2.3. Transformación al modelo relacional: generalidades

- El objetivo principal de esta fase es transformar el modelo conceptual en un **modelo lógico de datos** que represente los **requisitos de datos** de la organización
- En el diseño lógico se deben tener en cuenta aspectos como:
  - eliminar las posibles redundancias
  - buscar la máxima simplicidad y claridad
  - evitar módulos adicionales de programación
  - buscar un equilibrio entre las necesidades del usuario o aplicación y la eficiencia

- Para conseguir una mayor **portabilidad**, es conveniente utilizar las características propias del SGBD lo más tarde posible

- El diseño lógico se puede dividir en dos etapas:
  - **Diseño lógico estándar**. Partiendo del esquema conceptual (en nuestro caso el modelo E-R extendido), se elabora un esquema lógico estándar basado en el modelo de datos que soporte el SGBD elegido (generalmente, el modelo relacional)
  - **Diseño lógico específico**. Una vez obtenido el diseño lógico estándar, se transforma al lenguaje de definición de datos soportado por el SGBD seleccionado (ORACLE, DB2, MySQL, SYBASE, ACCESS, INFORMIX, etc.)

Existen herramientas CASE (Ingeniería del Software Asistida por Ordenador) que permiten crear gráficamente los diagramas E-R (o variaciones de ellos), y luego lo transforman en un esquema de bases de datos relacional basado en el LDD de un SGBD relacional específico

- La transformación de un diagrama del modelo E-R al modelo relacional está basado en los **tres principios** fundamentales:

1. Toda entidad se transforma en una relación (tabla)

2. Toda relación con cardinalidad máxima muchos a muchos (M:N) se transforma en una relación (tabla)

3. Toda relación con cardinalidad máxima uno a muchos (1:N) se puede transformar de dos formas:

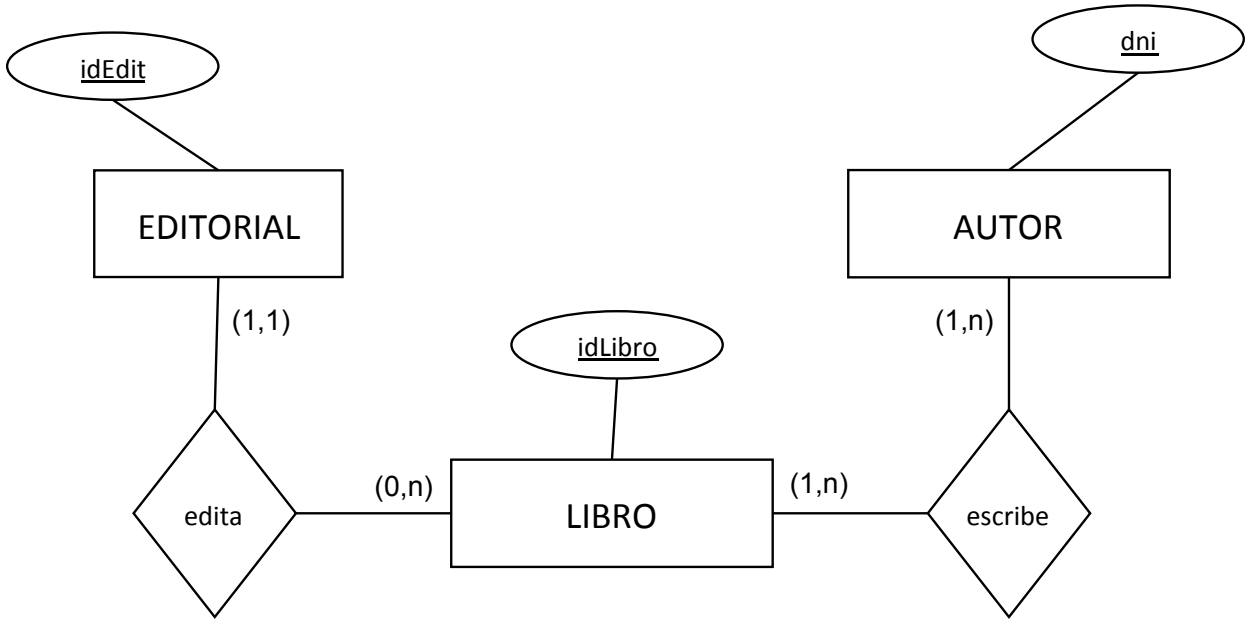
   - mediante propagación de clave
   - mediante transformación en una nueva relación (tabla)

- En el proceso de transformación del esquema conceptual al esquema lógico se pierde semántica, ya que el modelo relacional no distingue entre entidades y relaciones entre entidades. En este modelo sólo se utiliza el concepto de **tabla**

- Por su propia naturaleza, el modelo E-R **recoge más semántica** que el modelo relacional

- Para solucionar esto y poder mantener las reglas de negocio en la capa de los datos, se hace uso de los **objetos** que proporciona el modelo relacional específico (disparadores, procedimientos almacenados, etc.)

Ejemplo de transformación



LIBRO (idLibro, título, idioma, …, editorial)
CP: idLibro
CAj: editorial → EDITORIAL(idEdit)
VNN: título, idioma, editorial

EDITORIAL (idEdit, dirección, ciudad, país)
CP: idEdit
VNN: dirección, ciudad

AUTOR (dni, nombre, nacionalidad, fecha_nac, …)
CP: dni
VNN: nombre

ESCRIBE (libro, autor)
CP: (libro, autor)
CAj: libro → LIBRO(idLibro)
autor → AUTOR(dni)

2.3.1. Transformación de entidades

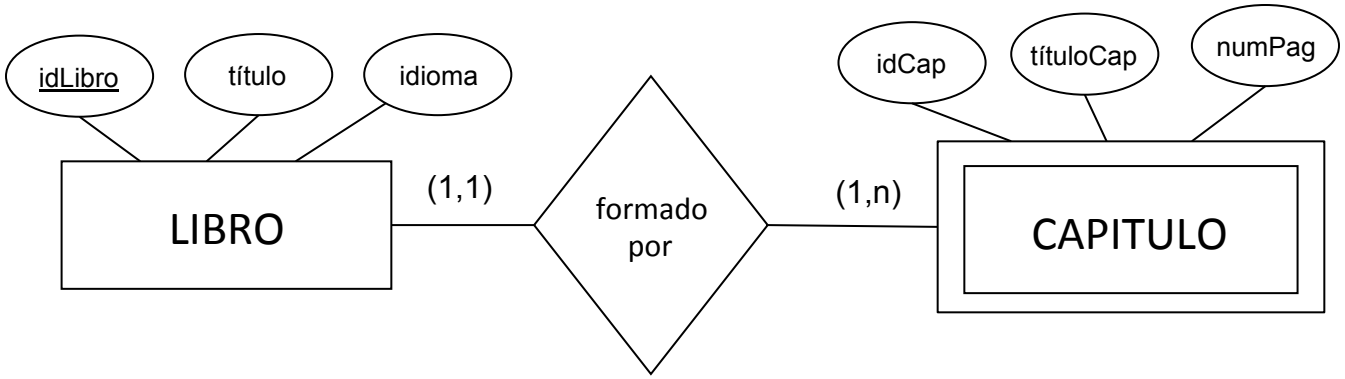
- Cada entidad (regular o débil) se transforma en una tabla
- Por convención, la tabla se nombrará igual que la entidad de la que proviene

