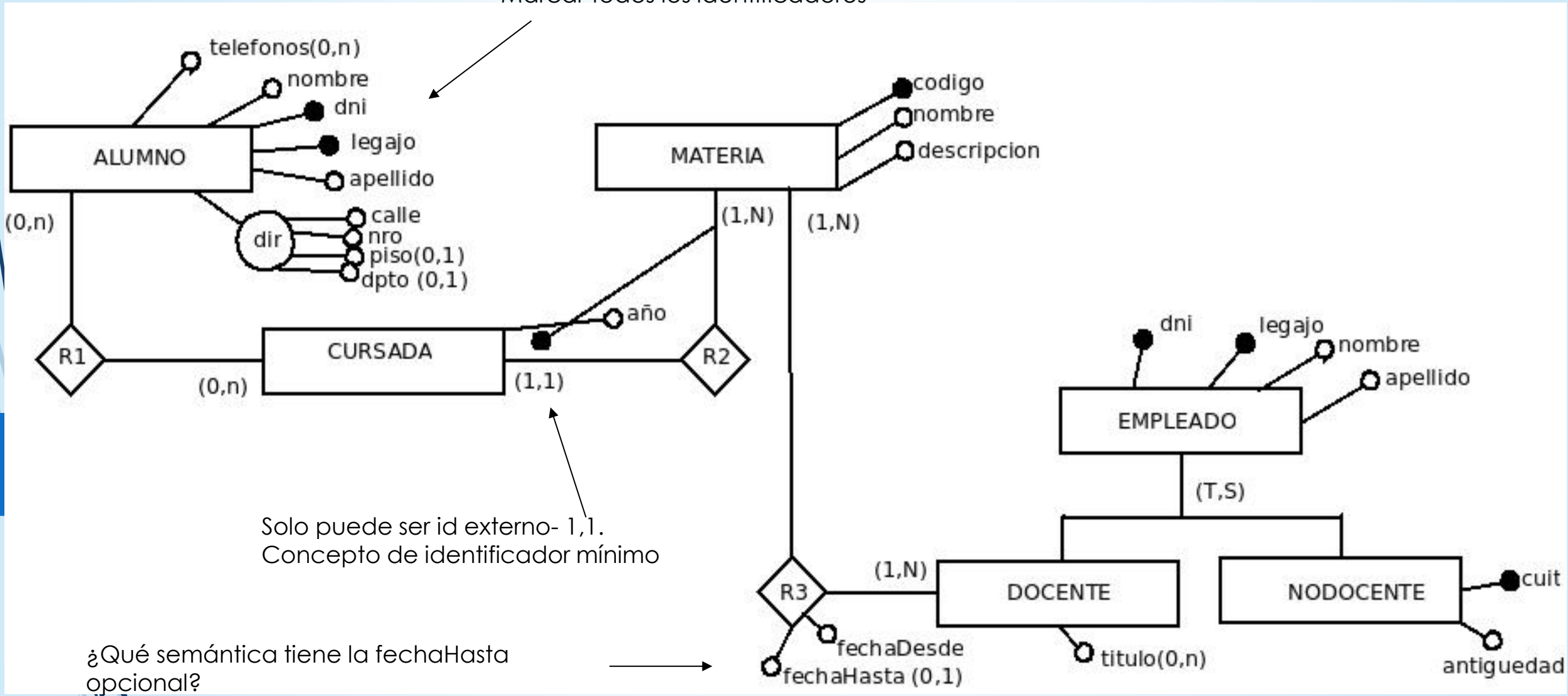
Relaciones

Cursada-Alumno/Cursada-Materia/ Docente-Materia

Completamos el modelo...

Ejercicio integrador

Marcar todos los identificadores



Solo puede ser id externo- 1,1.
Concepto de identificador mínimo

¿Qué semántica tiene la fechaHasta opcional?

Tarea para el hogar: Pensar diferentes formas de modelar el historial

¿PREGUNTAS?

Diseño de Bases de Datos

Modelado Lógico - Introducción

El propósito de la generación de un modelo ER Lógico es convertir el esquema conceptual en un modelo más cercano a la representación entendible por el SGBD.

Recordemos que el diseño conceptual busca representar, de la forma más clara posible, las necesidades del usuario. Una vez cumplido este paso, el diseño lógico busca representar un esquema equivalente, que sea más eficiente para su utilización.

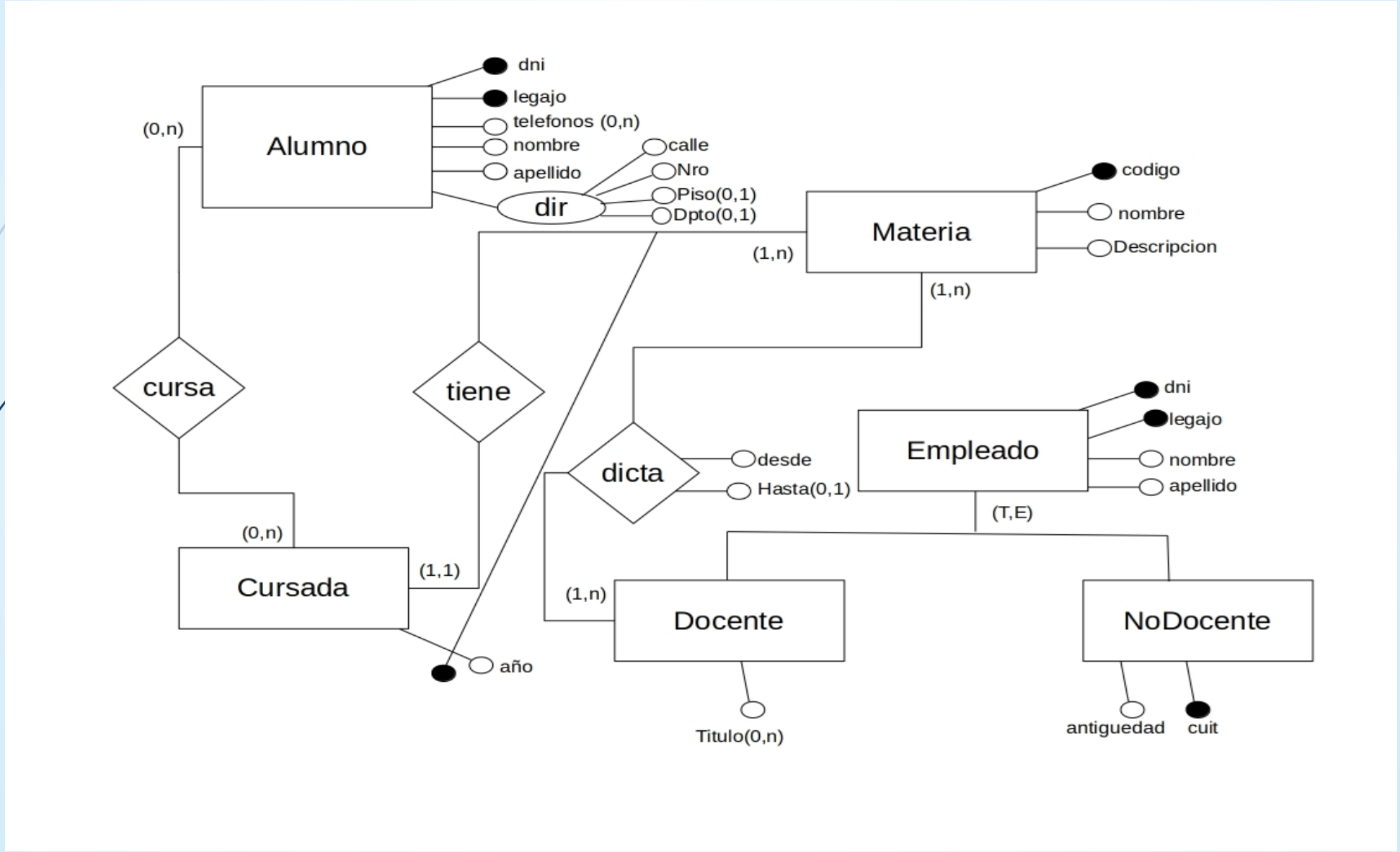
Decisiones sobre el Diseño Lógico

Las decisiones sobre el diseño lógico están vinculadas, básicamente, con cuestiones generales de rendimiento y con un conjunto de reglas que actúan sobre características del esquema conceptual que no están presentes en los SGBD relacionales.

