Bases de Datos

Databases: General concepts (DDBB)

An introduction through an example

Introduction

- If you understand this exercise (steps, consequences, alternative solutions, etc) then you understand the keys to data management.
- So, do it carefully.

The exercise: a telephone list

- Take a sheet of paper
- Write 5 telephone numbers and the name of its contact.

Maybe, you have a list similar to this. A table where rows describe the contacts and each column is a type of information (name, number).

Luis	1111111
María	2222222
Alfredo	3333333
Sonia	4444444
Ramón	5555555

- Search Sonia's phone number. Describe the process you have followed.
- Now, search for the contact with number 3333333. Have you followed the same process?
  - As you can prove, the search is faster when you order (index) the data.
- If you still do not appreciate the differences, try repeating the exercise with 100 or 10,000 contacts.

- Now on your sheet of paper, add to each contact the groups to which they belong

Luis	lab_bbdd	1111111
María	soccer, lab_bbdd, party	2222222
Alfredo	soccer	3333333
Sonia	lab_bbdd	4444444
Ramón	none or blank	5555555

- Search lab_bbdd's members and phone numbers. Describe the process you have followed.
  - Until you have verified all the data in the "group" column, you can not be sure that you have found all the members of lab_bbdd. This is because the data in the "group" column has no structure.
  - To solve it, maybe we can add more columns; one for each group.

Contact	soccer_grp	lab_grp	party_grp	number
Luis		lab_bbdd		1111111
María	soccer	lab_bbdd		2222222
Alfredo	soccer		party	3333333
Sonia		lab_bbdd		4444444
Ramón				5555555

- Search again the lab_bbdd's members and phone numbers. Describe the process you have followed. Easier?
- Delete contact alfredo and add Manuel, a contact of the swimming group and with the telephone number: 7777777

- You had to add a new column “swimming_grp”. On the other hand, the “party_grp” column is empty.

Contact	soccer_grp	lab_grp	party_grp	Swimming _grp	number
Luis		lab_bbdd			1111111
María	soccer	lab_bbdd			2222222
~~Alfredo~~	~~soccer~~	-	~~party~~		~~3333333~~
Sonia		lab_bbdd			4444444
Ramón					5555555
Manuel				Swimming	7777777

- Now on your sheet of paper, add a comment about each group.
- How many times do you have to write the same comment?

Contact	soccer_grp	lab_grp	party_grp	Swimming_grp	number
Luis		lab_bbdd			1111111
María	Soccer (the Sunday match)	lab_bbdd			2222222
~~Alfredo~~	~~Soccer (~~the Sunday match)	-	~~party~~		~~3333333~~
Sonia		lab_bbdd			4444444
Ramón					5555555
Manuel				Swimming	7777777

- Maybe, we could design another structure to make easier the maintenance of this telephone list.
- Make a proposal to add more groups and members, new contacts, delete members of the group, etc. easer?

- Split into tables with only data describing the same thing.
- Add new tables that relates things of different tables.

Contact	number
Luis	1111111
María	2222222
~~Alfredo~~	~~3333333~~
Sonia	4444444
Ramón	5555555
Manuel	7777777

Contact	Group
Luis	lab_bbdd
María	Soccer
María	lab_bbdd
~~Alfredo~~	~~Soccer~~
~~Alfredo~~	~~Party~~
Sonia	lab_bbdd
Manuel	swimming

Group	Comment
lab_bbdd	
Soccer	the Sunday match
Party	
swimming	

- Again, try to add more groups and members, new contacts, delete members of the group, etc. Easer?
- This is one of the keys of Relational Databases; to establish relationships between tables data on a single type of entities (contact, group, member).

- In SQL.