Transformación de atributos de entidades

- Cada atributo de una entidad se transforma en un atributo (columna) de la tabla a la que ha dado lugar la entidad

- Se tendrán en cuenta las siguientes características de los atributos:
  - El (o los) atributo(s) clave(s) de una entidad pasan a ser la clave primaria de la tabla
  - En las **entidades débiles**, la clave primaría de su tabla estará formada por la concatenación de los atributos de la clave de la entidad regular de la que depende y alguno o varios de sus propios atributos
  - El resto de atributos pasan a ser columnas de la tabla, teniendo en cuenta que a los **atributos obligatorios** se le debe aplicar la restricción **NOT NULL** y a las **claves alternativas** la restricción **UNIQUE**
  - Por razones de eficiencia, los atributos derivados pueden transformarse en columnas o simplemente calcularse cuando se necesiten
  - ¿Atributos multivaluados?

Ejemplo de transformación de entidad fuerte y débil



LIBRO (idLibro, título, idioma, …)
CP: idLibro
VNN: título, idioma

CAPITULO (idCap, idLibro, tituloCap, numPag, …)
CP: (idLibro, idCap)
VNN: tituloCap, numPag
CAj: idLibro → LIBRO

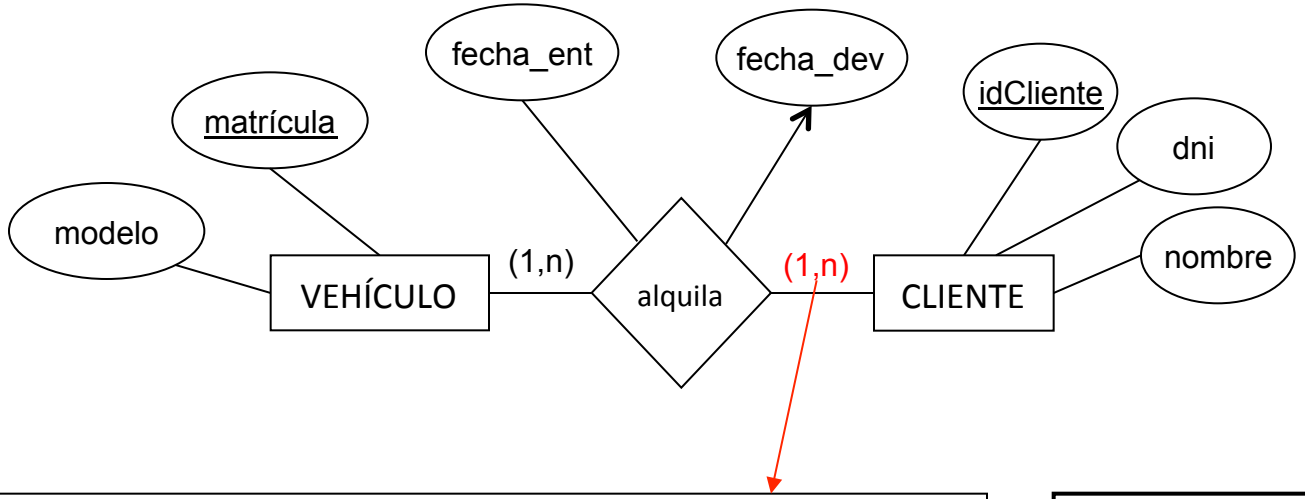
(en el modelo específico se debe incluir la cláusula **ON DELETE CASCADE**)

2.3.2. Transformación de relaciones

Relaciones Binarias

- **Relaciones "muchos a muchos"**
  - En el caso más general, una relación M:N se transforma en una tabla cuya **clave primaria** es la **concatenación de las claves primarias** de las tablas que se generan al transformar las entidades que forman parte de la relación. Junto a estos atributos se incluyen los atributos propios de la relación
  - Cada uno de los atributos que forman la clave primaria de esta tabla son **claves ajena** (FOREIGN KEY) que referencian a cada una de las tablas donde dicho atributo es clave primaria
  - En una transformación de una relación muchos a muchos, a veces resulta necesario incluir, en la clave primaria, algún **atributo propio** de la relación. Generalmente ocurre cuando se puede repetir la relación entre las dos mismas tuplas y/o deseamos mantener un histórico a lo largo del tiempo

Ejemplo



VEHÍCULO (matrícula, modelo, …)
CP: matrícula
VNN: modelo

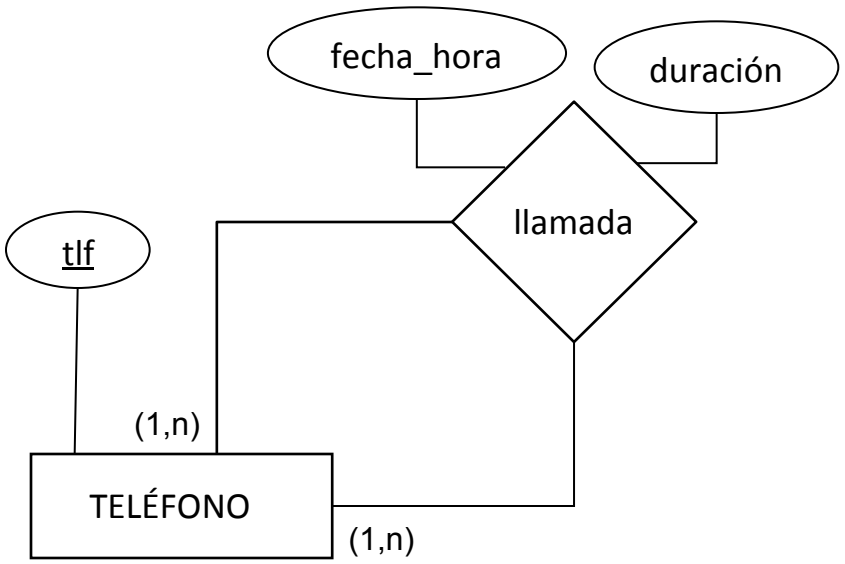
CLIENTE (idCliente, dni, nombre, …)
CP: idCliente
Único: dni
VNN: dni, nombre

