INGENIERÍA INDUSTRIAL
UNIVERSIDAD DE CHILE

FACULTAD DE CIENCIAS
FÍSICAS Y MATEMÁTICAS
UNIVERSIDAD DE CHILE

IN3501 - Tecnologías de Información y Comunicaciones para la Gestión

CAPA DE DATOS

PROFESORES

Evelyn Andaur
Juan D. Velásquez
Gastón L'Huillier
Víctor Rebolledo Lorca

Temario

- Introducción.
- Redes, Internet y Web.
- Cliente Servidor de Múltiples Capas.
- La capa de datos.
- La capa de negocios.
- La capa de presentación.
- Administración de proyectos informáticos.

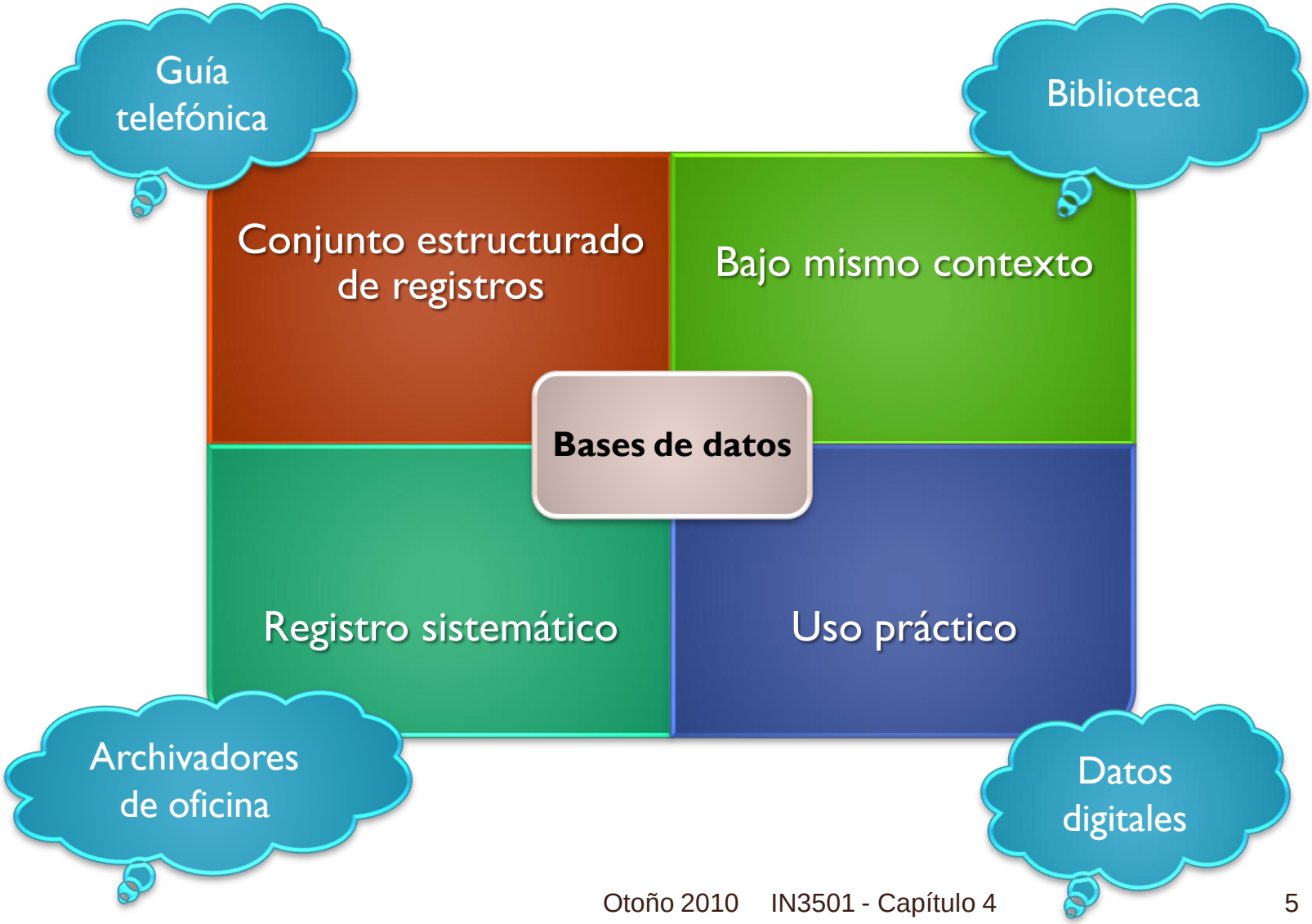
Capítulo IV

CAPA DE DATOS

Agenda

- Bases de datos
  - Definición
  - SABD o DBMS
- Modelamiento relacional
  - Modelo Entidad Relación
  - Normalización
  - Ejercicio
- Lenguaje SQL
  - Orientado a la consulta
  - Orientado a la administración
- Indexación

Bases de datos – Definición



Definiciones

- **Base de Datos**:
  "Un conjunto lógicamente coherente de datos relacionados, construido para una cierta aplicación".
- **Sistema Administración de Bases de Datos (SABD o DBMS):**
  "Software que permite a las bases de datos ser definidas, además de construidas y operarlas".
- **Diagrama Entidad – Relación (ER):**
  Modelo lógico de la base de datos para representar información de manera estructurada.
- **Structured Query Language (SQL):**
  Lenguaje de consulta estructurado para interactuar con la BD.

Bases de Datos – SADB o DBMS

Administradores de Bases de datos relacionales

DBMS Open Source

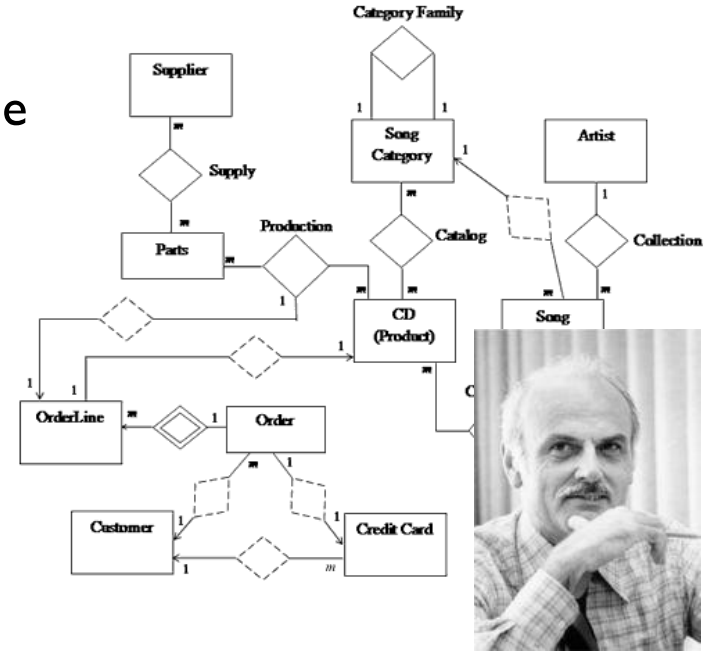
DBMS Comerciales

Operaciones de Persistencia o CRUD

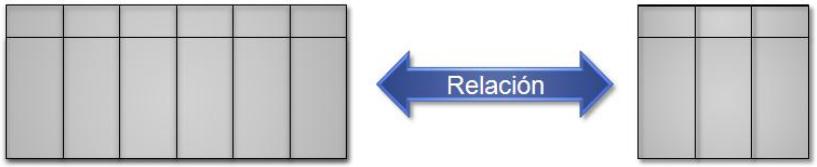
Create
Read
Update
Delete

Modelo Entidad-Relación

- **Representación conceptual** de datos estructurados mediante un conjunto de Entidades y Relaciones entre estas entidades.
- Basado en:
  - Lógica de predicado
  - Teoría de conjuntos



- Postulado en 1970 por **Edgar Frank Codd** en laboratorios de IBM
  - Nuevo paradigma en el Modelamiento de datos

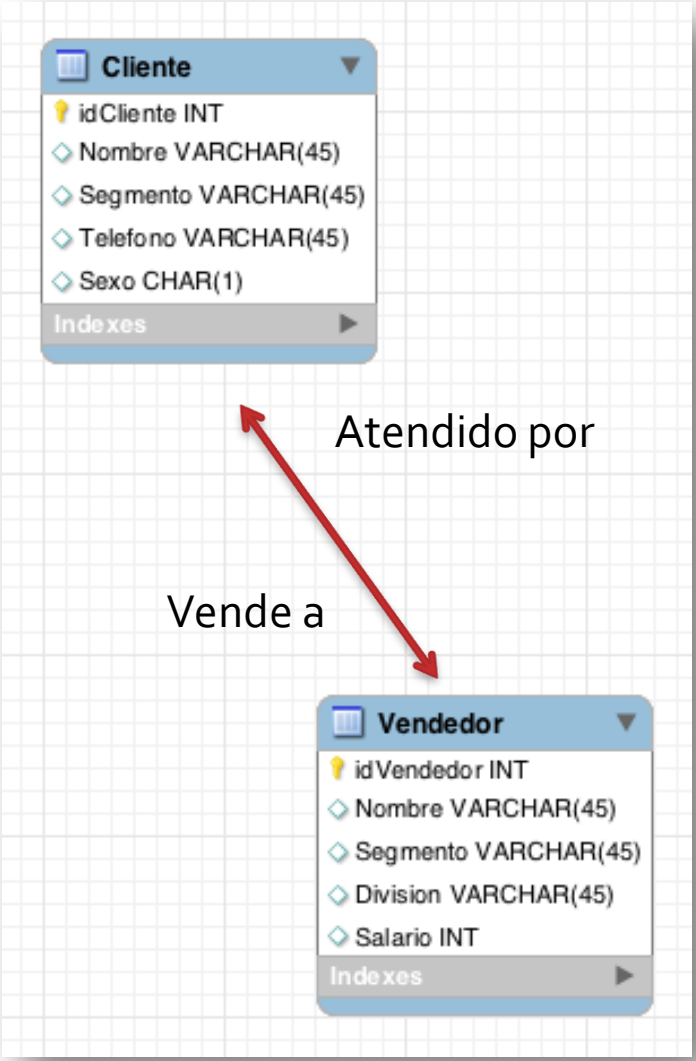


Modelo Entidad-Relación (2)

- **Características**
  - Proceso **Top-Down**
  - *Abstracción del negocio* y llevarlo a *entidades* y *relaciones*
  - El **modelo es genérico**
    - Es necesaria una herramienta para implementarlo: DBMS

- **Ventajas**
  - Relativamente *fácil de entender*
  - *Base matemática sólida*
  - *Estandarizado*
  - Cualquier consulta es *factible* de ser resuelta
  - Reduce ***redundancia***
    - Minimiza los errores de **consistencia**

Modelo Entidad-Relación
Definiciones



- **Entidad**: Objetos que puedo caracterizar dentro del problema a modelar.
  Por ejemplo: clientes, vendedores, compañías, etc.
- **Relación**: Asociación directa que ocurre entre dos entidades
- **Atributo**: Características que definan la entidad. Por ejemplo: Nombre, Segmento, Teléfono, Edad, etc.

Modelo Entidad-Relación
Los Atributos

- **Llave única**: Corresponde a uno o dos atributos cuyos valores identifican de forma única los registros de una tabla.

  En otras palabras, **sus valores no se repiten**.

  Por ejemplo:
  - **RUT** de un cliente
  - **Número de Matricula** y **Código Curso** para la entidad que describe las inscripciones de alumnos en los distintos cursos. (el par jamás se repite)

Modelo Entidad-Relación

Los Atributos

- **Llave primaria**: Es una **llave única** que sirve para construir las relaciones con otras entidades.
- **Atributo no primario**: Es aquel atributo que **no pertenece** a la llave primaria.

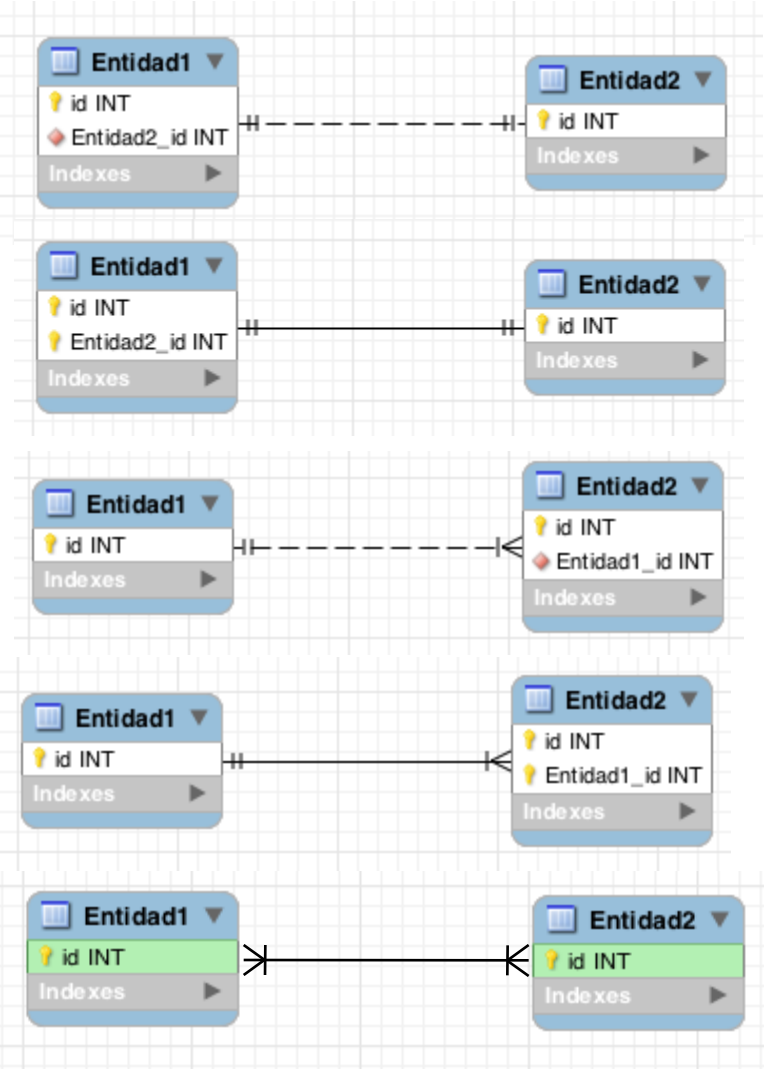
Modelo Entidad-Relación

Sobre las relaciones

- En el modelo Entidad-Relación, las relaciones se representan mediante *referencias de la llave primaria de una entidad sobre otra entidad*.
- **Llave foránea**: Corresponde a la referencia de una llave primaria de otra entidad relacionada
- **OJO:**
  - No siempre la llave foránea formará parte de la llave primaria de la entidad donde reside.