Decisiones sobre el Diseño Lógico

- Resolver las Jerarquías
- Resolver Atributos Compuestos
- Resolver Atributos Polivalentes

Modelo Conceptual

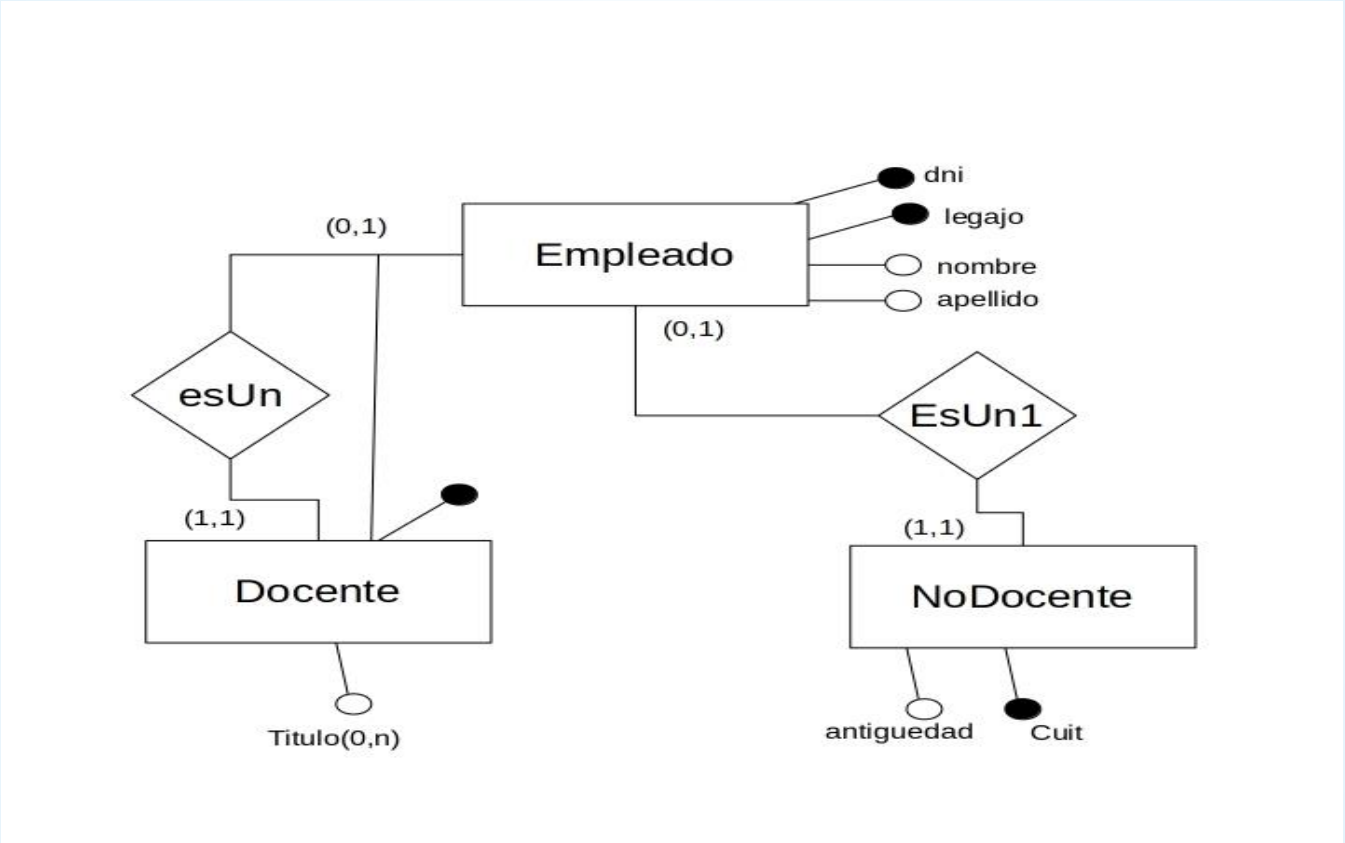


Resolver Jerarquías

- **Total Exclusiva (T, E):** Tres posibilidades, dejar todo, dejar sólo los hijos o dejar sólo al padre.
- **Total Superpuesta (T, S):** Dos posibilidades, dejar todo o dejar sólo al padre. No se puede eliminar al padre.

- **Parcial Exclusiva (P, E):** Dos posibilidades, dejar todo o dejar sólo al padre. No se puede eliminar al padre.
- **Parcial Superpuesta (P, S):** Dos posibilidades, dejar todo o dejar sólo al padre. No se puede eliminar al padre.

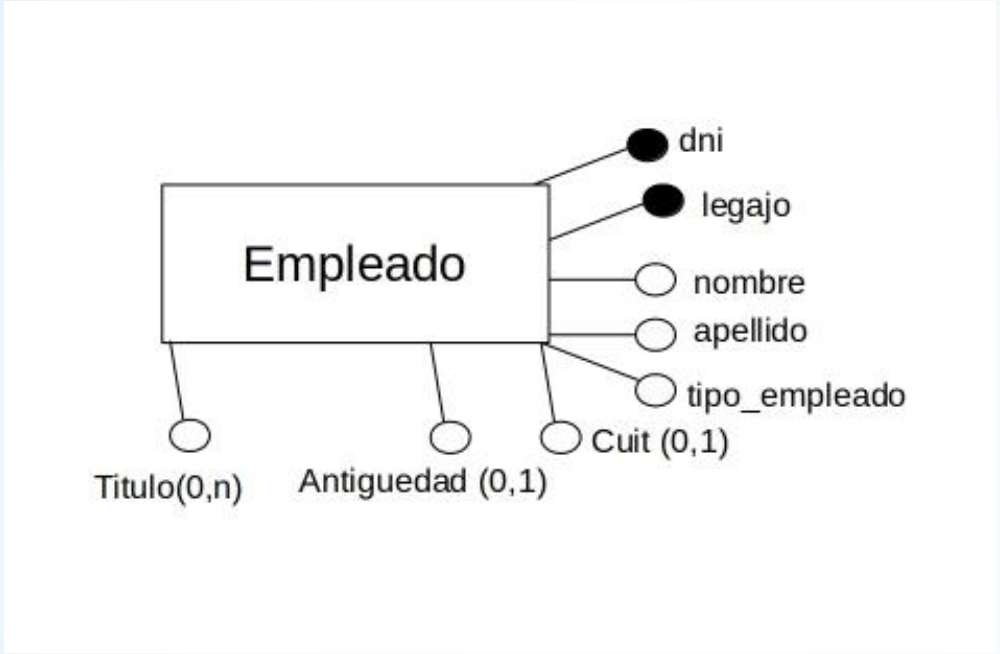
Resuelvo Jerarquía (T,E) - Primera opción - **dejar todas las entidades**



-Si las entidades hijas no tienen identificador debo bajarlo desde el padre. Caso contrario es opcional - NoDocente puedo no bajarlo, pero si lo bajo no debo cruzarlo con CUIT

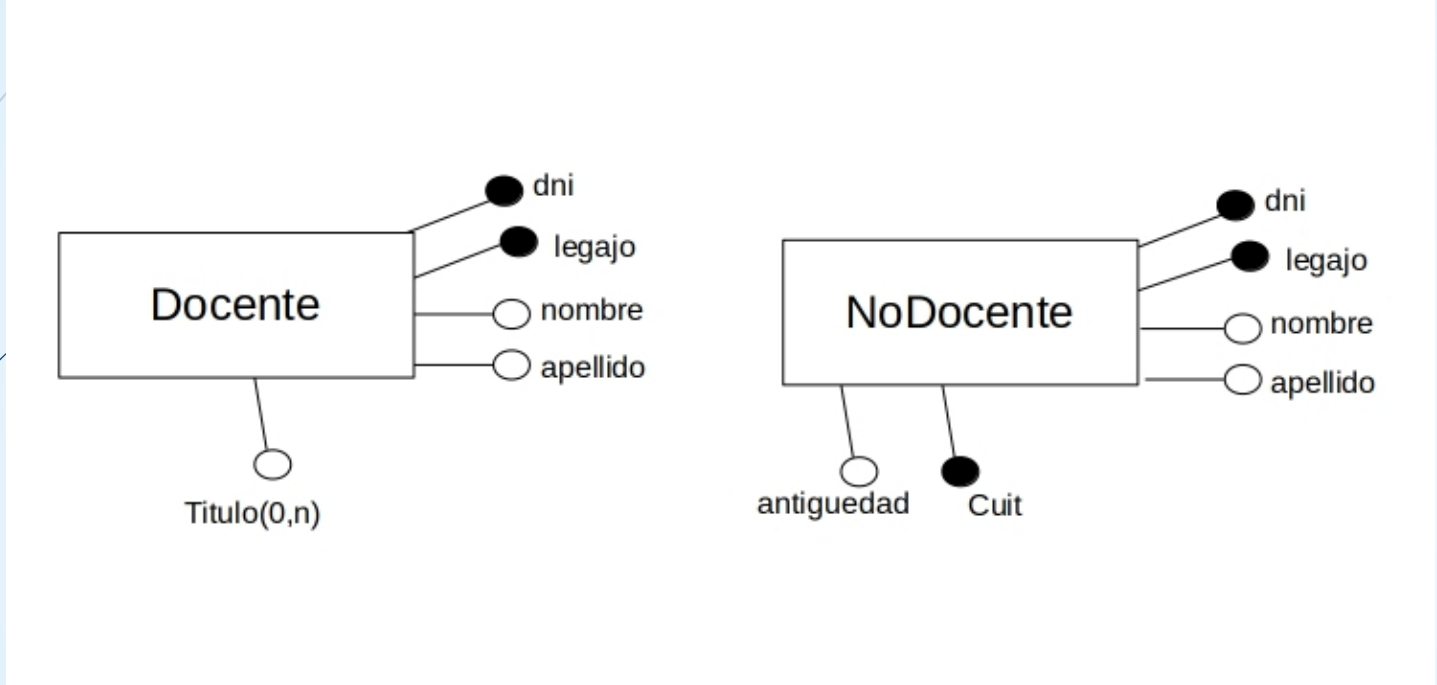
-(Se dibuja el identificador externo tomándolo desde la línea de la relación no desde la entidad)

Resuelvo Jerarquía (T,E) - Segunda opción - **dejar solo al padre**



- Todos los atributos de los hijos pasan al padre.
- Deben pasar como no obligatorios. Idem las relaciones en los hijos pasan como relaciones opcionales (mínima 0).
- Si en el hijo era un atributo identificador, debe dejar de serlo. (Nunca un identificador puede ser opcional).
- Si bien puede deducirse es una buena opción agregar un atributo que identifique qué tipo de empleado es (tipo_empleado).