ALQUILA (matrícula, idCliente, fecha_ent, fecha_dev, ...)
CP: (matrícula, idCliente, fecha_ent)
CAj: matrícula → VEHÍCULO
idCliente → CLIENTE

La transformación es la misma.
Hay, por tanto, pérdida de semántica al pasar del modelo conceptual al lógico

- Hay situaciones en **las que no se cumple** la regla general. Dependerá de las "*reglas de negocio*" del problema que estemos resolviendo

Ejemplo



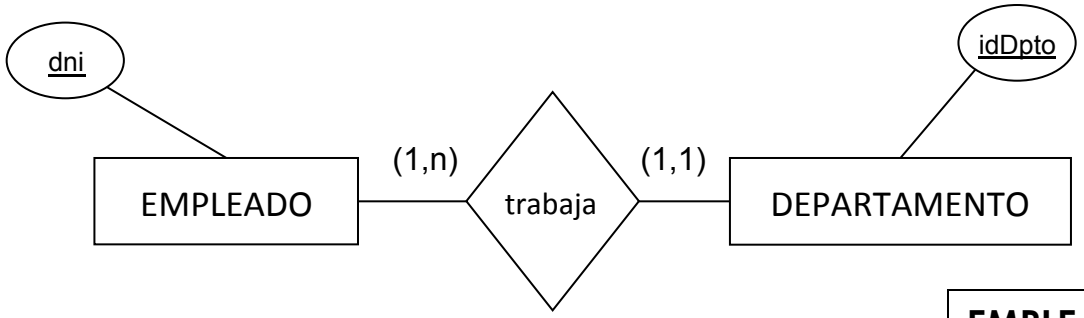
LLAMADA (tlf_o, tlf_d, fecha_hora, duración)
CP: (tlf_o, fecha_hora)
Único: (tlf_d, fecha_hora)
VNN: duración

- la clave principal no puede estar formada únicamente por los teléfonos origen y destino puesto que impediría que, desde un teléfono, se pudiera llamar al mismo teléfono más de una vez
- añadir la fecha a la clave solucionaría ese problema
- sin embargo, esa solución permitiría que desde un teléfono se pudiera llamar a varios teléfonos a la misma vez (sólo se repetiría la combinación de "tlf_o, fecha_hora"). Esa situación no se puede dar en este problema
- en este caso hay un subconjunto de atributos que también identificaría cada tupla de la relación. Por tanto, esa será la clave principal

- **Relaciones "uno a muchos"**
  - Para transformar este tipo de relaciones existen **2 posibles soluciones**:

    1. Propagar el/los atributo/s que se ha/n elegido como **clave primaria** de la entidad que tiene cardinalidad máxima 1 a la que tiene máxima M, no creándose una tabla adicional con la relación. La tabla con cardinalidad máxima M tendrá, en este caso, uno o varios atributos que hacen de **clave ajena**

- Ejemplo



EMPLEADO (dni, …, departamento)
CP: dni
CAj: departamento → DEPARTAMENTO(idDpto)
VNN: departamento

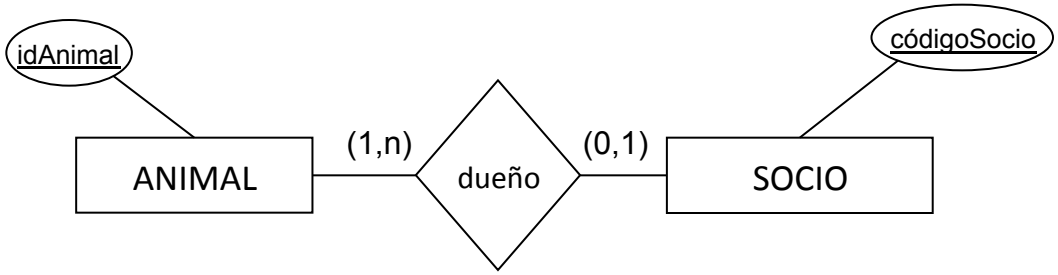
DEPARTAMENTO (idDpto, …)
CP: idDpto

2. Transformar la relación **en una tabla** como si se tratara de una relación "muchos a muchos". En este caso, la clave primaria de la tabla estará formada por la clave primaria de la tabla a la que corresponde la cardinalidad M

– Veamos los casos en los que resulta conveniente adoptar esta opción:

A. Cuando el número de ocurrencias relacionadas de la entidad que propaga su clave es muy pequeño. Esto producirá muchos valores nulos, lo cual, generalmente, no es conveniente en el modelo relacional

Ejemplo



SOLUCIÓN A

ANIMAL (idAnimal, nombre, raza, …, dueño)
CP: idAnimal
CAj: dueño → SOCIO (códigoSocio)
VNN: nombre

SOCIO (códigoSocio, dni, …)
CP: códigoSocio
Único: dni

SOLUCIÓN B

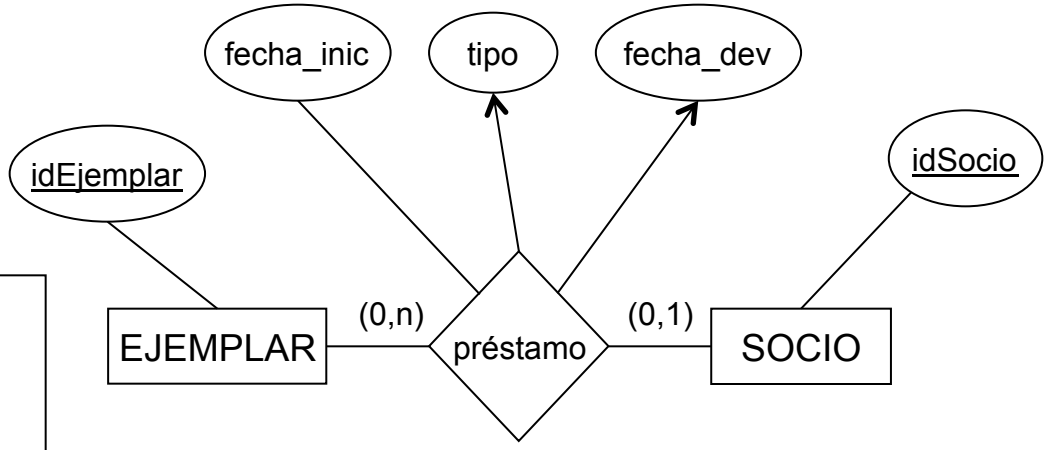
ANIMAL (idAnimal, nombre, raza, …)
CP: idAnimal
VNN: nombre