Modelo Entidad-Relación

Tipos de Relaciones



Cada registro de la Entidad1 puede estar relacionado con 0 o 1 registro de la Entidad2

Cada registro de la Entidad1 está relacionado a sólo 1 registro de la Entidad2

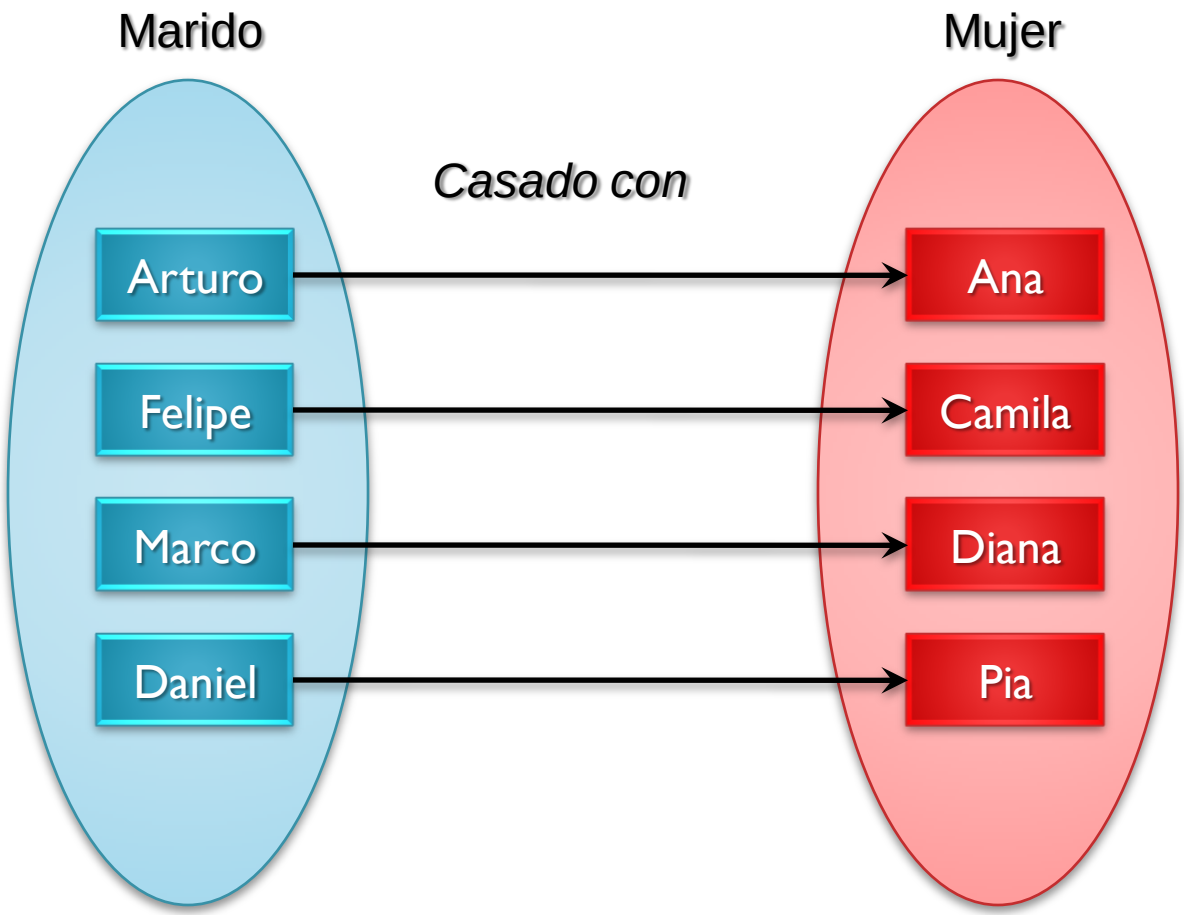
Cada registro de la Entidad1 puede estar relacionado con 0 o N registros de la Entidad2 (N >= 1)

Cada registro de la Entidad1 está relacionado con N registros de la Entidad2 (N >= 1)

Un registro de la Entidad1 está relacionado con N registros de la Entidad2 (N >= 1) & Un registro de la Entidad2 está relacionado con N registros de la Entidad1 (N >= 1)

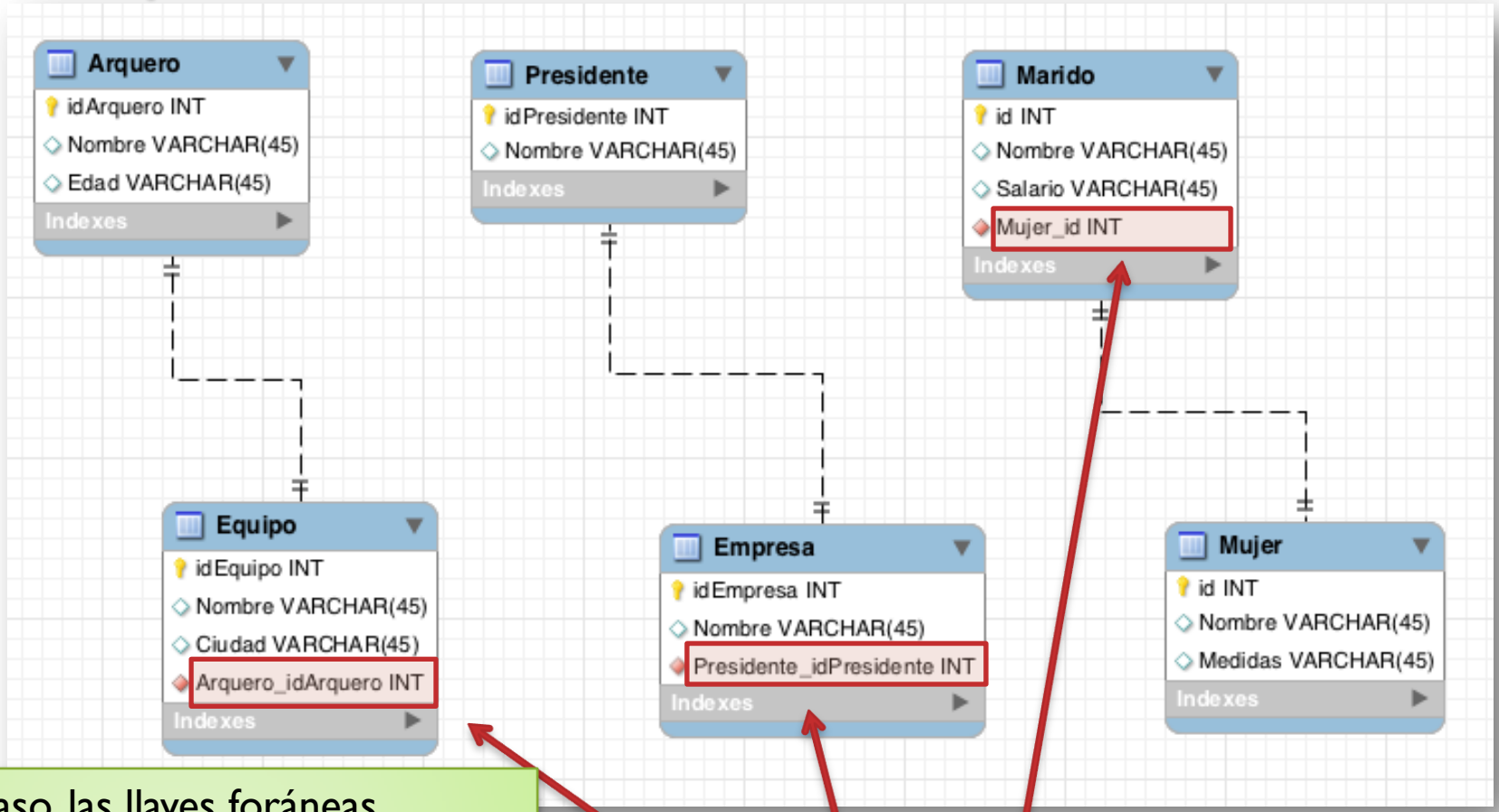
Modelo Entidad-Relación

Tipos de Relaciones: "1 a 1"



Modelo Entidad-Relación

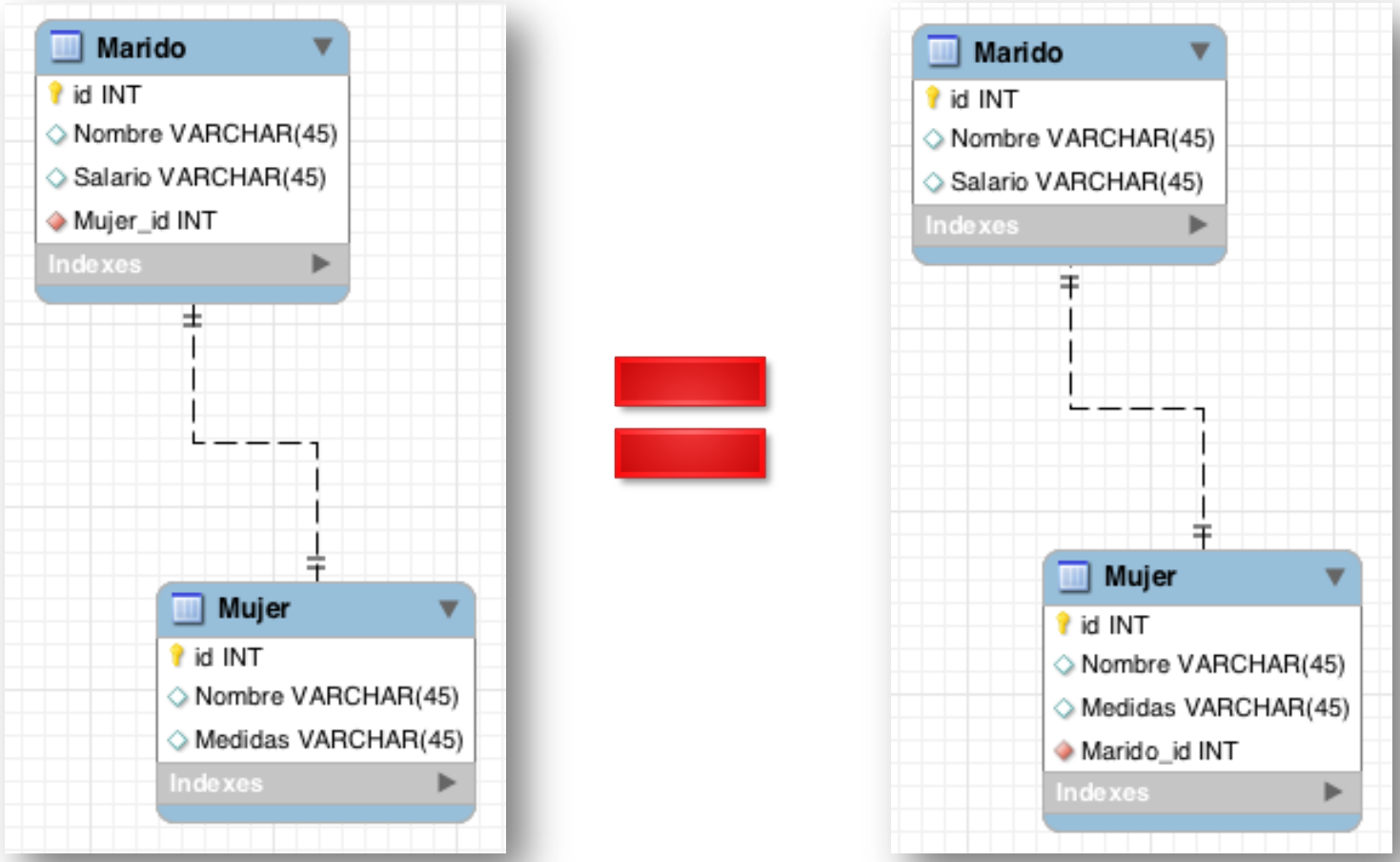
Tipos de Relaciones:"1 a 1"



En este caso, las llaves foráneas **NO** son parte de las llaves primarias de las entidades: Equipo, Empresa y Marido