[Importante]

```
CREATE TABLE contact
(
   contact character varying(50) NOT NULL primary key,
   number bigint
);
```

```
CREATE TABLE cgroup
(
   cgroup character varying(50) NOT NULL primary key,
   comment text
);
```

```
CREATE TABLE contactgroup
   (contact character varying(50) references contact(contact),
   cgroup character varying(50) references cgroup(cgroup),
   PRIMARY KEY (contact, cgroup)
);
```

Tues 22/01/19

1- Al diseñar la consulta hay que verificar que los datos que devuelven las consultas son los que tienen que estar únicamente.
2- Identificar los datos que se manejan
3 - Indexar (ordenar)
4 - CRUD eficiencia
- frecuencia de actualizar la tabla

• Bases de datos no relacionales => BIG DATA
↳ orientadas a grafos

Aunque dos bases de datos tengan la misma info., si están orientadas a consultas diferentes, la base de datos tendrá una estructura diferente (Ley de Pareto).

[Introducción a SQL]

SQL lenguaje standard para cualquier BD.

Definición de datos

Los datos se colocan en las columnas de las tablas.

Tipos de datos

Smallint, integer, bigint, decimal (=), numeric, real, double precission,
smallserial (autoincrementing integer), bigserial, money
-> usar el tamaño que más se ajusta a las necesidades

SERIAL -> longitud integer y comienza desde el 1.... en adelante.

• money -> la precisión la determina la configuración i.c. monetary.
• LOCALE : moneda española (decimal con coma), moneda inglesa (decimal con punto), ...

• NUMERIC (4, 2) -99'99 ... 99'99
NUMERIC (4) - 9999 ... 9999

character (n) (tamaño de campo fijo) -> el sistema trunca la información (puede informar del error)
char (n) se completa rellenando con carácteres en blanco.

character, varying (n), -> tamaño variable con límite
varchar (n)

text -> variable de longitud ilimitada (fichero a parte)

Tipos binarios

Se guarda el fichero en el disco dura y guardamos el path.
CLOB y BLOB

date -> yyyy/mm/dd COMILLAS SIMPLES

time -> hh:mm:[ss:[mmm]]
podemos guardar la zona horaria [am / pm] [zzz] (código de la zona horaria)

time stamp -> time and date w/ or without time zone

interval -> rangos de fecha y tiempo

Tipo Booleano

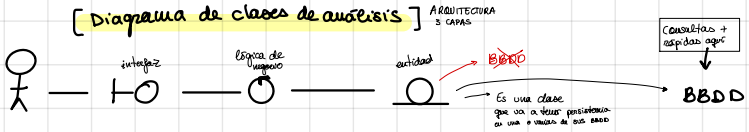
Diferentes carácteres para indicar TRUE o FALSE.

Crear una tabla

```
CREATE TABLE product ( IF NOT EXIST -> no sobreescribe
    product_no integer,        nombre_columna tipo de datos
    name text,                 (entrecomillado comillas simples)
    price numeric (7,2)        (ejemplo 50000.00)
);
```

ERP -> todas las aplicaciones que tiene una empresa están guardadas en una única BBDD / único repositorio.

[Diagrama de clases de análisis] ARQUITECTURA 3 CAPAS



- Usamos el esquema por defecto
- Crea un tipo de datos compuesto -> producto
  - Se accede nombredelatabla.nombredelcampo

- tableoid / WITH OIDS -> identificadar de objeto

CONSTRAINTS / RESTRICCIÓNES

- Reglas que deben cumplir los datos introducidos
  - column constraints -> 1 columna
  - table constraints -> varias columnas

  · Usamos CKECK (...)
  - cuando se crea la tabla le podemos poner nombre CONSTRAINT nombre

- default : valor por defecto (si no hay se pone a null)

- NOT NULL : campo obligatorio
- UNIQUE : el dato en esa columna es único
  -> combinación de dos dígitos (ejemplo código postal)
  UNIQUE (a, c) (pareja a, c)

PRIMARY KEY

- único y no nulo UNIQUE, NOT NULL
- indexa la tabla por esa columna
  PRIMARY KEY (a, c)

  a no nulo, c no nulo (espacio en blanco NO ES NULO)
  a y c se pueden repetir
  el conjunto de (a,c) no se puede repetir

FOREIGN KEYS

REFERENCES : valor obligado a ya existir en otra tabla => problemas de CONSISTENCIA
  REFERENCES product (product_no)
  INTEGRIDAD REFERENCIAL