SOCIO (códigoSocio, dni, …)
CP: códigoSocio
Único: dni

DUEÑO (idAnimal, códigoSocio)
CP: idAnimal
CAj: idAnimal → ANIMAL
códigoSocio → SOCIO
VNN: códigoSocio

B. Cuando la relación tiene atributos propios y no deseamos propagarlos para no perder semántica

Ejemplo



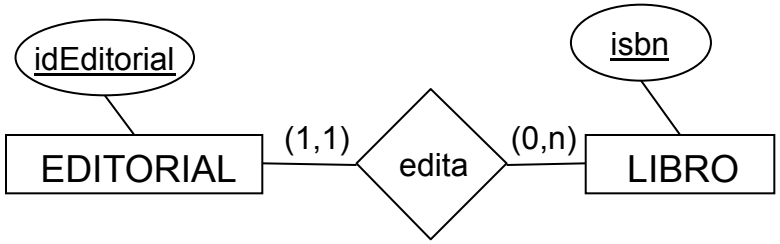
EJEMPLAR (idEjemplar, título, …)
CP: idEjemplar
VNN: título

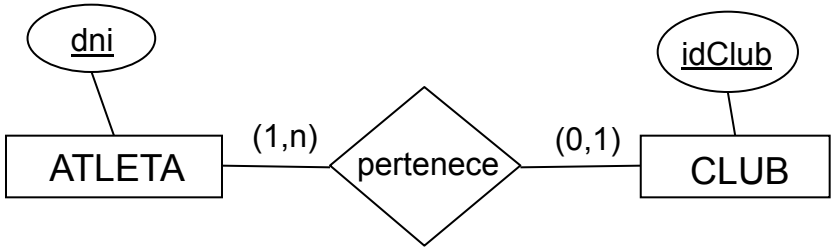
SOCIO (idSocio, dni, nombre, …)
CP: idSocio
Único: dni
VNN: dni, nombre

PRESTAMO (idEjemplar, idSocio, fecha_inic, fecha_dev, tipo)
CP: idEjemplar
CAj: idEjemplar → EJEMPLAR
idSocio → SOCIO
VNN: idSocio, fecha_inic

- Cuando se transforma una relación hay que prestar especial atención en las **cardinalidades mínimas**, ya que nos proporcionan una semántica importante a la hora de asignar algunas restricciones a los valores de los atributos, principalmente, la permisión o no de **valores nulos**

Ejemplos





EDITORIAL (idEditorial, dirección, …)
CP: idEditorial

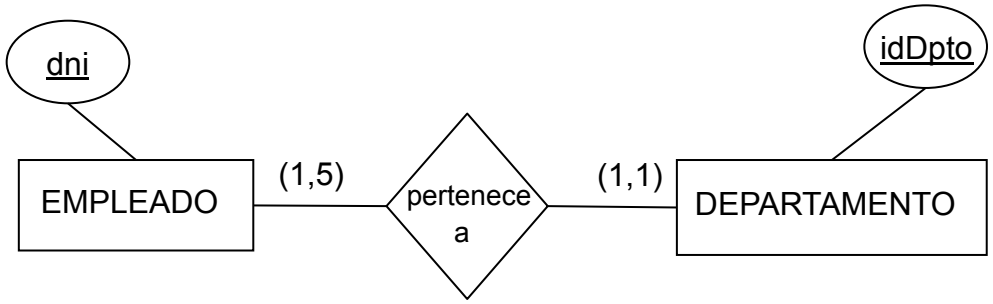
LIBRO (isbn, título, …, editorial)
CP: isbn
CAj: editorial → EDITORIAL(idEditorial)
VNN: editorial

CLUB (idClub, nombre, dirección, …)
CP: idClub

ATLETA (dni, nombre, …, club)
CP: dni
CAj: club → CLUB(idClub)

- Otra característica que se debe recoger en la transformación de las relaciones es la cardinalidad mínima o máxima cuando ésta es un número distinto de 0 y 1 (mínima) o 1 y n (máxima). Generalmente se controlan con **aserciones** (no las implementa Oracle) o **disparadores**

- Ejemplo



EMPLEADO (dni, nombre, …, departamento)
CP: dni
CAj: departamento → DEPARTAMENTO(idDpto)
VNN: departamento

DEPARTAMENTO (idDpto, nombre, …)
CP: idDpto

A tener en cuenta

- Cuando se desarrolla una aplicación que interactúa con un SGBD, hay que decidir en qué capa se van a controlar o testear las **restricciones** o **reglas de negocio**
- Es recomendable controlar las restricciones **a nivel de base de datos**
- De esa forma, el código será mucho más sencillo y **portable**

Volviendo al ejemplo anterior

```
public void insertaEmpleado (Empleado empleado) throws SQLException  {
    Connection accesoBD = conexion.getConexion();

    ps = accesoBD.prepareStatement("INSERT INTO EMPLEADO VALUES (?, ?, ?)");

    ps.setString(1, empleado.getDni());
    ps.setString(2, empleado.getNombre());
    ps.setString(3, empleado.getDepartamento());
    ps.executeUpdate();
    System.out.println("Empleado insertado correctamente");

    ps.close();
    conexion.desconexion();
}
```

El método **insertaEmpleado()** pertenece a la clase **ManejaEmpleado**, que pertenece al modelo (MVC) o capa de persistencia

Opción 1 (menos recomendable)

- Realizar el control en las capas más externas (aplicación)

```
public void insertaEmp1() {
      int numEmpleados = 0;
      String dpto = "D001";
      Empleado emp = new Empleado("99999999Z", "Saúl", dpto);
      try {
         ManejaEmpleado E = new ManejaEmpleado();
         numEmpleados = E.NumEmpleados(dpto);
         if (numEmpleados == 5) {
            System.out.println("No se puede insertar. Ya hay " +
                  numEmpleados + " empleados en el departamento " + dpto);
         }
         else E.insertaEmpleado(emp);

      } catch (SQLException se) {
         System.out.println(se.getMessage());
     }
}
```

Opción 2 (más recomendable)