Resuelvo Jerarquía (T,E) - Tercer opción - Dejar solo a los hijos



- Se deben bajar los atributos del padre a cada uno de los hijos.
- También se deben bajar las relaciones del padre (si hubiera).

Resolver Atributos Compuestos

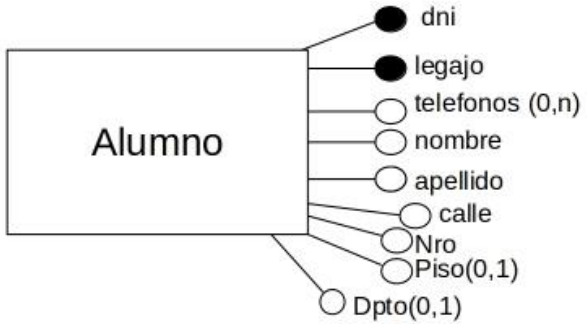
Las dos formas más utilizadas para eliminar los atributos compuestos son:

- **Considerar sólo los atributos individuales**
- **Considerar todo en un sólo atributo**

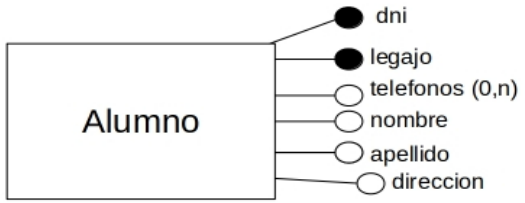
¿CUÁL ES LA MEJOR OPCIÓN? Ventajas y Desventajas

Resolver Atributos Compuestos

- Considerar sólo los atributos individuales

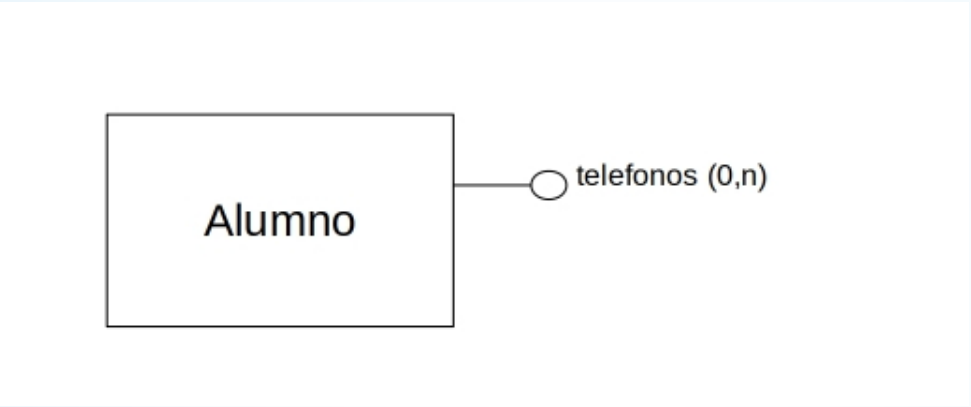


- Considerar todo en un solo atributo

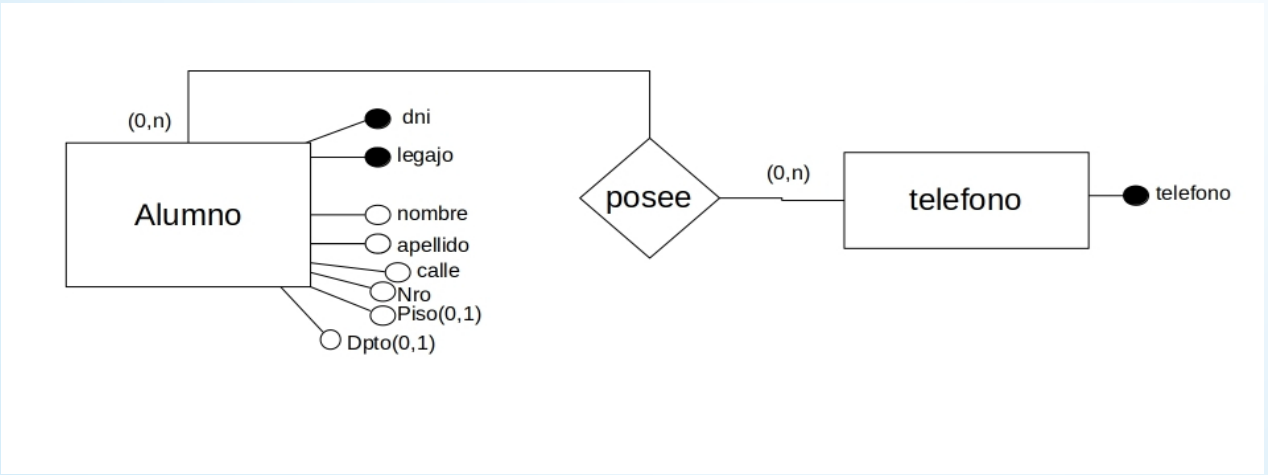


Resolver Atributos Polivalentes

Para resolver los atributos polivalentes se debe agregar una entidad y una interrelación.



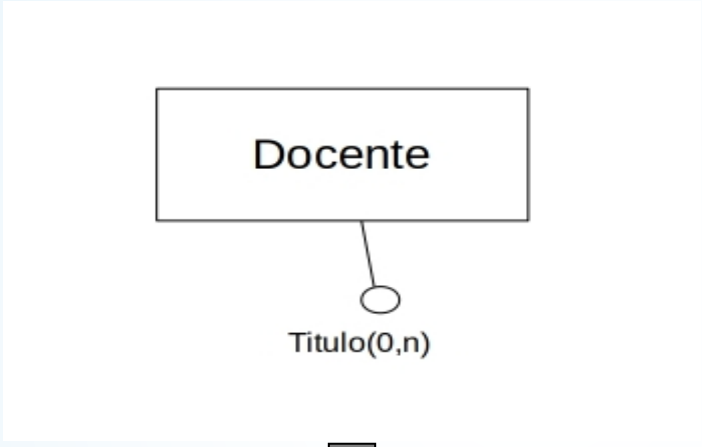
Solución 1: pasar como entidad de tipos



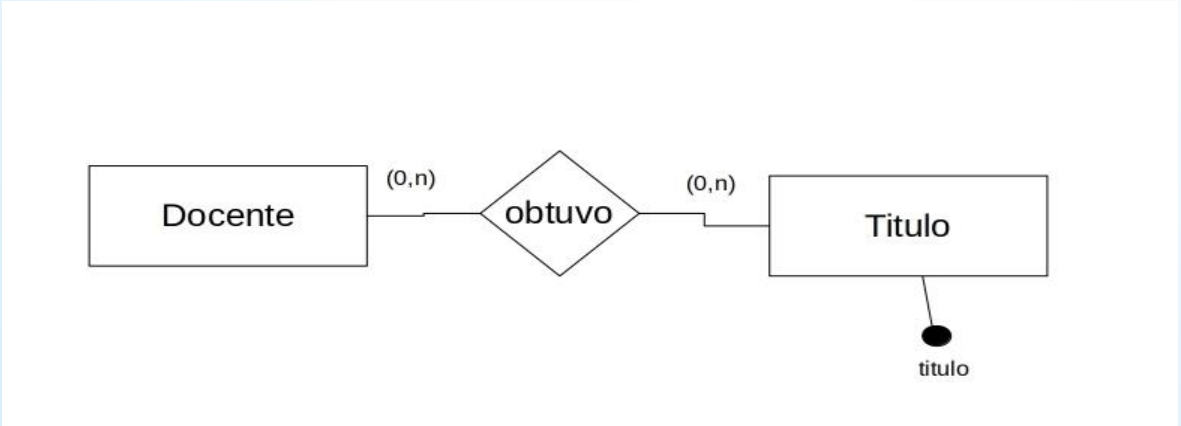
Observar cardinalidades

Resolver Atributos Polivalentes

Para resolver los atributos polivalentes se debe agregar una entidad y una interrelación.