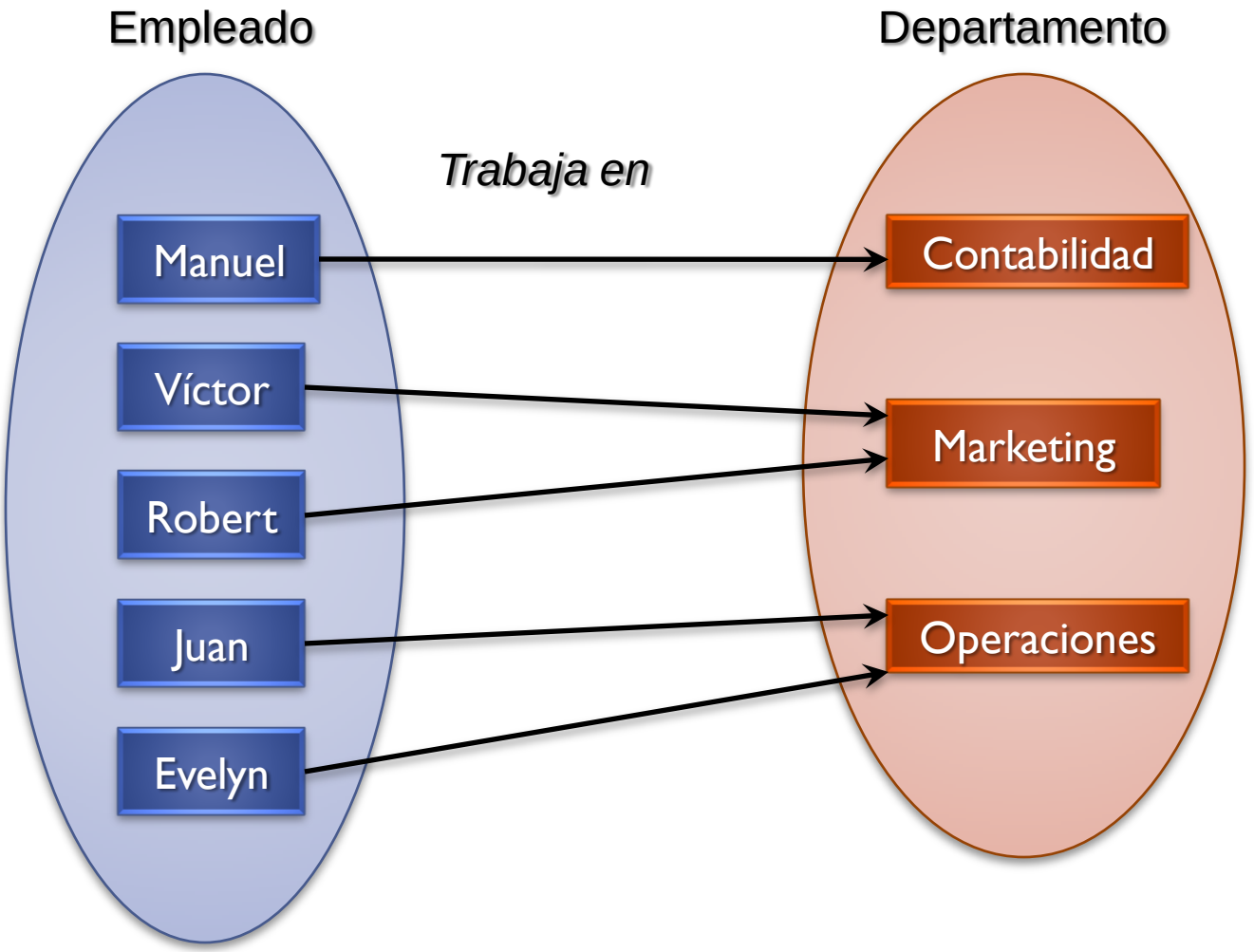
Modelo Entidad-Relación
Tipos de Relaciones:"1 a 1"

- OJO!!!



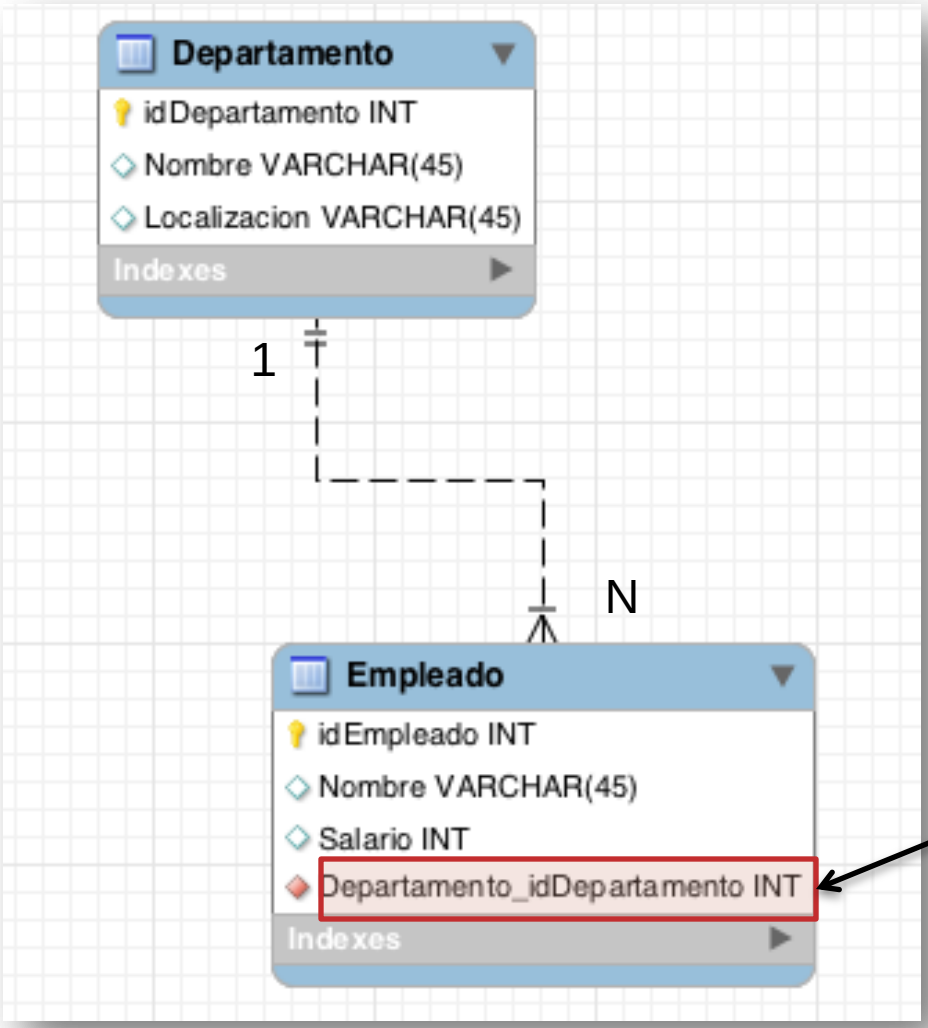
Modelo Entidad-Relación

Tipos de Relaciones: "1 a N"



Modelo Entidad-Relación

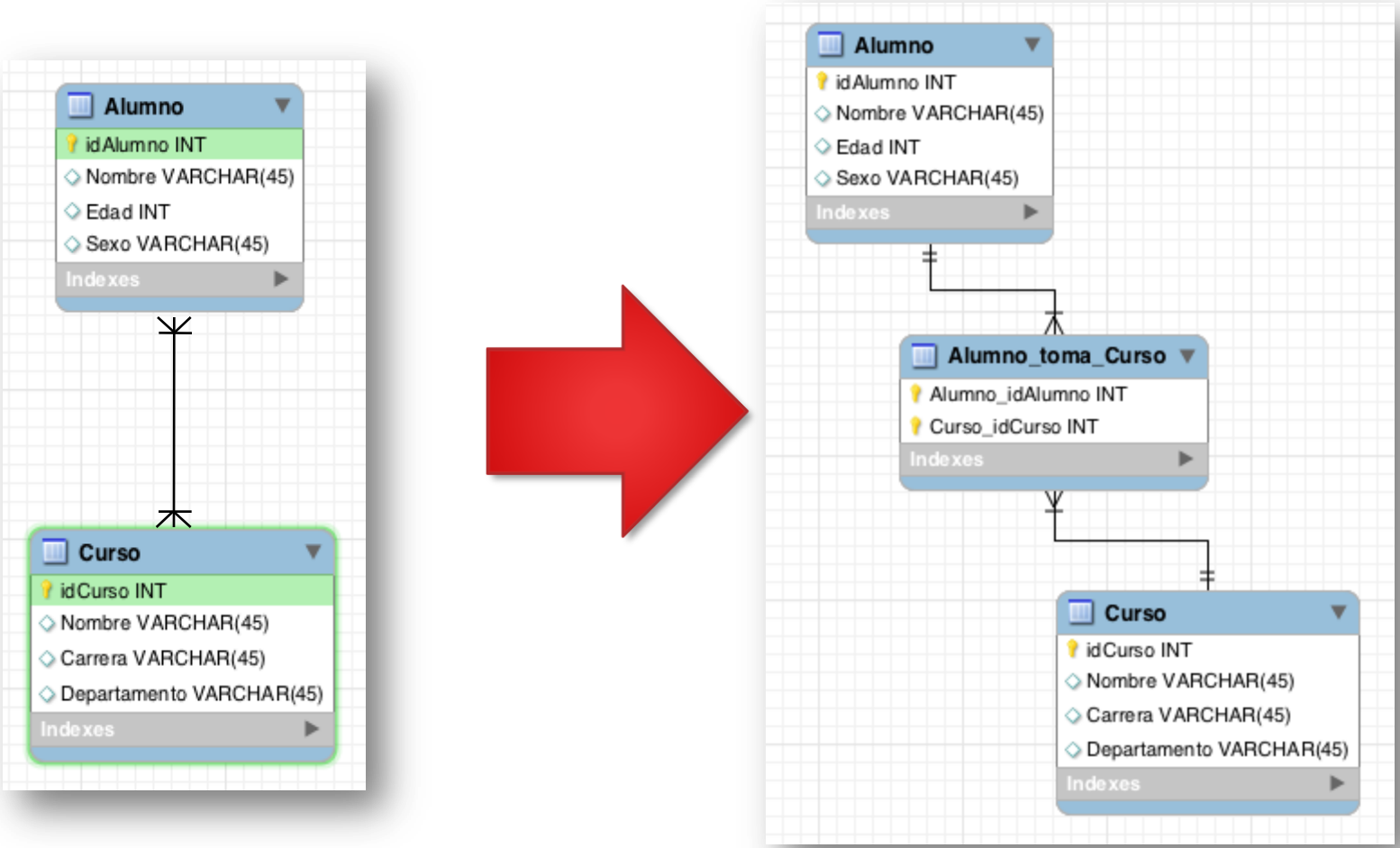
Tipos de Relaciones: "1 a N"



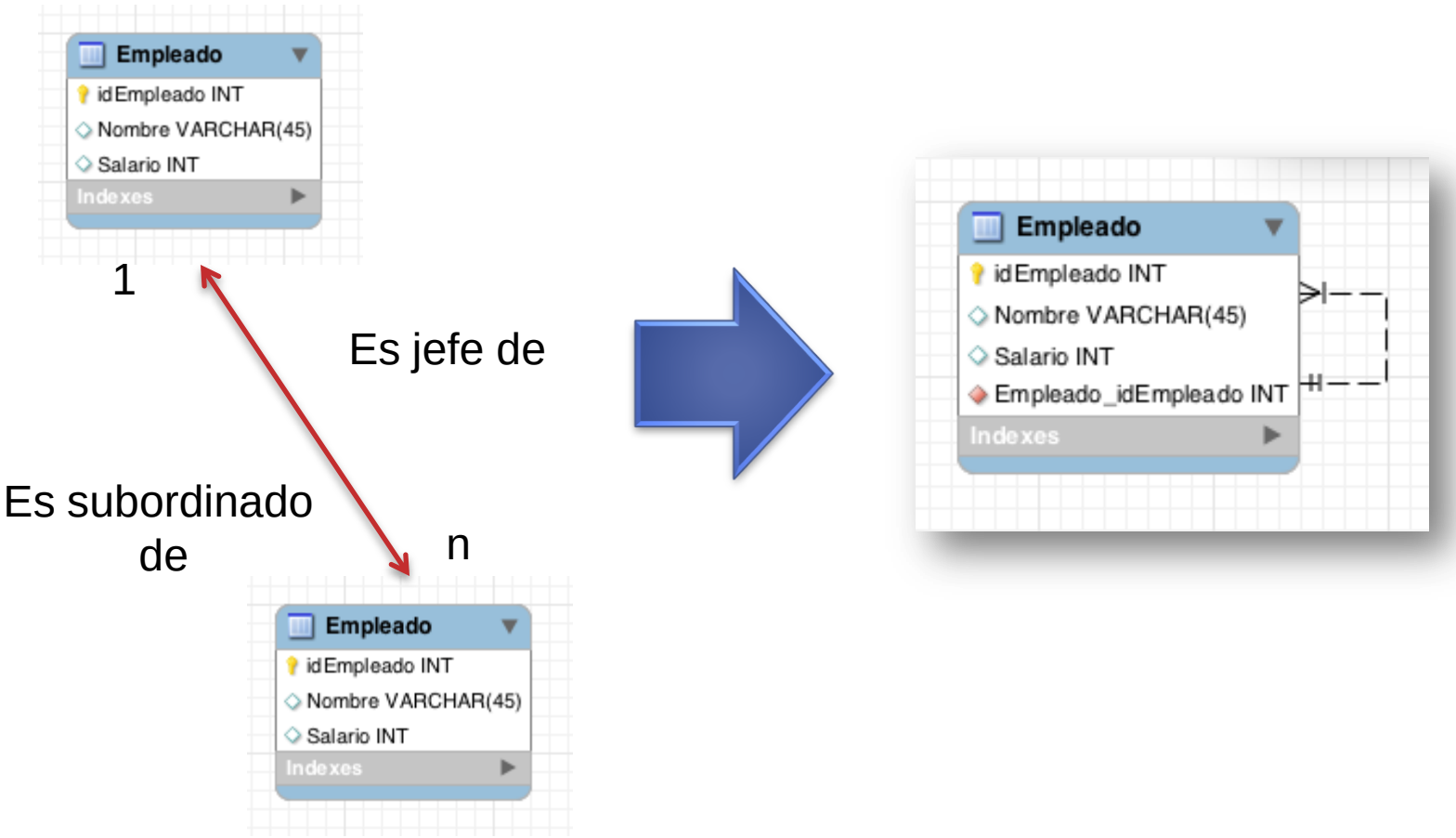
OJO con la posición de la llave foránea (en el lado de N)

Modelo Entidad-Relación

Tipos de Relaciones: "N a M"