- Realizar el control a nivel de base de datos

```
CREATE OR REPLACE TRIGGER maximoEmpleados
  BEFORE INSERT ON EMPLEADO
  FOR EACH ROW
DECLARE
  numEmpleados INTEGER;
BEGIN
  SELECT COUNT(*) INTO numEmpleados FROM EMPLEADO
         WHERE departamento = :new.departamento;
  IF numEmpleados == 5  THEN RAISE_APPLICATION_ERROR (-20001, 'El departamento ' ||
                  :new.departamento ||  'está completo');
  END IF;
END;
```

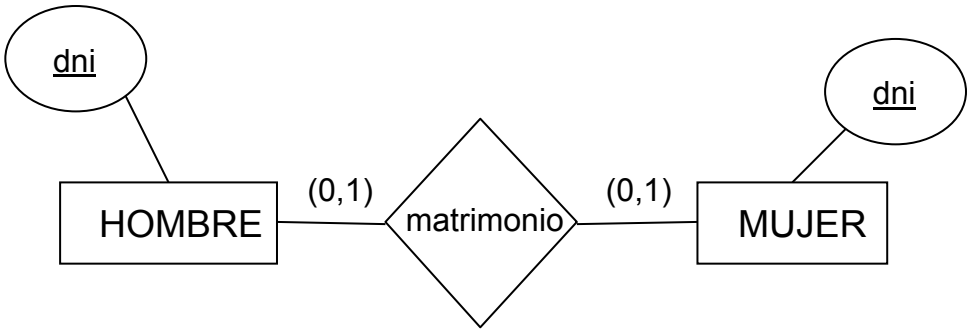
```
public void insertaEmp2() {
      String dpto = "D001";
      Empleado emp = new Empleado("99999999Z", "Saúl", dpto);
      try {
            ManejaEmpleado E = new ManejaEmpleado();
            E.insertaEmpleado(emp);
      }
        catch (SQLException se) {
          System.out.println(se.getMessage());
      }
}
```

- **Relaciones “uno a uno”**
  - En este tipo de relaciones no existe una regla fija para la transformación al modelo relacional
  - En general, podemos elegir entre:
    - **Generar una nueva tabla** como si se tratara de una relación "muchos a muchos" pero teniendo en cuenta que para formar la clave primaria de la nueva tabla basta con considerar sólo la clave primaria de una de las dos entidades
    - **Realizar una propagación de clave** como si fuera una relación "uno a muchos"
  - Los criterios para tomar la decisión de una regla u otra se basan en las cardinalidades mínimas, en recoger la mayor cantidad de semántica posible, en evitar los valores nulos y en conseguir la mayor eficiencia posible

- Veamos los casos más característicos:

A. Si las entidades que participan en la relación tienen cardinalidad (0,1) y se conoce a priori que **no hay mucha relación entre las entidades**, se genera una nueva tabla como si se tratase de una relación "muchos a muchos". El objetivo es **evitar los valores nulos** en caso de que se propagara la clave

Ejemplo



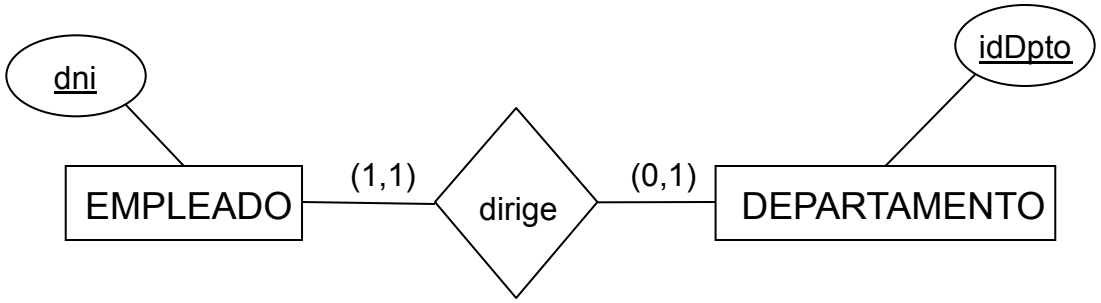
HOMBRE (dni, nombre, …)
CP: dni

MUJER (dni, nombre, …)
CP: dni

MATRIMONIO (dniH, dniM)
CP: dniM
CAj: dniH → HOMBRE(dni)
dniM → MUJER(dni)
VNN: dniH
Único: dniH

B. Si una de las entidades que participa en la relación posee cardinalidad (0,1) mientras que la otra es (1,1), se **propaga la clave** de la entidad con cardinalidad (1,1) a la tabla resultante de la entidad de cardinalidad (0,1). De esta forma **se evitan valores nulos**

Ejemplo

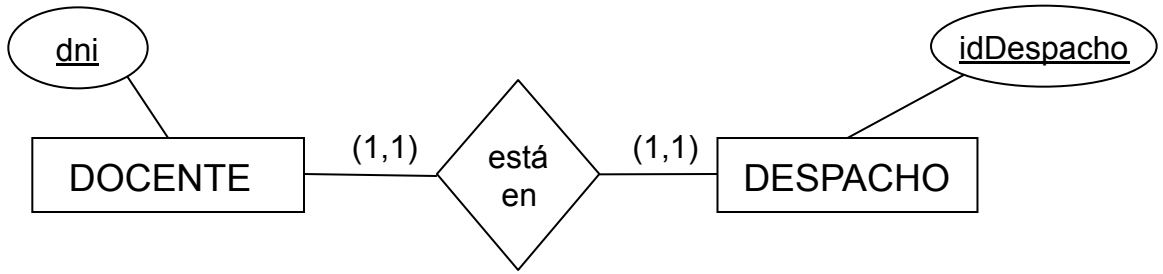


EMPLEADO (dni, nombre, dirección, …)
CP: dni

DEPARTAMENTO (idDpto, lugar, …, director)
CP: idDpto
CAj: director → EMPLEADO(dni)
VNN: director
Único: director

C. En el caso de que ambas entidades posean cardinalidad (1,1), se puede **propagar la clave de cualquiera de ellas** a la tabla resultante de la otra

Ejemplo



DOCENTE (dni, nombre, …)
CP: dni
VNN: nombre

DESPACHO (idDespacho, planta, …, docente)
CP: idDespacho
CAj: docente → DOCENTE(dni)
VNN: planta, docente
Único: docente

DOCENTE (dni, nombre, …, despacho)
CP: dni
CAj: despacho → DESPACHO(idDespacho)
VNN: nombre, despacho
Único: despacho

DESPACHO (idDespacho, planta, …)
CP: idDespacho
VNN: planta

UBICACIÓN (dni, nombre, …, idDespacho, planta, …)
CP: dni
VNN: nombre, planta, idDespacho
Único: idDespacho