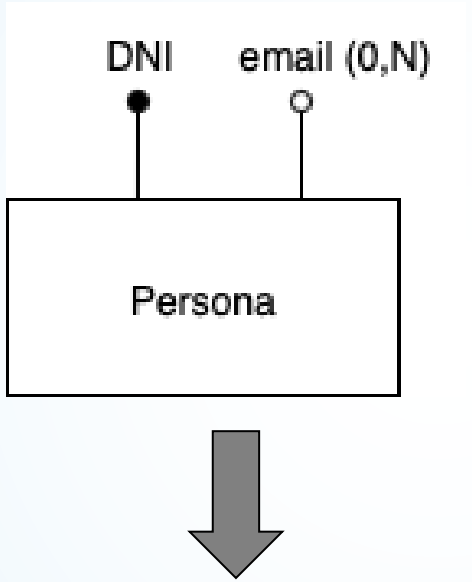
Solución 1: pasar como entidad de tipos



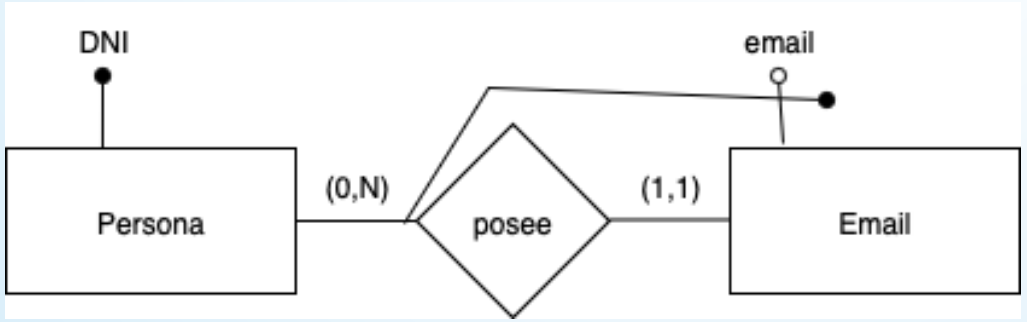
Observar cardinalidades

Resolver Atributos Polivalentes

Para resolver los atributos polivalentes se debe agregar una entidad y una interrelación.

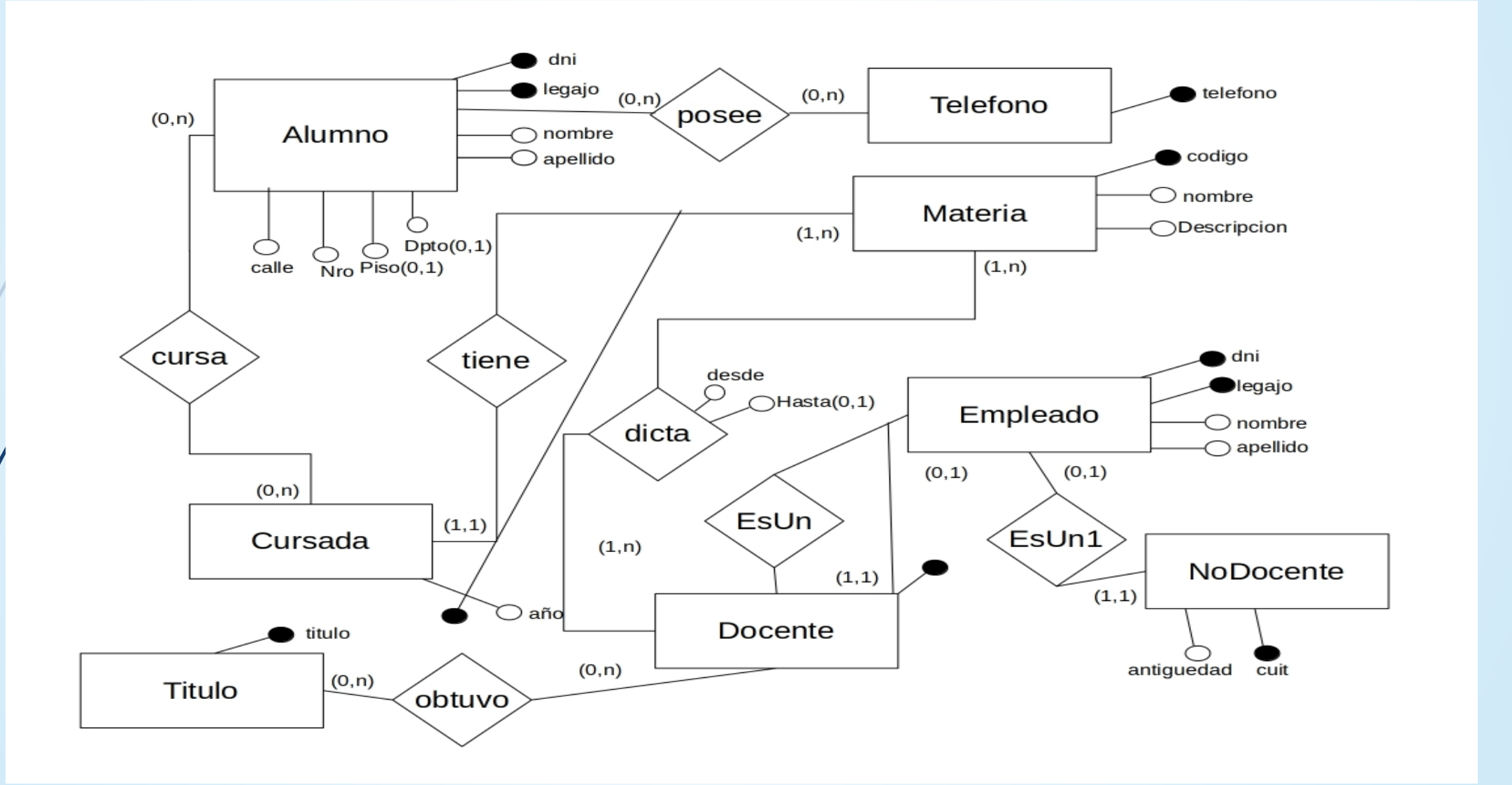


Solución 2: pasar como entidad débil



Observar cardinalidades

Modelo Lógico Final



Errores comunes al pasar al modelo lógico

1. No diferenciar correctamente entre atributos opcionales y polivalentes.
2. Convertir jerarquías sin considerar el contexto.
3. Agregar identificadores innecesarios.
4. Dejar entidades sin identificar.
5. Modelar incorrectamente los atributos polivalentes.

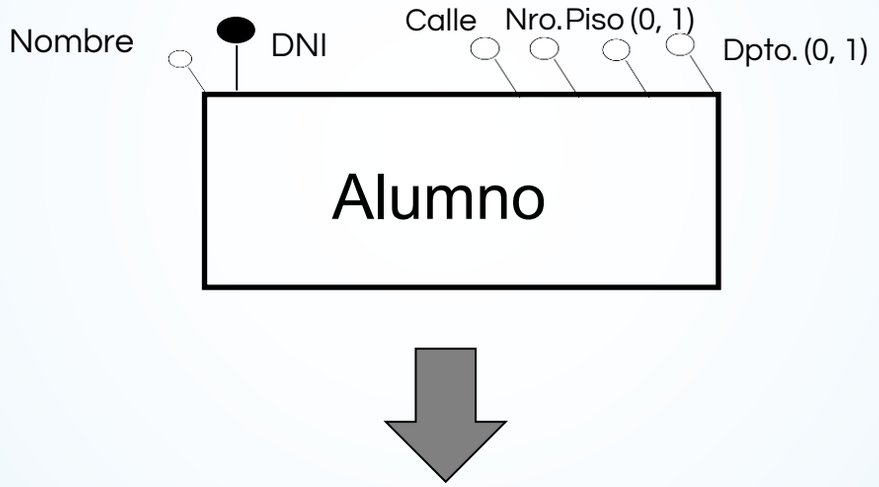
Diseño de Bases de Datos

Introducción

El modelo físico (relacional) representa a una BD como una colección de archivos denominados tablas. Cada tabla se denomina relación y está integrada por filas y columnas. Cada fila se denomina tupla y cada columna representa un atributo.

Conversión de entidades

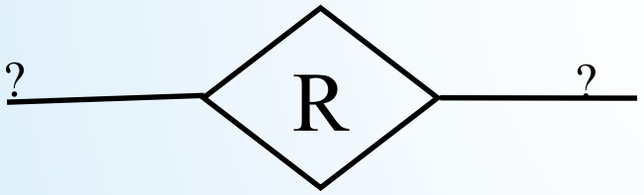
➔Cada entidad se transforma en una tabla.



Alumno= (dni, nombre, calle, *nro, piso?, dpto?)*

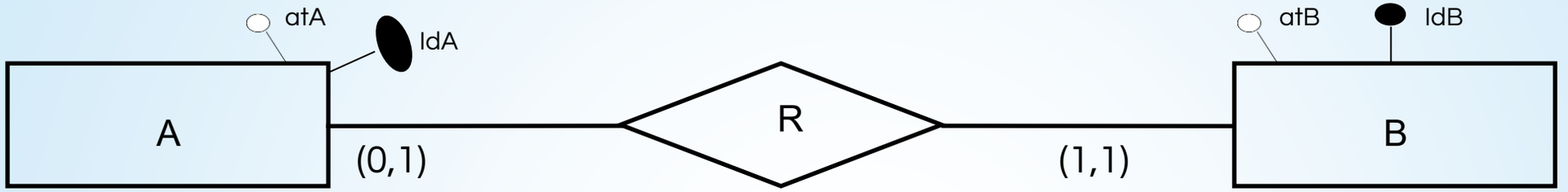
Conversión de Relaciones

➔Una relación puede o no ser una tabla.