Modelo Entidad-Relación
Relaciones Reflexivas

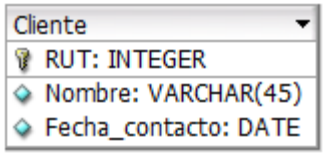


Capa de Datos

NORMALIZACIÓN

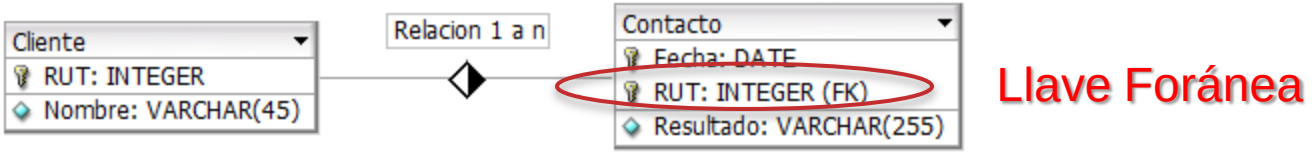
Normalización – Primera Forma Normal

- Exige que cada fila en la tabla esté libre de **"Grupos repetitivos"**
  - Cada atributo de la entidad debe ser **atómico**



RUT	Nombre	Fecha_contacto
17476803-4	Juan D.Velásquez	10/5/2009 22/01/2010
13456980-2	Pablo Catalán	30/4/2009

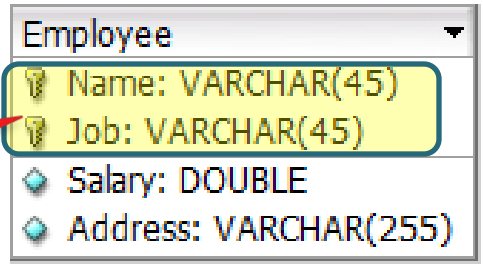
Atributo **Fecha de Contacto** es multivariado



Llave Foránea

Normalización – Segunda Forma Normal

- Exige que **todo registro** debe ser accedido a través de la **llave primaria completa.**



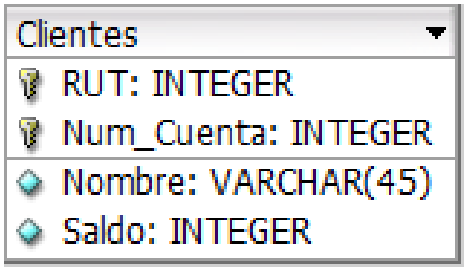
Llave compuesta

Name	Job	Salary	Address
Smith	Welder	14.75	123 4th St
Smith	Programmer	24.50	123 4th St
Smith	Waiter	7.50	123 4th St
Jones	Programmer	26.5	4 Moose Lane
Jones	Bricklayer	.50	4 Moose Lane
Adams	Analyst	28.50	88 Tiger Circle

* **Address** sólo depende de **Name**
* **Salary** sólo depende de **Job**

Redundancia en datos

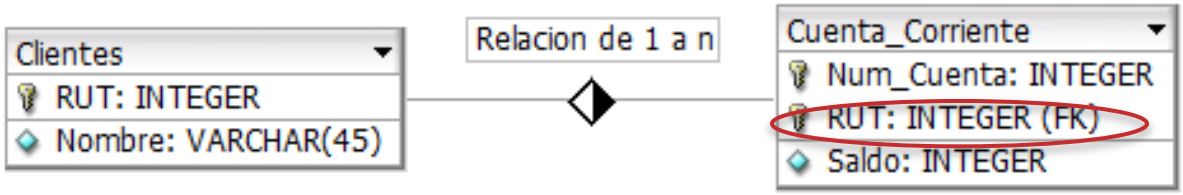
Normalización – Segunda Forma Normal (Ejemplo II)



Nombre sólo depende de **RUT**

Saldo sólo depende de **Num_Cuenta**

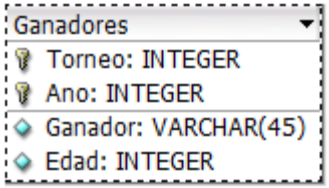
→ Separar en dos entidades



Llave Foránea

Normalización – Tercera Forma Normal

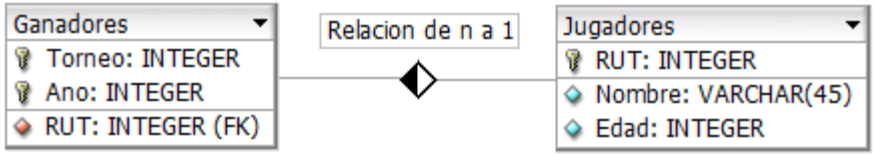
- Una entidad estará en 3FN cuando:
  - Está en Segunda Forma Normal (2FN)
  - Ningún atributo no-primario de la tabla es dependiente transitivamente de la clave primaria.



El nombre del **Ganador** depende del **Torneo** y del **Año** en que se realizó

La **Edad** depende directamente del nombre del **Ganador**

➔ **Edad** depende transitivamente de la llave primaria



EJERCICIO

Sistema de Ventas

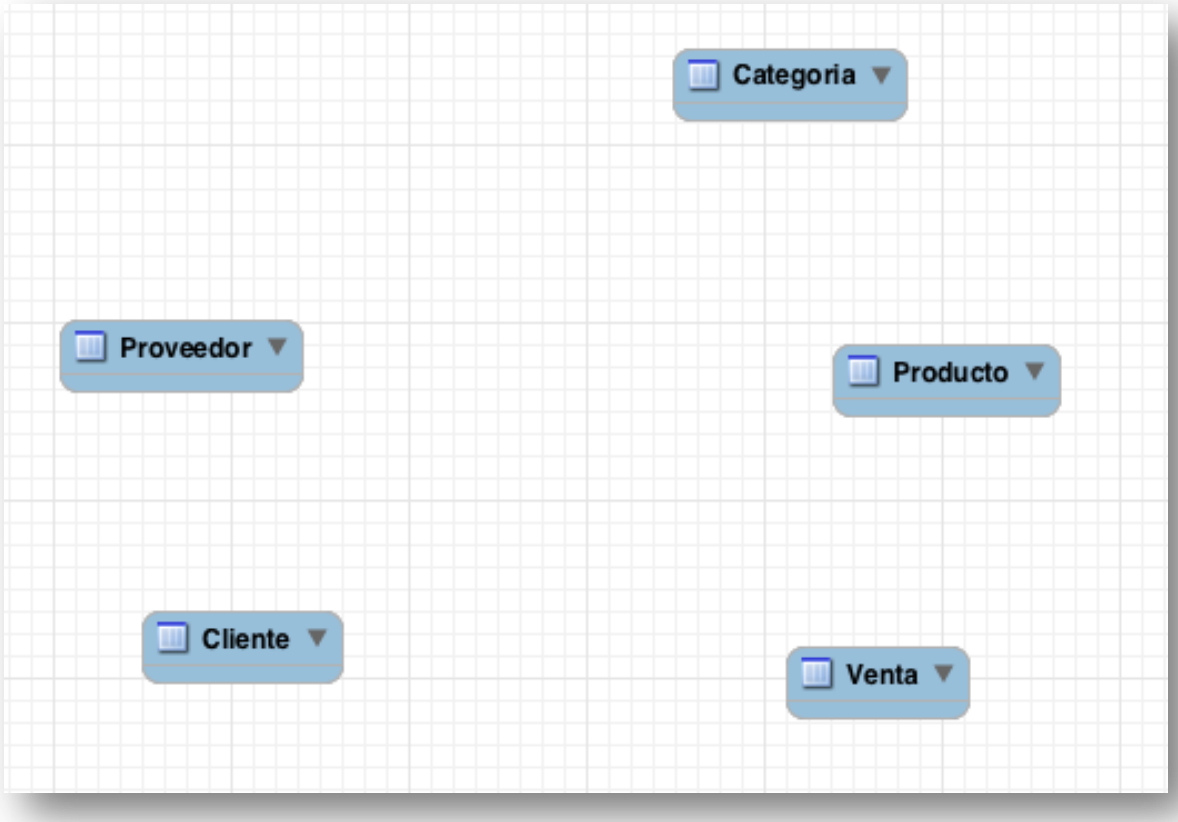
- Le contratan para hacer una BD que permita apoyar la gestión de un sistema de ventas.
- La empresa necesita llevar un control de proveedores, clientes, productos y ventas.
- Un proveedor tiene un RUT, nombre, dirección, teléfono y página web. Un cliente también tiene RUT, nombre, dirección, pero puede tener varios teléfonos de contacto. La dirección se entiende por calle, número, comuna y ciudad. Un producto tiene un id único, nombre, precio actual, stock y nombre del proveedor.
- Además se organizan en categorías, y cada producto va sólo en una categoría. Una categoría tiene id, nombre y descripción.
- Por razones de contabilidad, se debe registrar la información de cada venta con un id, fecha, cliente, descuento y monto final. Además se debe guardar el precio al momento de la venta, la cantidad vendida y el monto total por el producto.

Solución
Paso 1: Identificar Entidades

- Le contratan para hacer una BD que permita apoyar la gestión de un sistema de ventas.
- La empresa necesita llevar un control de **proveedores**, **clientes**, **productos** y **ventas**.
- Un **proveedor** tiene un RUT, nombre, dirección, teléfono y página web. Un **cliente** también tiene RUT, nombre, dirección, pero puede tener varios teléfonos de contacto. La dirección se entiende por calle, número, comuna y ciudad. Un **producto** tiene un id único, nombre, precio actual, stock y nombre del proveedor.
- Además se organizan en **categorías**, y cada producto va sólo en una categoría. Una **categoría** tiene id, nombre y descripción.
- Por razones de contabilidad, se debe registrar la información de cada **venta** con un id, fecha, cliente, descuento y monto final. Además se debe guardar el precio al momento de la venta, la cantidad vendida y el monto total por el producto.

Solución

Paso 1: Identificar Entidades

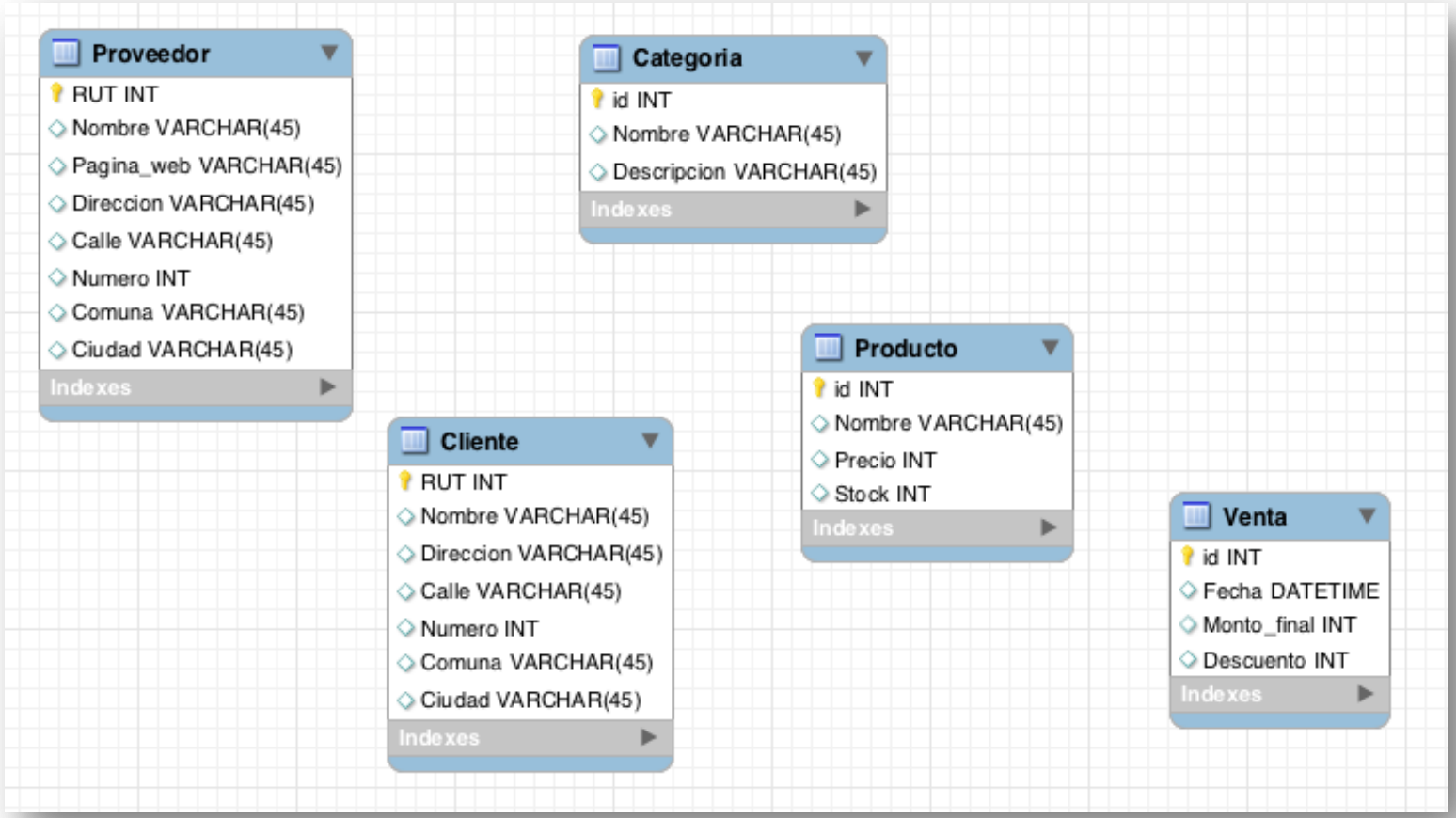


Solución
Paso 2: Identificar Atributos

- Le contratan para hacer una BD que permita apoyar la gestión de un sistema de ventas.
- La empresa necesita llevar un control de proveedores, clientes, productos y ventas.
- Un **proveedor** tiene un *RUT, nombre, dirección, teléfono y página web*. Un **cliente** también tiene *RUT, nombre, dirección*, pero puede tener varios *teléfonos de contacto*. La **dirección** se entiende por *calle, número, comuna y ciudad*. Un **producto** tiene un *id único, nombre, precio actual, stock y nombre del proveedor*.
- Además se organizan en **categorías**, y cada producto va sólo en una categoría. Una categoría tiene *id, nombre y descripción*.
- Por razones de contabilidad, se debe registrar la información de cada **venta** con un *id, fecha, cliente, descuento y monto final.* Además se debe guardar el precio al momento de la venta, la cantidad vendida y el monto total por el producto.