En este tipo de relaciones también es posible almacenar toda la información **en una única tabla**

3.3.2. Relaciones Reflexivas

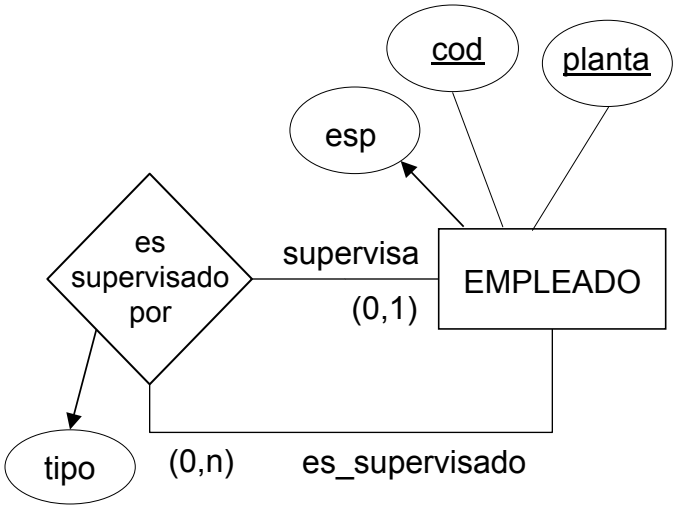
- Para transformar relaciones reflexivas se aplican las mismas reglas que para las relaciones binarias pero con algunas consideraciones, ya que en este caso **sólo hay una entidad** asociada a la relación:

A. En el caso de cardinalidades **"uno a uno"** o **"uno a muchos"**, en los que se realice propagación de clave, la clave principal de la entidad pasa a ser, además, clave ajena que referencia a la propia tabla. Además, siempre que existan, se incluirán los atributos propios de la relación

B. Si en el caso anterior se decide **crear una tabla** para transformar la relación, la clave primaria de la nueva tabla estará formada por la clave primaria de la entidad

- Para almacenar la semántica de la relación se incluirán, de nuevo, los atributos que forman la clave primaria de la entidad (a este conjunto se le debe aplicar la restricción de **valor no nulo**). Ambos conjuntos de atributos, por separado, serán claves ajenas de la entidad. Además, en caso de que existan, se incluirán los atributos propios de la relación

Ejemplo



EMPLEADO (cod, planta, esp, …, codJefe, plantaJefe, tipo)
CP: (cod, planta)
CAj: (codJefe, plantaJefe) → EMPLEADO(cod, planta)

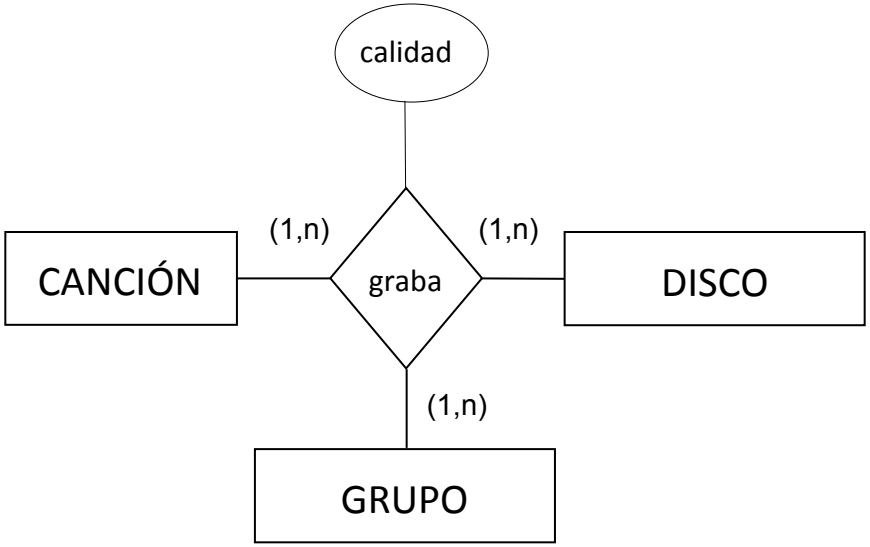
EMPLEADO (cod, planta, esp, …)
CP: (cod, planta)

ES_SUPERVISADO_POR (cod, planta, codJefe, plantaJefe, tipo)
CP: (cod, planta)
CAj: (cod, planta) → EMPLEADO
(codJefe, plantaJefe) → EMPLEADO(cod, planta)
VNN: codJefe, plantaJefe

3.3.3. Relaciones Ternarias

- **Relaciones "muchos a muchos a muchos"**
  - Este tipo de relación se transforma en una tabla cuya clave primaria es la **concatenación de las claves primarias** de las tablas surgidas al transformar las entidades que forman parte de la relación
  - Junto a estos atributos se incluyen los atributos propios de la relación
  - Cada uno de los atributos que forman la clave primaria de esta tabla es **clave ajena** respecto a cada una de las tablas donde dicho atributo es clave primaria

Ejemplo



CANCIÓN (idC, título, duración, …)
CP: idC

GRUPO (idG, nombre, nacionalidad, …)
CP: idG

DISCO (idD, título,...)
CP: idD

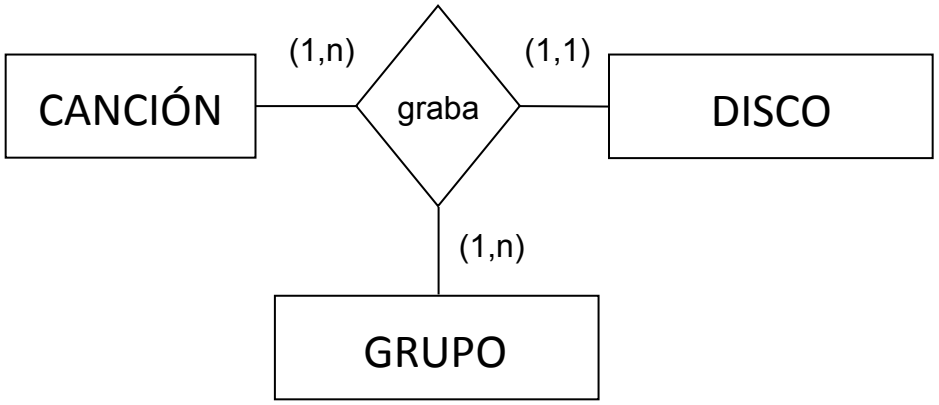
GRABA (idC, idG, idD, calidad)
CP: (idC, idG, idD)
CAj: idC → CANCIÓN
idD → DISCO
idG → GRUPO
VNN: calidad

En el caso de que una canción de un grupo no tuviese que estar obligatoriamente en un disco, ¿sería correcto modelarlo con una relación ternaria?