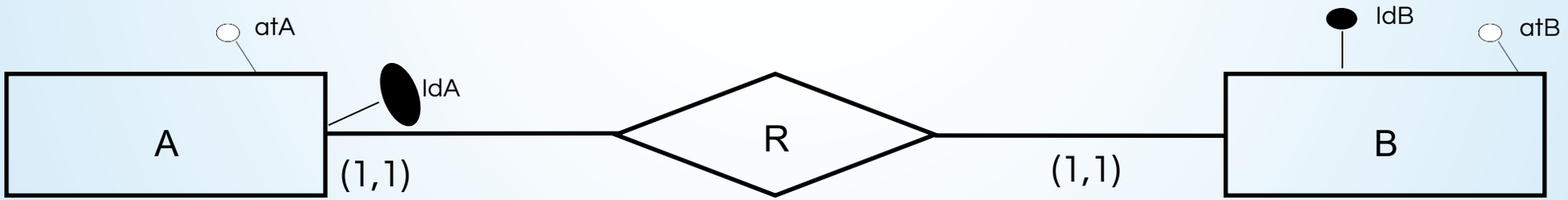
Que R sea una tabla o no depende de la cardinalidad de la relación.

Conversión de Relaciones



B=(<u>idB</u>, idA(fk), atB)

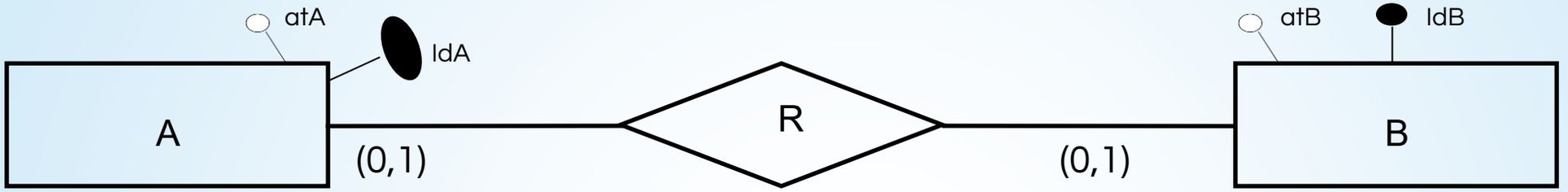
A=(<u>idA</u>, atA)



B=(<u>idB</u>, idA(fk), atB) ó B=(<u>idB</u>, atB)

A=(<u>idA</u>, atA) A=(<u>idA</u>, idB(fk), atA)

Conversión de Relaciones

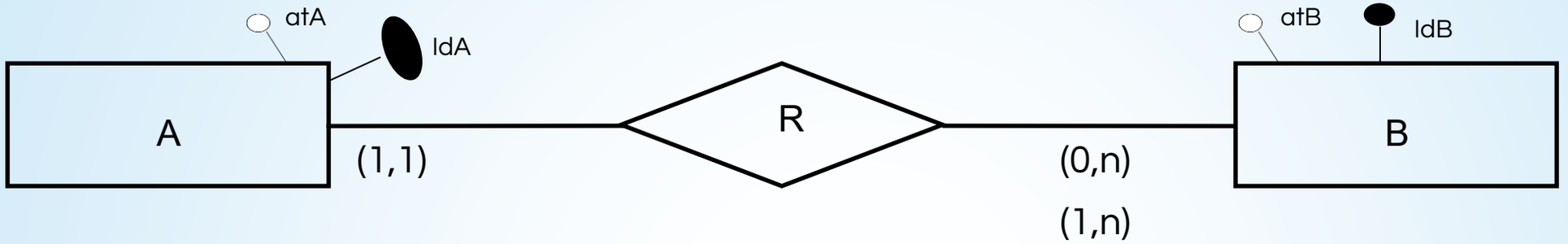


B=(<u>idB</u>, atB)

A=(<u>idA</u>, atA)

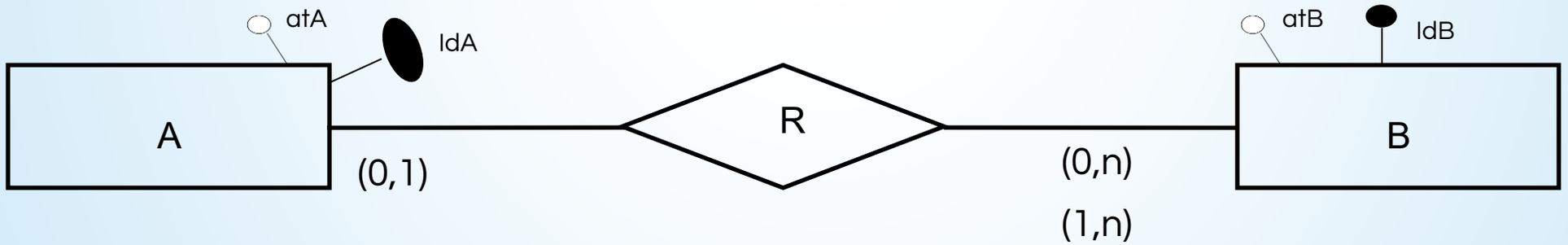
R=(<u>idA(fk)</u>, idB(fk)) o R=(idA(fk), <u>idB(fk)</u>)

Conversión de Relaciones



A=(<u>idA</u>, idB(fk), atA)

B=(<u>idB</u>, atB)

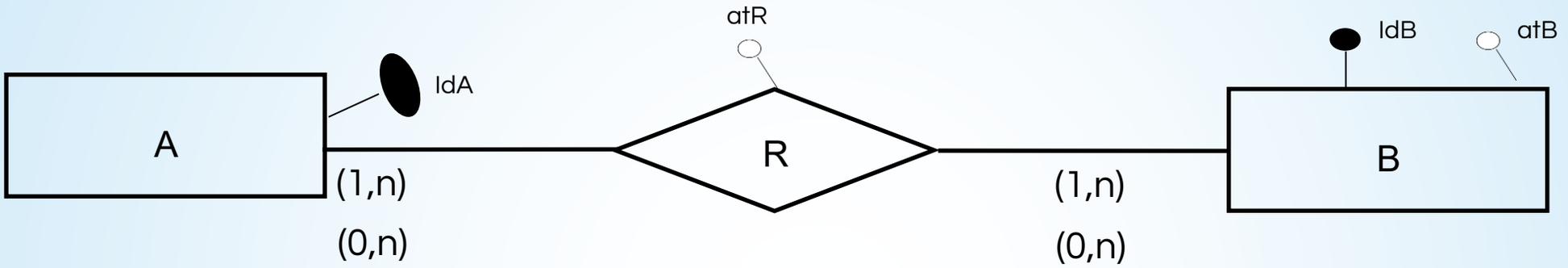


A=(<u>idA</u>, atA)

B=(<u>idB</u>, atB)

R=(<u>idA(fk)</u>, idB(fk))

Conversión de Relaciones



A=(<u>idA</u>)

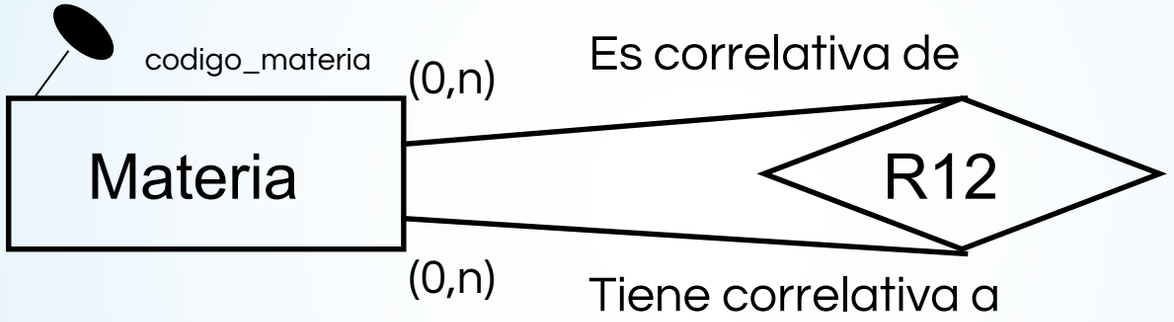
B=(<u>idB</u>, atB)

R=(<u>idA(fk), idB(fk)</u>, atR)

Dependiendo del dominio atR puede o no formar parte de la clave primaria.

Ejemplo en donde sería parte de la clave: un empleado puede trabajar en una misma área en dos períodos diferentes, por lo tanto necesito la fecha de inicio además de la clave del empleado y el área.