Solución

Paso 2: Identificar Atributos



Solución
Paso 3: Identificar Relaciones

- Le contratan para hacer una BD que permita apoyar la gestión de un sistema de ventas.
- La empresa necesita llevar un control de proveedores, clientes, productos y ventas.
- Un proveedor tiene un RUT, nombre, dirección, teléfono y página web. Un cliente también tiene RUT, nombre, dirección, pero puede tener varios teléfonos de contacto. La dirección se entiende por calle, número, comuna y ciudad. **Un producto tiene un id único, nombre, precio actual, stock y nombre del proveedor**.
- Además se organizan en categorías, y **cada producto va sólo en una categoría**. Una categoría tiene id, nombre y descripción.
- Por razones de contabilidad, se debe registrar **la información de cada venta con un id, fecha, cliente, descuento y monto final**. Además se debe guardar el precio al momento de la venta, la cantidad vendida y el monto total por el producto.

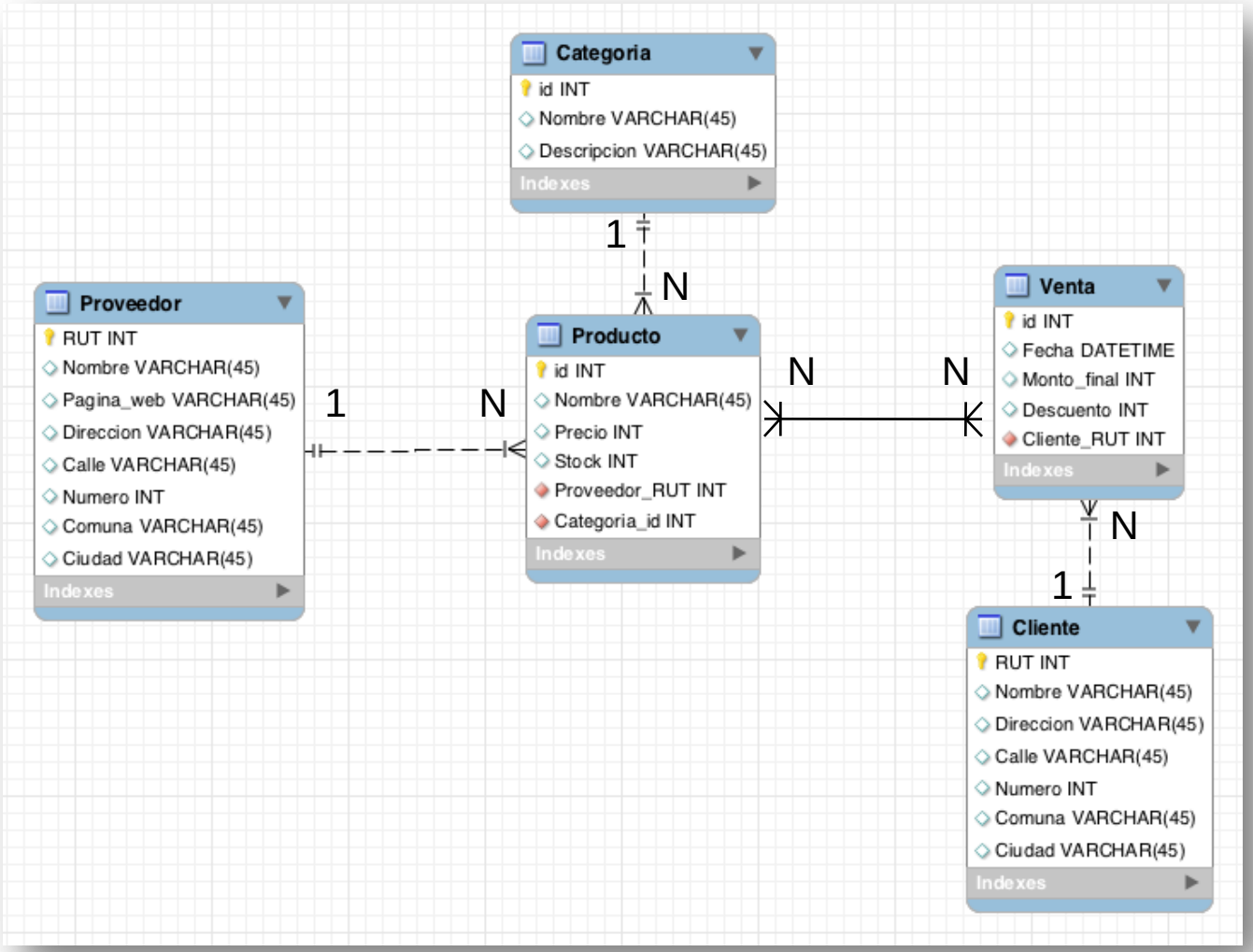
Solución

Paso 3: Identificar Relaciones

- **Proveedor – Producto**
  - Un producto es provisto por un solo proveedor
  - Un proveedor puede proveer más de un producto
  - → **Relación "1 a N"**
- **Categoría – Producto**
  - Un producto pertenece a una sola categoría
  - Una categoría puede tener muchos productos
  - → **Relación de "1 a N"**
- **Cliente – Venta**
  - Un venta esta asociada sólo a un cliente
  - Un cliente puede hacer múltiples ventas
  - → **Relación de "1 a N"**
- **Venta – Producto**
  - Una venta puede tener múltiples productos
  - Un producto puede estar inserto en múltiples ventas
  - → **Relación de "N a M"**

Solución

Paso 3: Identificar Relaciones

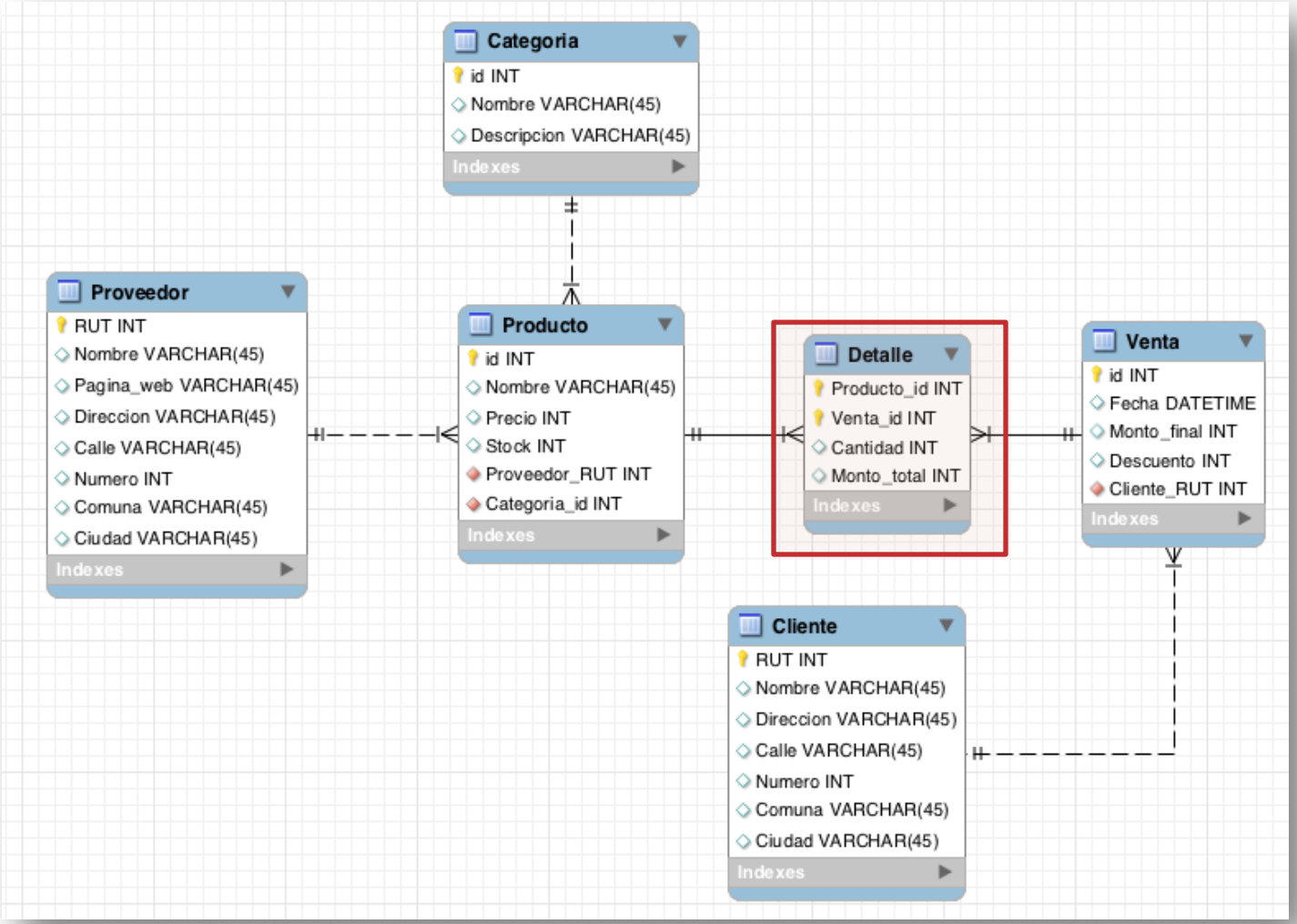


Solución
Paso 4: Entidades y Relaciones no directas

- Le contratan para hacer una BD que permita apoyar la gestión de un sistema de ventas.
- La empresa necesita llevar un control de proveedores, clientes, productos y ventas.
- Un proveedor tiene un RUT, nombre, dirección, teléfono y página web. Un cliente también tiene RUT, nombre, dirección, pero puede tener varios teléfonos de contacto. La dirección se entiende por calle, número, comuna y ciudad. Un producto tiene un id único, nombre, precio actual, stock y nombre del proveedor.
- Además se organizan en categorías, y cada producto va sólo en una categoría. Una categoría tiene id, nombre y descripción.
- Por razones de contabilidad, se debe registrar la información de cada venta con un id, fecha, cliente, descuento y monto final. **Además se debe guardar el precio al momento de la venta, la cantidad vendida y el monto total por el producto.**

Solución

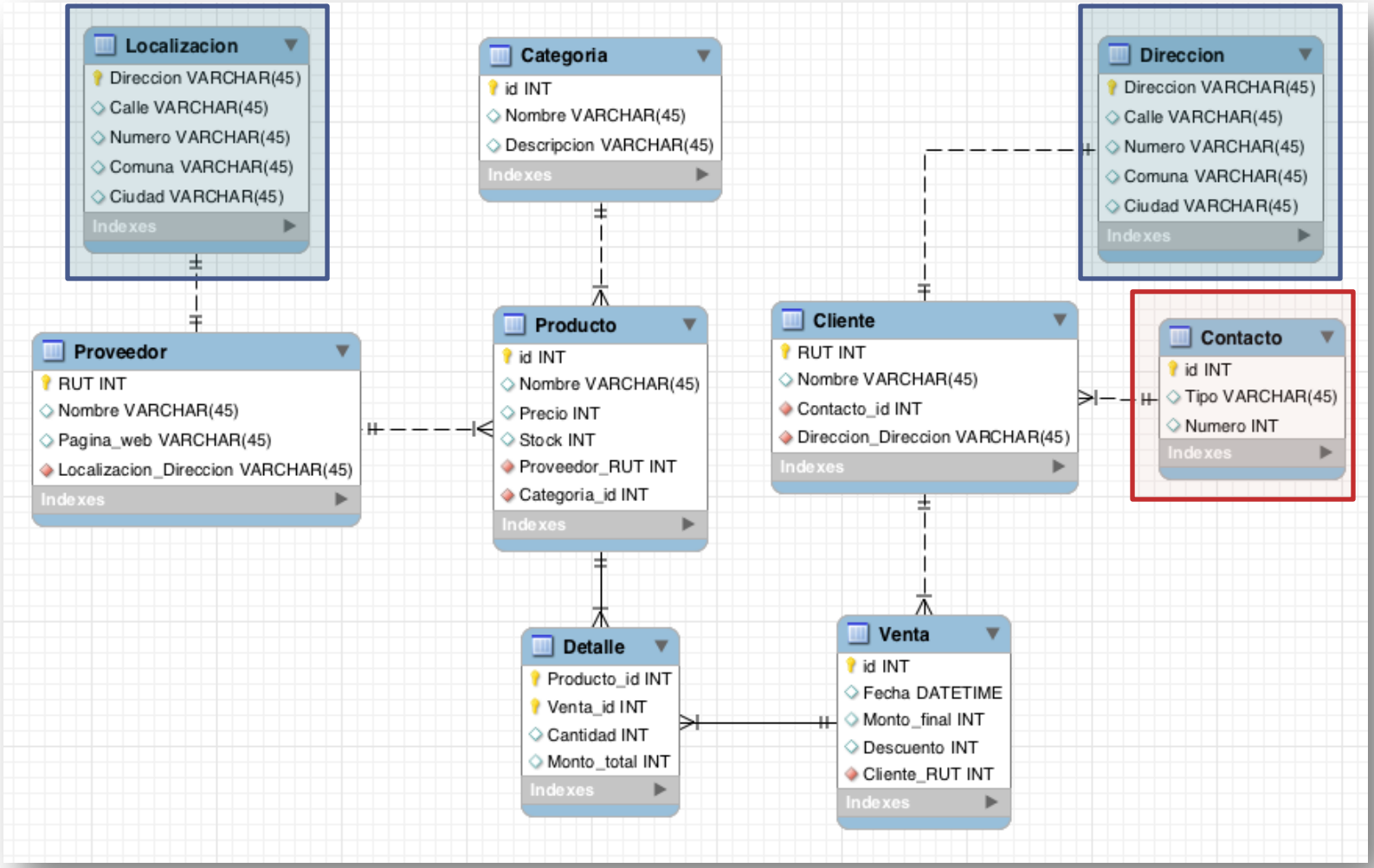
Paso 4: Entidades y Relaciones no directas



Solución
Paso 5: Normalizar

- Le contratan para hacer una BD que permita apoyar la gestión de un sistema de ventas.
- La empresa necesita llevar un control de proveedores, clientes, productos y ventas.
- Un proveedor tiene un RUT, nombre, dirección, teléfono y página web. **Un cliente también tiene RUT, nombre, dirección, pero *puede tener varios teléfonos de contacto (viola 1FN)***. La **dirección se entiende por calle, número, comuna y ciudad (viola 3FN)**. Un producto tiene un id único, nombre, precio actual, stock y nombre del proveedor.
- Además se organizan en categorías, y cada producto va sólo en una categoría. Una categoría tiene id, nombre y descripción.
- Por razones de contabilidad, se debe registrar la información de cada venta con un id, fecha, cliente, descuento y monto final. Además se debe guardar el precio al momento de la venta, la cantidad vendida y el monto total por el producto.