- **Relaciones "muchos a muchos a uno"**
  - Este tipo de relación se transforma en una tabla cuya clave primaria es la **concatenación de las claves primarias** de las tablas que corresponden a la **cardinalidad M** surgidas al transformar las entidades que forman parte de la relación
  - Junto a estos atributos se incluyen los atributos propios de la relación **más la clave primaria** de la tabla que corresponde a la cardinalidad 1
  - Cada uno de los atributos que forman la clave primaria de esta tabla (y los atributos añadidos de la relación de cardinalidad 1) son **claves ajenas** respecto a cada una de las tablas donde dicho atributo es clave primaria

Ejemplo

- Una canción de un grupo sólo puede estar grabada en un disco
- En un disco, una canción puede haber sido grabada por varios grupos
- Un disco de un grupo puede tener varias canciones



CANCIÓN (idC, título, duración, …)
CP: idC

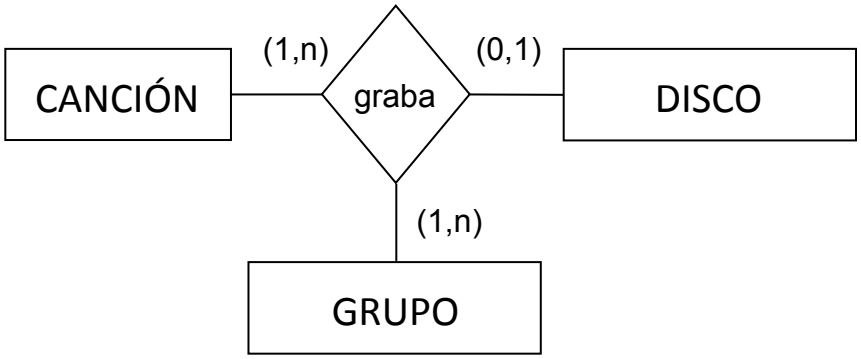
GRUPO (idG, nombre, nacionalidad, …)
CP: idG

DISCO (idD, título,...)
CP: idD

GRABA (idC, idG, idD)
CP: (idC, idG)
CAj: idC → CANCIÓN
idD → DISCO
idG → GRUPO
VNN: idD

A tener en cuenta

- Este diseño está pensado para reflejar **toda** la información de las grabaciones existentes. Resultaría ineficaz si quisiéramos reflejar la lista de canciones grabadas por un grupo, independientemente de si éstas están o no recogidas en un disco. En tal caso podría haber muchos valores en la columna **idD** sin rellenar



CANCIÓN (idC, título, duración, …)
CP: idC

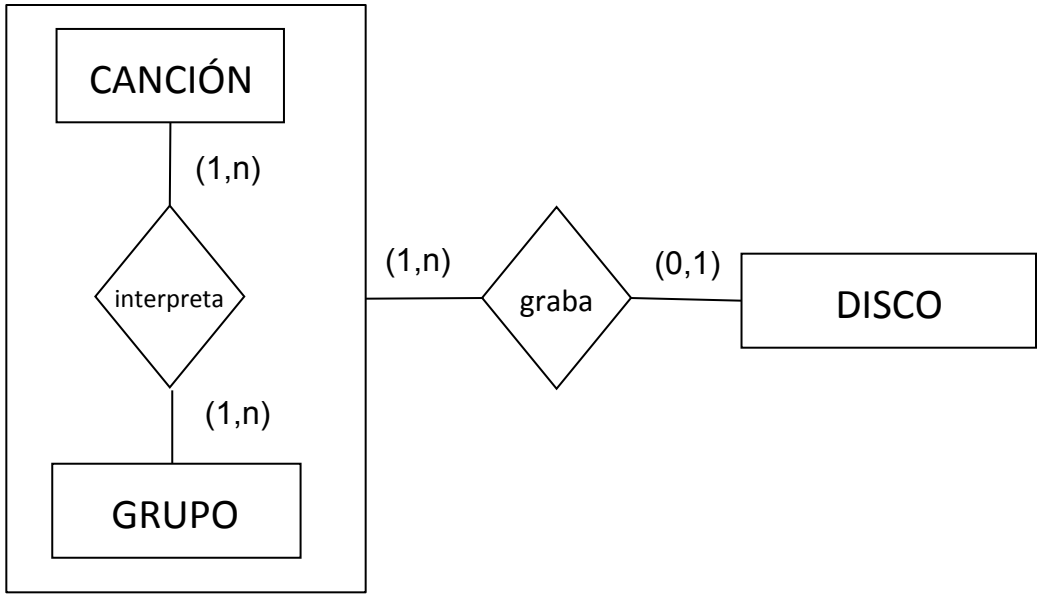
GRUPO (idG, nombre, nacionalidad, …)
CP: idG