Conversión de Relaciones



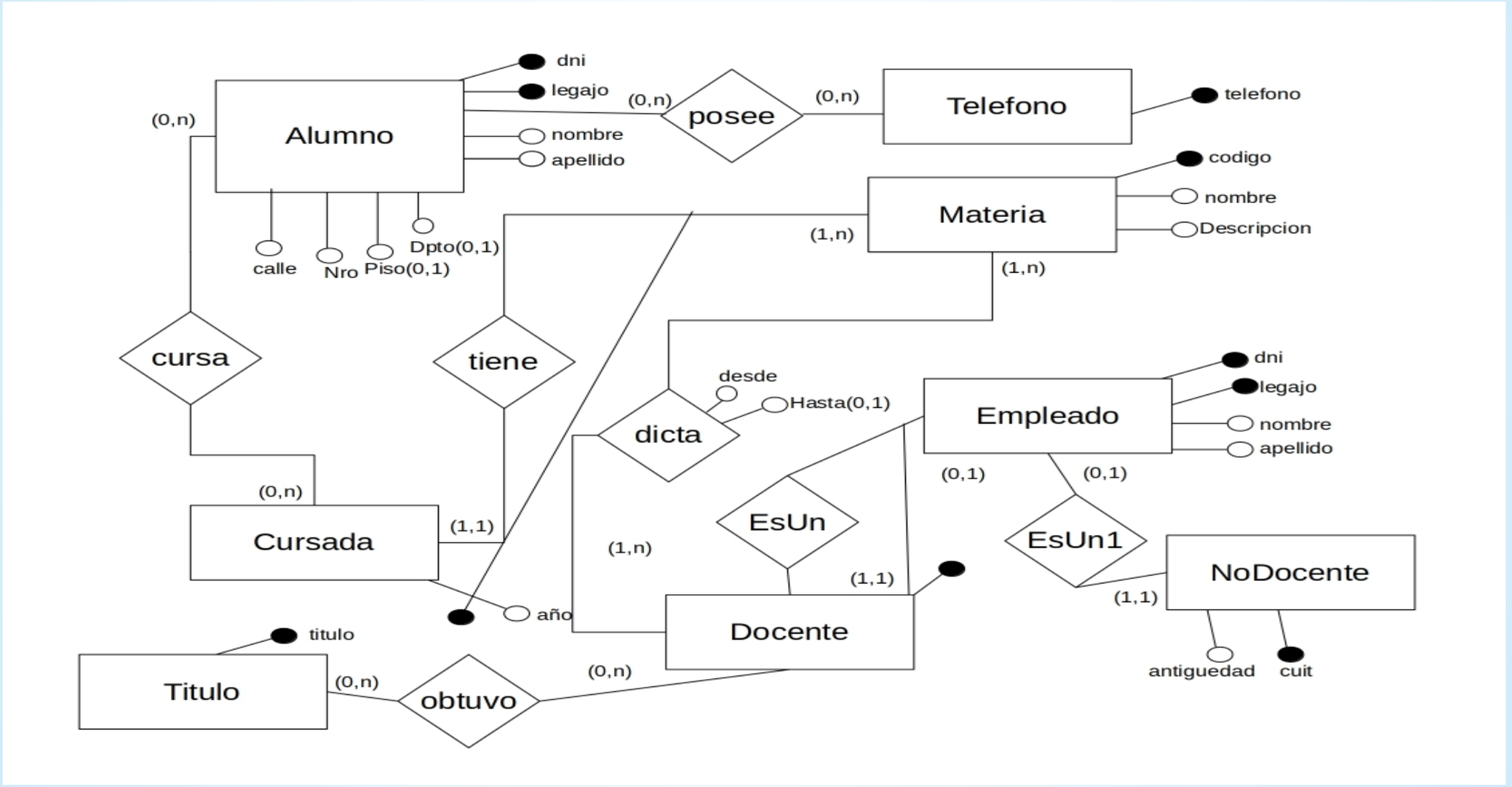
R12 = (Código_Materia(fk), Código_Materia_Correlativa(fk))

Debo renombrar!

Modelado Físico

¿Pasamos el modelo lógico propuesto al físico?

Modelo Lógico



Modelo Físico Final

Alumno = (dni, legajo, nombre, apellido, calle, nro, piso?, dpto?)
Telefono = (telefono)
Materia = (codigo, nombre, descripcion)
Cursada = (año, codigo (fk))
Empleado = (dni, legajo, nombre, apellido)
Docente = (dni(fk))
NoDocente = (cuit, dni(fk), antiguedad)
Titulo = (titulo)
Obtuvo= (dni(fk), titulo(fk))
posee = (dni(fk), telefono(fk))
dicta = (codigo(fk), dni(fk), desde, hasta?)
cursa = (dni(fk), (año, codigo) (fk))

Errores típicos

- No marcar claves primarias en alguna tabla
- Indicar una clave primaria compuesta porque había dos posibles identificadores en vez de elegir uno
- Introducir un campo “id” de forma innecesaria
- Resolver incorrectamente las relaciones que tienen cardinalidad opcional en algún lado
- Marcar claves primarias no mínimas
- No subrayar con una línea continua la PK en caso de ser compuesta
- No marcar las FK en las tablas (tanto las derivadas de entidades como de relaciones)
- Olvidarse convertir alguna entidad o relación
- En jerarquías ya resueltas, trasladar todos los atributos de la tabla padre a las hijas en lugar de solo su clave primaria.

Diseño de Bases de Datos

Álgebra relacional (AR)

Se denomina álgebra relacional a un conjunto de operaciones simples sobre tablas, a partir de las cuales se definen operaciones más complejas mediante composición. Define, por tanto, un lenguaje de manipulación de datos.

Operaciones - Selección $\sigma_p (T)$

Produce una tabla que contiene únicamente aquellas tuplas de T que satisfacen el predicado p.

Tabla persona

Nombre	Apellido
Pedro	Lopez
Carlos	Griguol
Juan	Perez
Carlos	Bilardo
Gustavo	Lopez

$\sigma_{\text{nombre='Carlos'}} (\text{persona})$

Nombre	Apellido
Carlos	Griguol
Carlos	Bilardo

Operaciones - Proyección

$$\pi_{al, \ldots an}(T)$$

Produce una tabla que tiene un subconjunto de atributos de T eliminando tuplas duplicadas.

Tabla persona

Nombre	Apellido
Pedro	Lopez
Carlos	Griguol
Juan	Perez
Carlos	Bilardo
Gustavo	Lopez

$$\pi_{\text{nombre}}(\text{persona})$$

Nombre
Pedro
Carlos
Juan
Gustavo

Operaciones - Unión $T1 \cup T2$

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	Lopez
Carlos	Griguol
Gustavo	Lopez

Tabla tenis

Nombre	Apellido
Rafael	Nadal
Roger	Federer
Gustavo	Lopez

futbol ∪ tenis

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

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	Lopez
Carlos	Griguol
Gustavo	Lopez

Tabla tenis

Nombre	Apellido
Rafael	Nadal
Roger	Federer
Gustavo	Lopez

futbol ∩ tenis

Nombre	Apellido
Gustavo	Lopez

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	Lopez
Carlos	Griguol
Gustavo	Lopez

Tabla tenis

Nombre	Apellido
Rafael	Nadal
Roger	Federer
Gustavo	Lopez

futbol – tenis

Nombre	Apellido
Pedro	Lopez
Carlos	Griguol

Operaciones – producto cartesiano

$$T1 \times T2$$

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

Tabla persona

nombre	apellido	codLoc
Pedro	Lopez	1
Carlos	Griguol	2
Carlos	Bilardo	1

Tabla localidad

codLoc	nombreLoc
1	La Plata
2	Berisso

persona x localidad

nombre	apellido	persona .codLoc	localidad .codLoc	nombreLoc
Pedro	Lopez	1	1	La Plata
Pedro	Lopez	1	2	Berisso
Carlos	Griguol	2	1	La Plata
Carlos	Griguol	2	2	Berisso
Carlos	Bilardo	1	1	La Plata
Carlos	Bilardo	1	2	Berisso

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 persona

nombre	apellido	codLoc
Pedro	Lopez	1
Carlos	Griguol	2
Carlos	Bilardo	1

Tabla localidad

codLoc	nombreLoc
1	La Plata
2	Berisso

persona |x| localidad

nombre	apellido	codLoc	nombreLoc
Pedro	Lopez	1	La Plata
Carlos	Griguol	2	Berisso
Carlos	Bilardo	1	La Plata

Operaciones – Renombre

$$\rho T\,(Table)$$

Renombra la tabla Table a T.

Operaciones – Asignación

$$A \Leftarrow Consulta$$

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

Operaciones – División

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 deber estar incluido en T1.