Sistema de Ventas



Capítulo IV

CAPA DE DATOS

Agenda

- Bases de datos
  - Definición
  - SABD o DBMS
- Modelamiento relacional
  - Modelo Entidad Relación
  - Normalización
  - Ejercicio
- Lenguaje SQL
  - Orientado a la consulta
  - Orientado a la administración
- Indexación

Lenguaje SQL

- Modelo ER → Modelamiento de base de datos
- DBMS → Implementar dicho modelo
- **Conclusión**: Tenemos datos almacenados

Lenguaje SQL

Lenguaje SQL - Historia

- En 1974, en el Laboratorio de IBM en San Jose D. Chamberlin definió un lenguaje llamado '*Structured English Query Language*' o SEQUEL.
- Una versión mejorada de SEQUEL/ fue definida en 1976, pero su nombre fue cambiado por razones legales SQL.
- IBM produjo posteriormente un DBMS prototipo llamado System R, basado en SEQUEL/2
- En 1987, ANSI e ISO publicaron un estándar inicial para SQL.
- En 1989, ISO publicó una adición que definió ' una característica de realce de la integridad '.
- En 1992, la primera revisión importante al estándar de ISO ocurrió, designado SQL2 o SQL/92.

Objetivos

- SQL **Orientado a la Consulta**:
  - Uso de la componente **WHERE** para agregar condiciones a las Consultas.
  - Ordenar resultados de las Consultas usando **ORDER BY**.
  - Uso de Funciones Adicionales.
  - Agrupar datos usando **GROUP BY** y **HAVING**.
  - Uso de Sub-Consultas (o consultas anidadas).
  - Usar **JOIN** entre tablas.
  - Operaciones de Conjuntos (**UNION**, **INTERSECT**, **EXCEPT**)
- SQL **Orientado a transacciones y Administración**
  - Como realizar transacciones sobre la Base de Datos usando (**INSERT**, **UPDATE**, and **DELETE**.)
  - Comandos **CREATE** y **DELETE** (usuarios, tablas, etc.).

Lenguaje de consulta SQL

- Permite **obtener cualquier conjunto** de datos presentes en una base de datos relacional mediante una sentencia.

```
SELECT [DISTINCT | ALL]
{* | [column_expression [AS new_name]] [,...] }
FROM table_name [alias] [, ...]
[WHERE condition]
[GROUP BY column_list]
[ HAVING condition]
[ORDER BY column_list]
```

Lenguaje de consulta SQL

```
SELECT [DISTINCT | ALL]
{* | [column_expression [AS new_name]] [,...] }
FROM table_name [alias] [, ...]
[WHERE condition]
[GROUP BY column_list]
[ HAVING condition]
[ORDER BY column_list]
```

FROM Especifica las tablas que se usaran
WHERE Establece los filtros.
SELECT Especifica las columnas que se consultaran.
GROUP BY Permite agrupar los datos.
HAVING Permite generar filtros sobre los grupos de datos.
ORDER BY Especifica el orden de los datos.

SQL
Seleccionar Atributos

- Listar **todos los atributos** de la tabla **Staff**.

```
SELECT sno, fname, lname, address, tel_no
FROM   Staff;
```

- Se puede usar * como abreviación para todas las columnas :

```
SELECT * FROM Staff;
```

- Se puede limitar la consulta con el comando **limit**

```
SELECT * FROM Staff LIMIT 10
```

SQL
Uso de DISTINCT

- Para eliminar aquellos registros **duplicados**, se recomienda el uso del comando DISTINCT, por ejemplo en la tabla **Staff**.

```
SELECT DISTINCT fname, lname
FROM   Staff;
```

- Con la consulta anterior, se entrega un listado con **todos los nombres y apellidos distintos** presentes en la tabla **Staff**.

SQL
Campos calculados

- Se pueden realizar operaciones matemáticas a campos cuyo tipo lo permita (NUMBER; INT; DOUBLE; LONG, etc.)

```
SELECT fname, salary/12 AS salario_mensual
FROM   Staff;
```

- Para renombrar el campo a obtener, se puede utilizar el comando AS.
- Otras operaciones matemáticas comúnmente utilizadas son: ABS, MOD, EXP, SQRT, LN, CEILING, FLOOR, ROUND, POW

SQL
Condición de búsqueda

- A cualquier consulta se le pueden agregar condiciones de despliegue de información. Con el comando WHERE, se agregan las condiciones:

```
SELECT fname, salary/12 AS salario_mensual
FROM   Staff
WHERE salario_mensual > 500.000;
```

- Al renombrar el campo con 'AS', se puede utilizar esta variable en lo que sigue de la consulta.
- Recordar toda consulta se puede describir como **"Seleccionar X, de la tabla Y, que cumpla Z"**

SQL
Comparación y Ordenamiento

- Existe una serie de comandos utilizados para distintas acciones. Por ejemplo,
  - **LIKE o RLIKE:** permite comparar cadenas de caracteres

```
SELECT nombre
FROM Clientes
WHERE nombre RLIKE 'juan'
```

  - **ORDER BY:** permite ordenar sobre algún atributo

```
SELECT nombre, sueldo, edad
FROM empleados
ORDER BY sueldo DESC, edad ASC
```

  - Generalmente se utiliza junto con **DESC** o **ASC** si se quiere hacer un ordenamiento Descendente o Ascendente.

SQL
Sentencias Agregadas

- Los operadores de agregación se utilizan para realizar operaciones complejas sobre los atributos de las consultas:
  - **MIN:** permite encontrar el valor mínimo sobre un conjunto de valores
  - **MAX:** permite encontrar el valor máximo
  - **AVG:** permite determinar el valor promedio
  - **COUNT:** permite contar la cantidad de valores obtenidos
  - **SUM:** permite sumar los valores asociados a un conjunto de valores.
- Estos operadores, se pueden utilizar en conjunto con el operador de agrupamiento **GROUP BY** y **HAVING**

SQL
Sentencias Agregadas (2)

- **GROUP BY** permite agrupar los resultados sobre una de las columnas o atributos indicados en la consulta.

```
SELECT SUM(monto_ventas), nombre, id_empleado
FROM Empleados
GROUP BY id_empleado
```

- La consulta anterior entrega la suma total de las ventas asociadas a un empleado en particular
- Dependiendo de la consulta, se debe tener cuidado al agrupar por valores que sean diferenciadores (e.g. "id_empleado" y no "nombre")

SQL
Sentencias Agregadas (3)

- **HAVING** permite filtrar los resultados utilizando funciones de agregación en su argumento.

```
SELECT AVG(salario), profesion
FROM Empleados
GROUP BY profesion
HAVING AVG(salario) > 5000000
```

- La consulta anterior entrega la suma total de las ventas asociadas a un empleado en particular
- Dependiendo de la consulta, se debe tener cuidado al agrupar por valores que sean diferenciadores (e.g. "id_empleado" y no "nombre")

SQL
Consultas sobre Consultas

- Existen alternativas de hacer consultas sobre consultas.
  - Una opción es crear una tabla en base a una determinada consulta y posteriormente hacer una consulta sobre esa tabla. Esto es una mala práctica, ya que se van agregando nuevas tablas al sistema no consideradas en el modelamiento original, y hay que tener especial cuidado en eliminarlas después de utilizarlas.
  - Dependiendo del sistema de administración de bases de datos, se pueden hacer vistas temporales sobre una consulta y después utilizarlas como una tabla dentro de otras consultas. Estas son descartadas por el sistema al momento que el usuario termina su sesión.

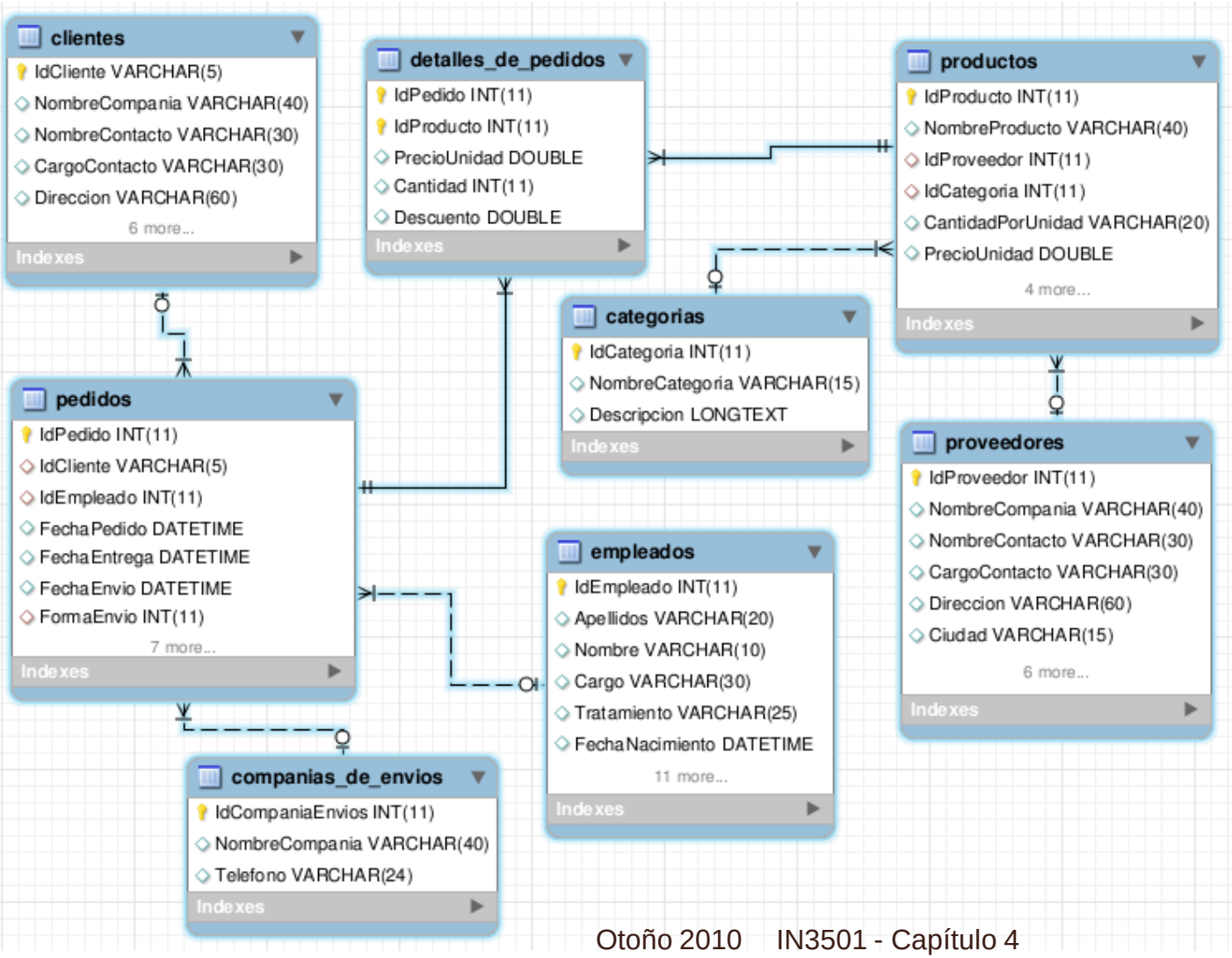
SQL
Consultas sobre Consultas (2)

- Otra alternativa son las **consultas anidadas.** En este caso es posible hacer consultas sobre consultas pero ejecutando de forma secuencial las consultas necesarias.

```
SELECT D.nombre_depto
FROM depto D, empleados E
WHERE cantidad = Nempleos AND Nempleos IN
        (SELECT Nempleos
        FROM Empleados
        GROUP BY id_depto
        HAVING COUNT(*) > 100)
```