DISCO (idD, título,…)
CP: idD

GRABA (idC, idG, idD)
CP: (idC, idG)
CAj: idC → CANCIÓN
idD → DISCO
idG → GRUPO

A tener en cuenta

- Para reflejar esta situación, resulta más eficaz hacer uso de una agregación



CANCIÓN (idC, título, duración, …)
CP: idC

GRUPO (idG, nombre, nacionalidad, …)
CP: idG

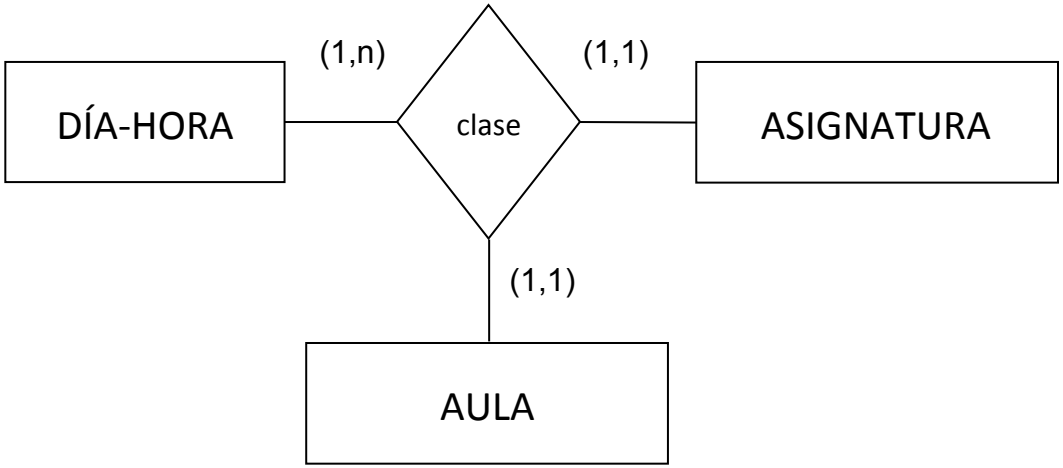
DISCO (idD,título,...)
CP: idD

INTERPRETA (idC, idG)
CP: (idC, idG)
CAj: idC → CANCIÓN
idG → GRUPO

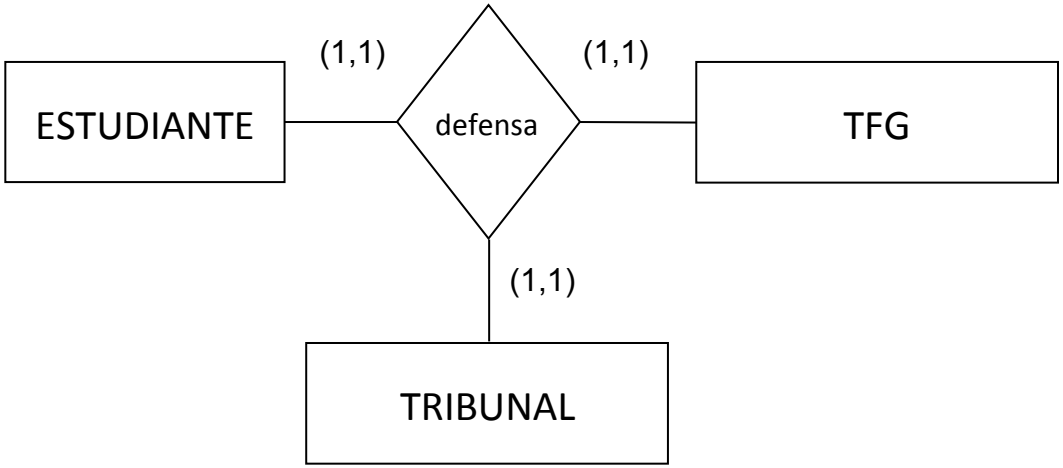
GRABA (idC, idG, idD)
CP: (idC, idG)
CAj: (idC, idG) → INTERPRETA
idD → DISCO
VNN: idD

Nótese que, en este caso, no habrá valores sin rellenar en la columna idD de la tabla GRABA, pues las parejas canción-grupo que no estén en un disco se guardan sólo en INTERPRETA

- ¿Relaciones “muchos a uno a uno”?



- **¿Relaciones "uno a uno a uno"?**



Suposiciones

Un estudiante puede defender más de un trabajo
Si un estudiante defiende más de un trabajo, debe ser con tribunales distintos
Si dos estudiantes hacen el mismo trabajo, lo defenderán en tribunales distintos

- **Ejercicio. Federación Ciclista Internacional**

- Se desea almacenar información de las distintas vueltas (Giro de Italia, Tour de Francia, Vuelta a España, etc.) que se celebrarán en la temporada 2018-19, las cuales tendrán un identificador y queremos guardar información como el nombre, el número de etapas, la longitud en km., y las fechas de inicio y final

- Las etapas de cada vuelta tienen un número de orden dentro de la vuelta (primera, segunda, ..., undécima, decimocuarta, ...). Además, guardaremos sus ciudades de origen y destino, los kilómetros y la fecha de celebración

- Se desea saber qué ciclistas han disputado cada etapa, teniendo en cuenta que todo ciclista que está en la base de datos ha corrido, al menos, una etapa. Los ciclistas tendrán un identificador y se almacenará, además, su nombre y fecha de nacimiento. Se desea saber el tiempo invertido en cada etapa y si la ha completado (o se ha retirado)

- A los ciclistas se les hace un control de dopaje cada vez que disputan una etapa. Hay diversos tipos de controles de dopaje y a un ciclista se le pueden hacer varios tipos de controles pero solo uno en cada etapa. De cada control realizado a un ciclista se almacena sus niveles de sangre y oxígeno

- La organización cuenta con una serie de pulseras de actividad que se le entregan a cada ciclista en una vuelta pero que tienen que devolverla al final de la misma. En la pulsera se almacenarán datos como la velocidad media, máxima, etc.

Ejercicio. Entidad bancaria (I)

- Los empleados se identifican por un código pero también nos interesa almacenar su DNI, NSS, nombre y apellidos. Habrá que registrar su ciudad de residencia, teniendo en cuenta que hay ciudades donde no reside ningún empleado

- Interesa saber en qué ciudades están ubicadas las diferentes agencias de la entidad bancaria. Estas agencias se identifican por la ciudad en la que se encuentran y por un nombre que permite distinguir las agencias de una misma ciudad. También se desea almacenar el número de habitantes de las ciudades, así como la dirección y el número de teléfono de las agencias. En la base de datos hay ciudades donde no hay ninguna agencia

- Un empleado, en un instante determinado, trabaja en una sola agencia, pero puede ser trasladado a otra o, incluso, que vuelva a trabajar en una agencia donde ya había trabajado anteriormente. Se quiere tener constancia del historial del paso de los empleados por las agencias

- Los empleados pueden tener títulos académicos y, de aquellos que lo tienen, se quiere saber cuáles son dichos títulos

Ejercicio. Entidad bancaria (II)

- Cada empleado tiene una categoría laboral determinada (interventor, administrativo, etc.). A cada categoría le corresponde un sueldo base y un precio por hora extra. Se quiere saber la categoría actual de cada empleado, así como el sueldo y el precio de la hora extra de cada categoría

- Algunos de los empleados están afiliados a un sindicato. Hay que descontar de la nómina mensual la cuota sindical a los afiliados a cada sindicato. Esta cuota es única para todos los afiliados a un sindicato determinado. Es necesario almacenar las afiliaciones de los empleados y las cuotas correspondientes a los diferentes sindicatos

- Los empleados tienen la posibilidad de pedir diferentes tipos de préstamos (por matrimonio, adquisición de vivienda, estudios, etc.), que pueden ser concedidos o no. No hay limitación a la hora de pedir varios préstamos a la vez, siempre que no se pida más de uno del mismo tipo al mismo tiempo. Se quieren registrar los préstamos pedidos por los empleados, y hacer constar si se han concedido o no. Cada tipo de préstamo tiene establecidas diferentes condiciones de las que nos interesa saber el tipo de interés y su periodo de vigencia