Tabla personas

DNI	Nombre	Apellido
10000000	Pedro	Lopez
20000000	Carlos	Griguol
30000000	Pedro	Gonzalez

Tabla localidad

CodLoc	NombreLoc
1	La Plata
2	Berisso

Tabla viaje

IdViaje	Fecha	DNI	CodLoc
1	1/10/2024	10000000	1
2	3/10/2024	10000000	2
3	2/10/2024	2000000	2
4	4/10/2024	3000000	1

π DNI, CodLoc (Viaje) % π CodLoc (Localidad)

DNI
10000000

Actualización de tablas

Producto=(<u>codProd</u>, desc, existAct, existMin, pVAct)

- **Incorporar** el producto (1235, “tuerca de 9 mm”, 50, 10, $100):
  Producto $\Leftarrow$ Producto $\cup$ {(1235, “tuerca de 9 mm”, 50, 10, $100)}
- **Eliminar** el producto 893:
  Producto $\Leftarrow$ Producto $-$ $\sigma_{codProd=893}$(Producto)
- **Aumentar el 1% el precio de venta actual** de **todos** los productos:
  δ **pVAct $\Leftarrow$ pVAct * 1,01**(Producto)

Ejemplo práctico

Modelo Físico

PRODUCTO = (idproducto, nombre, códigobarra, preciocosto)
CLIENTE = (idcliente, nombre, dirección, idlocalidad(FK))
FACTURA = (idfactura, fecha, montofactura, idcliente(FK))
RENGLON = (idfactura(FK), renglon, idproducto(FK), precioventa, cantidad)
LOCALIDAD = (idlocalidad, descripcion)

- Obtener identificador, fecha y monto de todas las facturas del mes de agosto del año 2025.

$$\pi_{\text{idfactura, fecha, montofactura}} (\sigma_{\text{(fecha >= '01/08/2025')^(fecha <= '31/08/2025')}} (\text{Factura}))$$

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, descripcion)

- Obtener productos vendido al cliente Jose Perez.

$$\pi_{\text{nombre, preciocosto}} (\sigma_{\text{(cliente.nombre = “Jose Perez”)^(renglon.idProducto=producto.idproducto)}} (\text{cliente} \;|x|\; \text{factura} \;|x|\; \text{renglon} \times \text{producto}))$$

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, descripcion)

- Obtener productos vendido al cliente Jose Perez pero no vendidos a Juan Garcia.

$$\pi_{\text{nombre, preciocosto}} (\sigma_{\text{(cliente.nombre = “Jose Perez”)^(renglon.idProducto=producto.idproducto)}} (\text{cliente} \;|x|\; \text{factura} \;|x|\; \text{renglon} \;x\; \text{producto}))$$

–

$$\pi_{\text{nombre, preciocosto}} (\sigma_{\text{(cliente.nombre = “Juan Garcia”)^(renglon.idProducto=producto.idproducto)}} (\text{cliente} \;|x|\; \text{factura} \;|x|\; \text{renglon} \;x\; \text{producto}))$$

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, descripcion)

- Obtener el nombre y la dirección de aquellos clientes que hayan comprado productos con un precio de costo menor a $100.

$$\pi_{\text{nombre, direccion}}((\pi_{\text{idfactura}}((\sigma_{(\text{preciocosto} < 100)}(\text{Producto})) \,|x|\, \text{Renglon})) \,|x|\, \text{Factura}) \,|x|\, \text{Cliente})$$

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, descripcion)

- Dar de baja las facturas del cliente 'Ramon Perez'.

$$FacturasDePerez \Leftarrow \pi_{idfactura,\ fecha,\ montofactura,idCliente}\ Factura\ |x|\ (\sigma_{(nombre='Ramon\ Perez')}(Cliente))$$

$$RenglonesDePerez \Leftarrow \pi_{idfactura,\ renglon,idProducto,\ precioVenta,\ cantidad}\ FacturasDePerez\ |x|\ Renglon$$

$$Renglones \Leftarrow Renglones - RenglonesDePerez$$

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, descripcion)

- Nombre de clientes que compraron todos los productos

$$(\pi_{\text{idcliente, nombre, idProducto}} (\text{Cliente} \mid x \mid \text{Factura} \mid x \mid \text{Renglon})) \,\%\, \pi_{\text{idproducto}} (\text{Producto}))$$

Errores típicos

- **Utilizar el producto natural cuando no hay atributos en común o cuando existen atributos con el mismo nombre que no corresponde unir.**
- **Utilizar producto cartesiano sin selección.**
- **Realizar unión, intersección y diferencia con esquemas que no sean unión compatibles (equivalentes).**
- **Realizar una división entre dos tablas Tabla1 y Tabla2 en donde el esquema de Tabla 2 no esté contenido en Tabla1.**
- **No respetar la integridad referencial, dejando inconsistente la BD al realizar una eliminación.**
- **Realizar cruces innecesarios con tablas que no se necesitan para resolver la consulta.**

¿Consultas?

Diseño de Bases de Datos

Structured Query Language(SQL)

Lenguaje de consultas de BD, compuesto por dos submódulos:

- Módulo para definición del modelo de datos, denominado **DDL** (Data Definition Language).
- Módulo para la operatoria normal de la BD, denominado **DML** (Data Manipulation Language).

SQL- Definición de datos

- De base de datos: **CREATE|DROP SCHEMA**
- De dominios: **CREATE|ALTER|DROP DOMAIN**
- De tablas: **CREATE|ALTER|DROP TABLE**
- De vistas: **CREATE|DROP VIEW**
- De índices (algunos SGBD): **CREATE|DROP INDEX**

SQL- Manipulación de datos

- **SELECT** - Sentencia utilizada para listar contenido de una o varias tablas
- **INSERT** - Sentencia utilizada para agregar contenido en una tabla
- **UPDATE** - Sentencia utilizada para actualizar contenido de una tabla
- **DELETE** - Sentencia utilizada para eliminar contenido de una tabla

SQL- Manipulación de datos

SELECT

Formato básico

SELECT campo/s

FROM tabla/s

Ejemplo:

Alumno = (dni, nombre, apellido)

SELECT nombre

FROM alumno

En el SELECT puedo utilizar * que me mostrará todos los campos de las tablas involucradas sin necesidad de escribir uno por uno.

SQL- Manipulación de datos

SELECT

DISTINCT: se aplica a campos del select y elimina **tuplas** repetidas.

Ejemplo:

Alumno = (dni, nombre, apellido)

```
SELECT DISTINCT nombre
FROM alumno
```

Operador	Significado	Ejemplo
=	es igual a	Select nombre, apellido 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)
<>	distinto a	Select nombre From alumno Where (dni <> 24564321)
BETWEEN	entre (se incluyen extremos)	Select nombre From alumno Where (dni between 30000000 and 40000000)
LIKE	como	ejemplo próxima diapositiva

SQL- Operadores

LIKE: brinda gran potencia para aquellas consultas que requieren manejo de Strings. Se puede combinar con:

- **%**: representa cualquier cadena de caracteres, inclusive la cadena vacía.
- **_ (guión bajo)**: sustituye sólo el carácter del lugar donde aparece.

```
SELECT nombre, apellido
FROM alumno
WHERE (apellido LIKE '%or%')
```

```
SELECT nombre, apellido
FROM alumno
WHERE (nombre LIKE '_ _ _')
```

SQL- Operadores

Los atributos utilizados en el SELECT de una consulta SQL pueden tener asociados operaciones válidas para sus dominios.

SELECT apellido, dni + 5000

FROM alumno

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

SELECT nombre, apellido

FROM alumno

WHERE (nombre **IS NOT NULL**)

SQL- Operadores

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

SELECT *

FROM alumno, localidad

SELECT a.nombre as 'Nombre alumno' ,l.nombre as 'Nombre localidad'

AS: Renombre de atributos.

FROM alumno a,localidad l

Alias definido para una tabla.

SQL- Producto natural

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)

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

SQL- Producto Natural

LEFT JOIN: contiene todos los registros de la tabla de la izquierda, aún cuando no exista un registro correspondiente en la tabla de la derecha, para uno de la izquierda. Retorna un valor nulo (NULL) en caso de no correspondencia.

RIGHT JOIN: es la inversa del LEFT JOIN.

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

SQL- Operadores

- **UNION**: misma interpretación que en AR. No retorna tuplas duplicadas.
- **UNION ALL**: misma interpretación que la UNION pero retorna las tuplas duplicadas.

IMPORTANTE: Las consultas a unir deben tener esquemas compatibles

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

SQL- Operadores

- **EXCEPT**: cláusula definida para la diferencia de conjuntos.

IMPORTANTE: ambas consultas deben tener esquemas compatibles

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

- **INTERSECT**: cláusula para la operación de intersección

SQL- Operadores

ORDER BY: permite ordenar las tuplas resultantes por el atributo que se le indique. Por defecto ordena de menor a mayor (operador **ASC**). Si se desea ordenar de mayor a menor, se utiliza el operador **DESC**.

SELECT nombre, apellido,dni

FROM alumno

WHERE (dni>23000000) and (nombre='Luciana')

ORDER BY apellido, dni DESC

Dentro de la cláusula ORDER BY se pueden indicar más de un criterio de ordenación. El segundo criterio se aplica en caso de empate en el primero y así sucesivamente.

SQL- Funciones de agregación

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.

SQL- Agrupamiento

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

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

Qué información se puede mostrar cuando se realizó un agrupamiento?
Porque es importante agrupar además por PK?