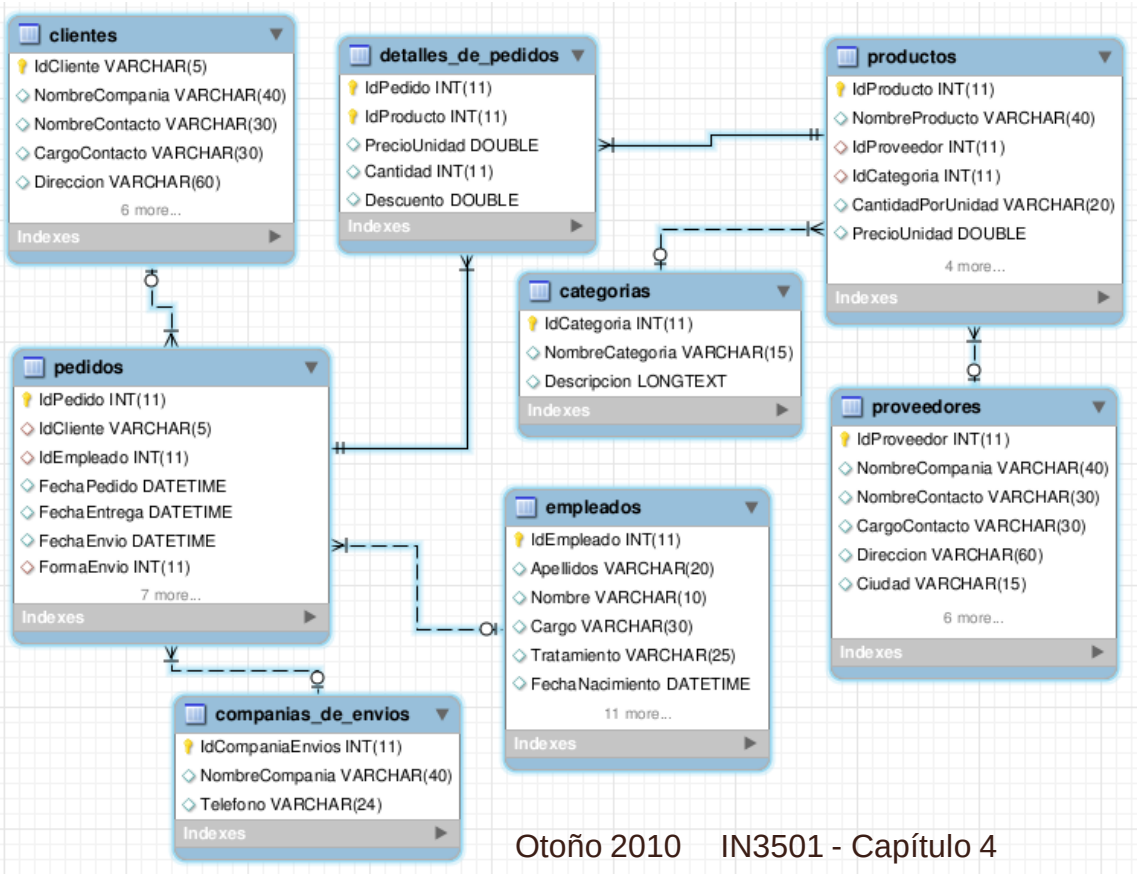
Consultas SQL

- Dado el siguiente diagrama ER



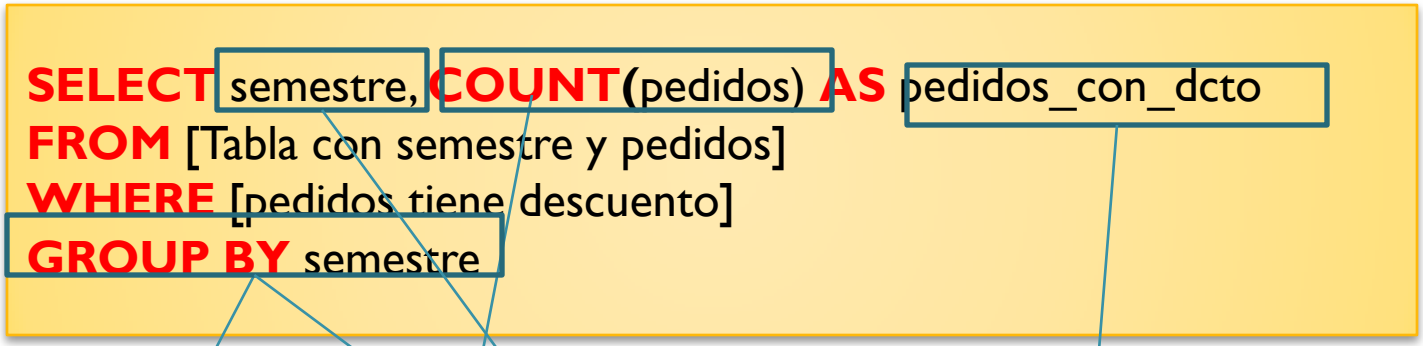
Consultas SQL
Ejemplo

- Determinar la cantidad de pedidos con descuento, agrupados por el semestre de cada año.



Consultas SQL
Paso 1: Crear consulta preliminar

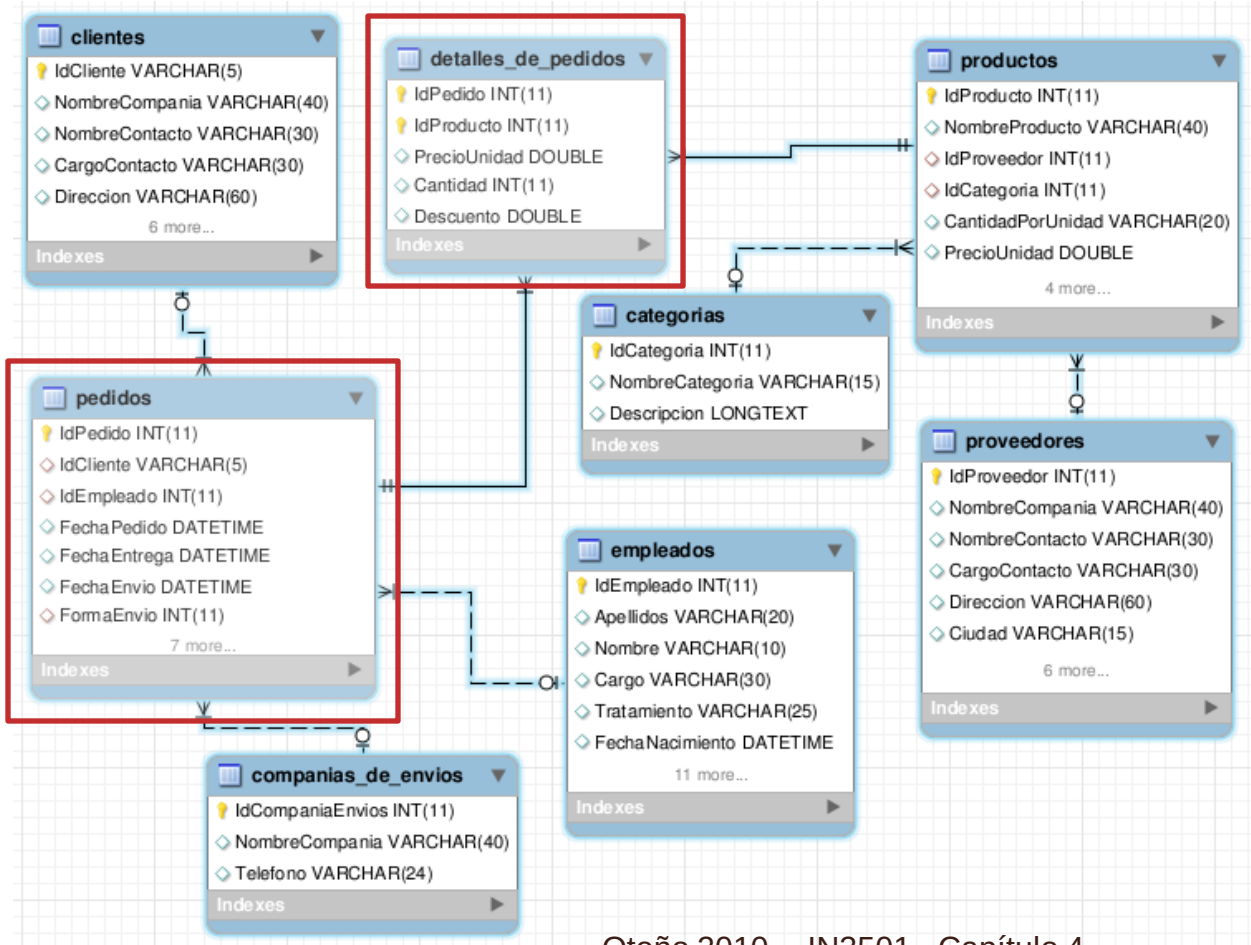
- Del texto inicial, se puede crear una consulta preliminar que representa a grandes rasgos los solicitado:



- Determinar la **cantidad** de **pedidos con descuento**, **agrupados** por el **semestre de cada año**.

Consultas SQL
Paso 2: Identificar tablas

- Identificar las tablas **relevantes para la consulta**



Consultas SQL
Paso 3: Actualizar Consulta

- Una vez identificadas las tablas, se puede actualizar la consulta, identificando los atributos solicitados en **SELECT**, cambiando el nombre las tablas en **FROM** y actualizando los requerimientos en **WHERE**. En ciertos casos, con lo anterior se puede terminar fácilmente una consulta.
- Se puede dar el caso que las tablas identificadas sí contengan la información necesaria, pero **no permitan concretar la consulta**. En ese caso, es necesario **crear una nueva tabla** con la información requerida.
- En el presente caso no son suficientes las tablas relacionadas, por lo que crearemos una nueva tabla.

Consultas SQL
Paso 4: Crear nuevas tablas

- Dado que es necesario, vamos a crear una nueva tabla que contenga todos aquellos **pedidos con descuento** junto **con su respectivo semestre**.

```
SELECT detalles_de_pedidos.IdPedido AS IdPedido ,
    CONCAT( IF( Month(pedidos.FechaPedido) <= 6 , "01-", "02-"),
    YEAR(pedidos.FechaPedido)) AS Semestre
FROM pedidos, detalles_de_pedidos
WHERE detalles_de_pedidos.IdPedido = pedidos.IdPedido
AND detalles_de_pedidos.Descuento <> 0
```

- Esta tabla nos permitirá agrupar los pedidos por semestre. Notar que utilizamos las **tablas identificadas anteriormente**.

Consultas SQL
Paso 5: Integrar nuevas tablas

- Una vez creada la tabla auxiliar, podemos actualizar la consulta anidando la consulta como una nueva tabla llamada "**semestres**":

```
SELECT semestre, COUNT(pedidos) AS pedidos_con_dcto
FROM (SELECT detalles_de_pedidos.IdPedido AS IdPedido ,
        CONCAT( IF( Month(pedidos.FechaPedido) <= 6 , "01-", "02-"),
      YEAR(pedidos.FechaPedido)) AS semestre
      FROM pedidos, detalles_de_pedidos
      WHERE detalles_de_pedidos.IdPedido = pedidos.IdPedido
      AND detalles_de_pedidos.Descuento <> 0) AS semestres
WHERE [pedidos tiene descuento]
GROUP BY semestre
```

Consultas SQL
Paso 5: Integrar nuevas tablas (2)

- Dado que en la sub consulta "semestres" ya tenemos considerada la condición que sean pedidos con descuento, podemos **eliminar la condición [pedidos tienen descuento]**

```
SELECT semestre, COUNT(pedidos) AS pedidos_con_dcto
FROM (SELECT detalles_de_pedidos.IdPedido AS IdPedido ,
        CONCAT( IF( Month(pedidos.FechaPedido) <= 6 , "01-", "02-"),
        YEAR(pedidos.FechaPedido)) AS semestre
        FROM pedidos, detalles_de_pedidos
        WHERE detalles_de_pedidos.IdPedido = pedidos.IdPedido
        AND detalles_de_pedidos.Descuento <> 0) AS semestres
~~WHERE [pedidos tiene descuento]~~
GROUP BY semestre
```

Consultas SQL
Paso 5: Integrar nuevas tablas (3)

- Además, es necesario **ajustar los atributos** y verificar que **no se cuenten más pedidos de los necesarios**.

```
SELECT semestres.semestre,
        COUNT( DISTINCT semestres.IdPedido) AS pedidos_dcto
FROM  (SELECT detalles_de_pedidos.IdPedido AS IdPedido ,
        CONCAT( IF( Month(pedidos.FechaPedido) <= 6 , "01-", "02-"),
        YEAR(pedidos.FechaPedido)) AS semestre
        FROM pedidos, detalles_de_pedidos
        WHERE detalles_de_pedidos.IdPedido = pedidos.IdPedido
        AND detalles_de_pedidos.Descuento <> 0) AS semestres
GROUP BY semestres.semestre
```

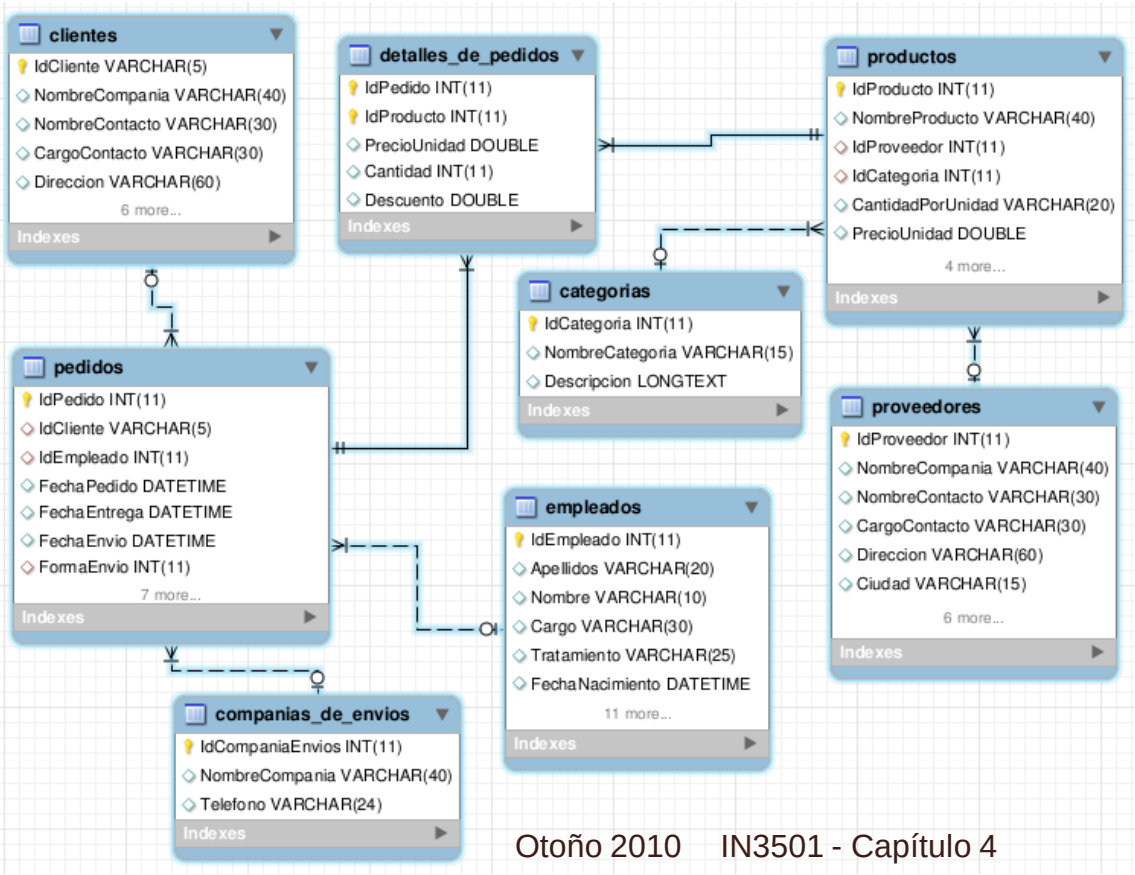
Consultas SQL
Resultado final

```
SELECT semestres.semestre,
            COUNT( DISTINCT semestres.IdPedido) AS pedidos
FROM  (SELECT detalles_de_pedidos.IdPedido AS IdPedido ,
             CONCAT( IF( Month(pedidos.FechaPedido) <= 6 , "01-", "02-"),
            YEAR(pedidos.FechaPedido)) AS semestre
            FROM pedidos, detalles_de_pedidos
            WHERE detalles_de_pedidos.IdPedido = pedidos.IdPedido
            AND detalles_de_pedidos.Descuento <> 0) AS semestres
GROUP BY semestres.semestre
```

Semestre	pedidos
01-1997	92
01-1998	117
02-1996	66
02-1997	105

Consultas SQL
Ejercicio

- Determinar los pedidos cuyo valor sea mayor o igual a 4.5 veces el valor medio de pedidos.



Consultas SQL Solución

```
SELECT detalles_de_pedidos.IdPedido AS IdPedido, 4.5*promedio.Media,
            monto_ventas.valorPedido
FROM
   (SELECT AVG(pedidos_tmp.SUMA) AS Media
    FROM
       (SELECT SUM(detalles_de_pedidos.PrecioUnidad*
                          detalles_de_pedidos.Cantidad*
                          (1-detalles_de_pedidos.Descuento)) AS SUMA
        FROM detalles_de_pedidos
        GROUP BY detalles_de_pedidos.IdPedido) AS pedidos_tmp
   ) AS promedio,
   (SELECT detalles_de_pedidos.IdPedido,
              SUM( detalles_de_pedidos.PrecioUnidad*
              detalles_de_pedidos.Cantidad*
              (1-detalles_de_pedidos.Descuento)) AS valorPedido
    FROM detalles_de_pedidos, pedidos
    WHERE detalles_de_pedidos.IdPedido = pedidos.IdPedido
    GROUP BY detalles_de_pedidos.IdPedido) AS monto_ventas,
    detalles_de_pedidos
WHERE monto_ventas.valorPedido >= 4.5*promedio.Media
AND detalles_de_pedidos.IdPedido = monto_ventas.IdPedido
GROUP BY detalles_de_pedidos.IdPedido
```

Consultas SQL Solución

```
SELECT detalles_de_pedidos.IdPedido AS IdPedido, 4.5*promedio.Media,
       monto_ventas.valorPedido
FROM
  (SELECT AVG(pedidos_tmp.SUMA) AS Media
   FROM
     (SELECT SUM(detalles_de_pedidos.PrecioUnidad*
                 detalles_de_pedidos.Cantidad*
                 (1-detalles_de_pedidos.Descuento)) AS SUMA
      FROM detalles_de_pedidos
      GROUP BY detalles_de_pedidos.IdPedido) AS pedidos_tmp
  ) AS promedio,
  (SELECT detalles_de_pedidos.IdPedido,
          SUM( detalles_de_pedidos.PrecioUnidad*
          detalles_de_pedidos.Cantidad*
          (1-detalles_de_pedidos.Descuento)) AS valorPedido
   FROM detalles_de_pedidos, pedidos
   WHERE detalles_de_pedidos.IdPedido = pedidos.IdPedido
   GROUP BY detalles_de_pedidos.IdPedido) AS monto_ventas,
   detalles_de_pedidos
WHERE monto_ventas.valorPedido >= 4.5*promedio.Media
AND detalles_de_pedidos.IdPedido = monto_ventas.IdPedido
GROUP BY detalles_de_pedidos.IdPedido
```

IdPedido	4.5*promedio.Media	valorPedido
10353	6862.79392746056	8593.27996798754
10360	6862.79392746056	7390.2
10372	6862.79392746056	9210.9
10417	6862.79392746056	11188.4
10424	6862.79392746056	9194.5599657476
10479	6862.79392746056	10495.6
10514	6862.79392746056	8623.45
10515	6862.79392746056	9921.29997348786
10540	6862.79392746056	10191.7
10691	6862.79392746056	10164.8
10816	6862.79392746056	8446.44999337569
10817	6862.79392746056	10952.8449786276
10865	6862.79392746056	16387.4999871477
10889	6862.79392746056	11380
10897	6862.79392746056	10835.24
10981	6862.79392746056	15810
11030	6862.79392746056	12615.05
11032	6862.79392746056	8902.5

SQL
Orientado a la administración

- Existe una serie de comandos utilizados para la administración de las bases de datos.
- Los comandos CREATE, DROP, DELETE, UPDATE, INSERT, entre otros son aquellos que permiten crear, eliminar, actualizar e insertar nuevos objetos en una base de datos.
- A continuación veremos brevemente el uso de algunos de estos comandos. Para más información se recomienda revisar el manual de referencias de MySQL: http://dev.mysql.com/doc/#manual

SQL
Insertar nuevos datos

- Dada una tabla, mediante el uso del comando **INSERT** es posible insertar un nuevo dato en ella.

```
INSERT INTO nombre_tabla [ (lista_de_columnas)]
VALUES (lista_de_valores)
```

- Un ejemplo de uso es el siguiente:

```
INSERT INTO staff (sno, fname, lname, position, salary, bno)
VALUES ('SG44', 'Anne', 'Jones', 'Assistant',8100, 'B3');
```

SQL
Actualizar datos

- Dada una tabla, mediante el uso del comando **UPDATE** es posible actualizar los datos.

```
UPDATE nombre_tabla
SET nombre_columna1 = valor1
      [, nombre_columna2 = valor2...]
[WHERE condicion_busqueda]
```

- Si no se específica WHERE, se actualiza toda la tabla

```
UPDATE staff
SET salario = salario*1.03;
```

```
UPDATE staff
SET salario = salario*1.05
WHERE cargo= 'Gerente';
```

SQL
Eliminación de datos

- Dada una tabla, mediante el uso del comando **DELETE** es posible eliminar datos.

```
DELETE FROM nombre_tabla
[WHERE condicion_busqueda]
```

- Si no se específica WHERE, se eliminan todos los registros de una tabla.

```
DELETE FROM staff;
```

```
DELETE FROM staff
WHERE cargo= 'CIO';
```

SQL
Creación de tabla

- Con el comando CREATE, se puede crear una tabla.

```
CREATE TABLE nombre_tabla
(nombre_columna  tipo_dato [NULL | NOT NULL] [,...])
```

- Además, el comando CREATE se puede utilizar para crear una nueva base de datos:

```
CREATE DATABASE in3501;
```

```
CREATE TABLE tabla_in3501
( id  INT NOT NULL,
  nombre VARCHAR(15) NOT NULL,
  apellido VARCHAR(15) NOTNULL)
```

SQL
Eliminación de tablas

- Con el comando DROP, se puede eliminar una tabla.

```
DROP TABLE nombre_tabla [RESTRICT | CASCADE]
```

- RESTRICT no permite que la tabla sea eliminada en caso que otras tablas dependan de ella.
- CASCADE elimina en cascada todas aquellas tablas o elementos que dependen de la tabla en cuestión.

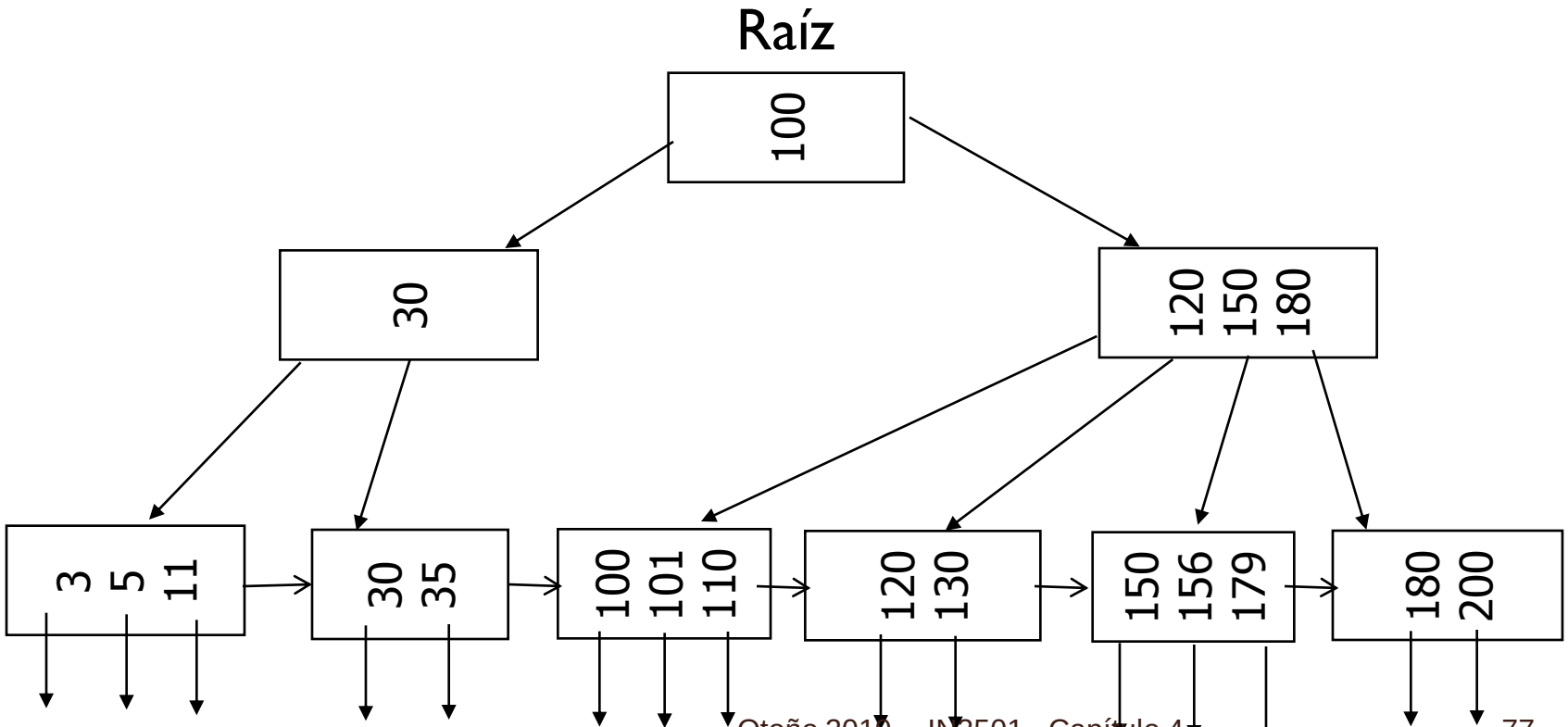
```
DROP TABLE tabla_in3501;
```

Indexación

- La tecnología que almacena los datos debe ser capaz de soportar fácil indexación.
- Existen varios tipos de índices (primarios, secundarios, estáticos, dinámicos)
- La utilidad de los índices se pueden asociar a la necesidad de tener índices de términos en un libro.
  - Al momento que un lector quiera llegar rápidamente a la página donde se tiene más información sobre algún término, se busca primero en el índice del libro.
- Dependiendo del tipo de información que se desee buscar, se pueden utilizar distintos tipos de índices.
- En lo que sigue, veremos brevemente los índices de **Hash** y **árboles B+**.

Indexación
Árboles B+

- Estructura del **árbol B+** permite indexar la información dinámicamente, dependiendo del rango al cuál pertenezca el valor a agregar.



Indexación
Árboles B+ (2)

- Propiedades estructurales del **árbol B+** permite modificar **dinámicamente** las relaciones entre los nodos, agregando nuevas estructuras a medida que nuevos objetos se van agregando.
- Estas propiedades facilitan la indexación de valores numéricos que serán buscados en rangos **(e.g. ¿Cuáles son los sueldos mayores a 500.000)**, o variables de caracteres, o fechas que serán buscadas en rangos **(e.g. ¿Cuáles son los productores cuyas ventas fueron las últimas 2 semanas?)**
- No son buena alternativa cuando se desean realizar búsquedas para valores puntuales.

Indexación
Índices de Hash

- Es un tipo de índice ampliamente conocido por ser **estático**, e ideal para la indexación de valores puntuales **(e.g. ¿Cuáles son los atributos asociados al producto cuyo ID es 12387gb?)**.
- Un índice de **Hash** organiza llaves de búsqueda con sus respectivos punteros a un conjunto de casillas.
- Una función de **Hash** es la que permite asociar los valores de las llaves de búsqueda al conjunto de casillas donde se almacenan los datos.
- Actualmente existen variantes de los índices de **Hash** estáticos que permiten modificar dinámicamente las casillas utilizadas.