SQL- Agrupamiento clausula HAVING

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 AVG(nota) >= 6
```

SQL- Subconsultas

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

SQL- Subconsultas

SELECT nombre, apellido

FROM alumno

WHERE dni **IN (SELECT** dni **FROM** examen **WHERE** nota < 4**)**

SQL- Cláusula Exists

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

```
SELECT dni, nombre, apellido
FROM alumno a
WHERE EXISTS (SELECT * FROM examen e  WHERE
a.dni=e.dni  and  e.nota=10 )
```

Condición de la consulta principal

SQL- Equivalencia AR

- T1|×|T2 ≡ SELECT * FROM T1 NATURAL JOIN T2 (equicombinación de tuplas de T1 y T2 sobre la foreign key de una que referencie a la primary key de la otra)
- T1|×|T2 ≡ SELECT * FROM T1 INNER JOIN T2 ON (...) (equicombinación de tablas sobre los campos que se especifiquen –cuando se quieren igualar campos que no coincidan con la *primary key* de una y la *foreign key* de la otra)
- T1|×|θT2 ≡ SELECT * FROM T1 NATURAL JOIN T2 WHERE
- T1 ×|T2 ≡ SELECT * FROM T1 LEFT JOIN T2 ON (...)
- T1 – T2 ≡ T1 EXCEPT T2

SQL- ABM

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

INSERT INTO alumno (dni, nombre, apellido) **VALUES** (2827893, ‘Raul’,'Perez');

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

DELETE FROM alumno **WHERE** dni=2222222**;**

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

UPDATE alumno **SET** nombre='Lorenzo' WHERE dni=22232425

SQL

¿Consultas?

Diseño de Bases de Datos

SQL

DML

Sucursal = (nombreSucursal, ciudadSucursal, activo)
Cliente = (codCliente, dni, nombCliente, dirCliente, viveCliente)
Prestamo = (nroPrestamo, importe, nombreSucursal(fk))
PropietarioPrestamo = (codCliente(fk), nroPrestamo(fk))
Cuenta = (nroCuenta, saldo, nombreSurcursal(fk))
PropietarioCuenta = (nroCuenta(fk), codCliente(fk))

- Ej: clientes con cuentas o préstamos en cualquier sucursal
- Ej: clientes con cuentas y préstamos en cualquier sucursal
- Ej: clientes con cuentas y sin préstamos

SQL

DML

1. **Clientes con cuentas o préstamos en cualquier sucursal**

   (**Select** dni, nombCliente
   From PropietarioCuenta **natural join** cliente) **Union**
   (**Select** dni, nombCliente
   From PropietarioPrestamo **natural join** cliente)

2. **Clientes con cuentas y préstamos en cualquier sucursal**

   (**Select** dni, nombCliente
   From PropietarioCuenta **natural join** cliente) **Intersect**
   (**Select** dni, nombCliente **From** PropietarioPrestamo **natural join** cliente)

3. **Clientes con cuentas y sin préstamos**

   (**Select** dni, nombCliente
   From PropietarioCuenta **natural join** cliente) **Except**
   (**Select** dni, nombCliente
   From PropietarioPrestamo **natural join** cliente)

SQL

DML

Sucursal = (nombreSucursal, ciudadSucursal, activo)
Cliente = (codCliente, dni, nombCliente, dirCliente, viveCliente)
Prestamo = (nroPrestamo, importe, nombreSucursal(fk))
PropietarioPrestamo = (codCliente(fk), nroPrestamo(fk))
Cuenta = (nroCuenta, saldo, nombreSurcursal(fk))
PropietarioCuenta = (nroCuenta(fk), codCliente(fk))

- Ej: cantidad de cuentas con saldo mayor a $50000
- Ej: saldo promedio de las cuentas de la sucursal ‘X’
- Ej: importe del mayor préstamo otorgado por la sucursal ‘Y’
- Ej: importe total asignado a prestamos

SQL

DML

Cantidad de cuentas con saldo mayor a $50000

Select COUNT(nroCuenta) as cantCuentas podria contar cualquier campo

From Cuenta

Where saldo > 50000

Saldo promedio de las cuentas de la sucursal 'X'

Select AVG(saldo) as Promedio

From Cuenta

Where nombreSucursal = 'X'

SQL

DML

Importe del mayor préstamo otorgado por la sucursal ‘Y’

```
Select MAX(importe) as Maximo
From Prestamo
Where nombreSucursal = ‘Y’
```

Importe total asignado a prestamos

```
Select Sum(importe) as importe
From Prestamo
```

SQL

DML

Sucursal = (nombreSucursal, ciudadSucursal, activo)
Cliente = (codCliente, dni, nombCliente, dirCliente, viveCliente)
Prestamo = (nroPrestamo, importe, nombreSucursal(fk))
PropietarioPrestamo = (codCliente(fk), nroPrestamo(fk))
Cuenta = (nroCuenta, saldo, nombreSurcursal(fk))
PropietarioCuenta = (nroCuenta(fk), codCliente(fk))

- Ej: obtener saldo promedio de las cuentas de c/ sucursal
- Ej: presentar las sucursales junto con el saldo promedio de sus cuentas siempre y cuando supere los $200.000
- Ej: contar el Nº de clientes que tienen cuentas en cada sucursal
- Ej: saldo promedio de las cuentas de c/ cliente que vive en La Plata y tiene al menos 3 cuentas

SQL

DML

Obtener saldo promedio de las cuentas de c/ sucursal

```
Select nombreSucursal, avg(saldo) as saldoProm
From cuenta
Group by nombreSucursal
```

Presentar las sucursales y su saldo promedio siempre y cuando supere los $200.000

```
Select nombreSucursal, avg(saldo)
From cuenta
Group by nombreSucursal
Having avg(saldo) > 200000
```

SQL

DML

Contar el Nº de clientes que tienen cuentas de cada sucursal

```
Select nombreSucursal, count(distinct codCliente)
From cuenta inner join propietariocuenta
on( cuenta.nroCuenta = propietariocuenta.nroCuenta)
Group by nombreSucursal
```

Saldo promedio de las cuentas de c/ cliente que vive en La Plata y tiene al menos 3 cuentas

```
Select nombCliente, avg(saldo)
From propietariocuenta inner join cuenta
(propietariocuenta.nroCuenta= cuenta.nroCuenta )
Inner join cliente on (propietariocuenta. codcliente =
cliente.codCliente )
Where viveCliente = “La Plata”
Group by propietariocuenta.codcliente, cliente.nombCliente
Having count (propietariocuenta.nroCuenta) >= 3
```

SQL

DML

Clientes con préstamos y cuentas en cualquier sucursal (alternativa a ∩)

```
Select distinct nombCliente
From propietarioprestamo  inner join cliente on
(propietarioprestamo. codcliente = cliente.codcliente  )
Where codcliente in (
        Select codcliente
        From propietariocuenta
)
```

SQL

DML

Clientes con préstamos y cuentas en una misma sucursal de "La Plata"

Select distinct nombCliente

From propietarioprestamo pp **inner join** prestamo p on(pp.nroPréstamo = p.nroPrestamo) **inner join** sucursal s on (p.nombreSucursal = s.nombreSucursal)

inner join cliente c on(pp.codCliente=c.codCliente)

Where s.ciudadSucursal="La Plata" **and** c.codCliente **in**

(**Select** codCliente

From propietariocuenta pc **inner join** cuenta c on (pc.nroCuenta = c.nroCuenta)

Where c.nombreSucursal = **s**.nombreSucursal)-> s corresponde a la consulta ppal

SQL

DML

Sucursal = (nombreSucursal, ciudadSucursal, activo)
Cliente = (codCliente, dni, nombCliente, dirCliente, viveCliente)
Prestamo = (nroPrestamo, importe, nombreSucursal(fk))
PropietarioPrestamo = (codCliente(fk), nroPrestamo(fk))
Cuenta = (nroCuenta, saldo, nombreSurcursal(fk))
PropietarioCuenta = (nroCuenta(fk), codCliente(fk))

- Ej: presentar las sucursales que tengan activo mayor que alguna otra
- Ej: presentar la sucursal que tenga activo superior a todas
- Ej: encontrar la sucursal que tiene el mayor saldo promedio
- Ej: clientes con cuenta en todas las sucursales

SQL

DML

Presentar las sucursales que tengan activo mayor que alguna otra

```
Select nombreSucursal
From sucursal
Where activo > some ( select activo
                      from sucursal)
```

Presentar la sucursal que tenga activo superior a todas

```
Select nombreSucursal
From sucursal
Where activo > =all ( select activo
                      from sucursal)
```

SQL

DML

Encontrar la sucursal que tiene el mayor saldo promedio

```
Select nombreSucursal
From cuenta
Group by nombreSucursal
Having avg (saldo) >= all (
        Select avg (saldo)
        From cuenta
        Group by nombreSucursal
)
```

SQL

DML

Cliente con cuentas en todas las sucursales

```
Select nombCliente
From cliente
where not exists
(select * from sucursal s
      Where not exists(select * from propietariocuenta pc
    inner join cuenta c on (pc.nroCuenta=c.nroCuenta)
     where c.nombreSucursal=s.nombreSucursal and
    pc.codCliente=cliente.codCliente))-> corresponden a las
    consultas de nivel superior
```