Wed 30/01/19

TRABAJO

1ª Entrega : mantener

- Descripción del sistema
- 2 casos de uso del sistema
- todas las req. de información
  - describir minuciosamente la información que el sistema debe devolver
  - precondición : pasos y casos de uso realizados antes de comenzar con el escenario normal
  - -> describir en comentarios entidades y atributos consultados

INDICADORES ELIMINATORIOS

- formato
- los casos de uso sean CRUD (crear, leer, actualizar y borrar)

Ejemplo : registro en una web

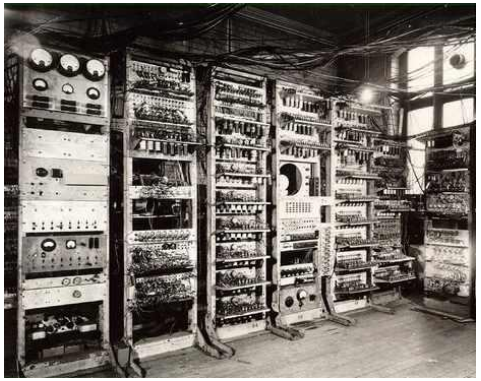
1. SISTEMAS DE GESTIÓN DE BASES DE DATOS

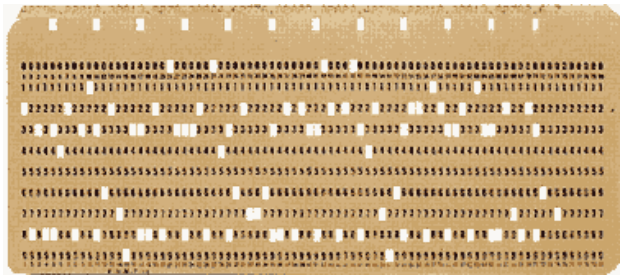
1. Origen y Evolución de las Bases de Datos
2. Concepto de Base de Datos: Objetivos de las Bases de Datos
3. Independencia de Datos. Arquitectura de Bases de Datos
4. Sistemas de Gestión de Bases de Datos
5. Administración de Bases de Datos

1. Origen y Evolución de las Bases de Datos

1. Origen y Evolución de las Bases de Datos

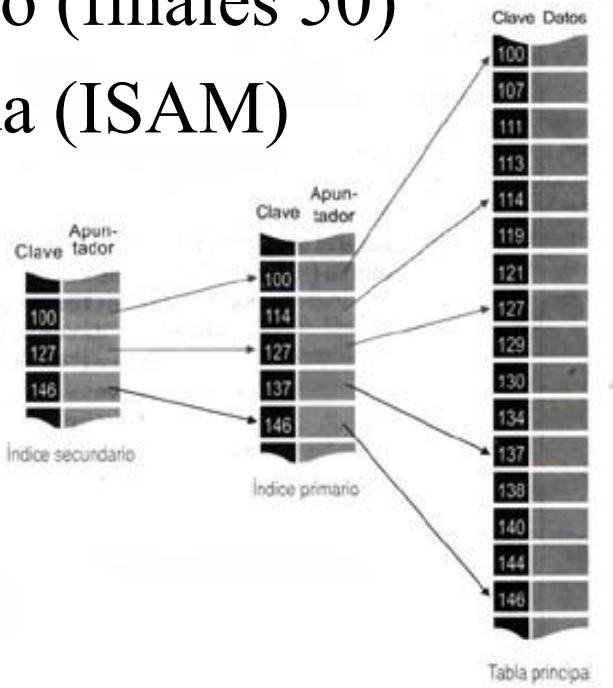
- Primera generación (años 50)
  - El fichero no existe, los datos sólo existen dentro de los programas
  - Nuevos soportes:
    - cinta perforada y magnética ➡ Acceso secuencial
  - Acceso a ficheros: lenguaje máquina y ensamblador





1. Origen y Evolución de las Bases de Datos

- Segunda generación (años 60)
  - Diálogo interactivo con la máquina
  - Disco magnético ➡ acceso directo (finales 50)
  - Organización secuencial indexada (ISAM)
  - Sistemas de ficheros



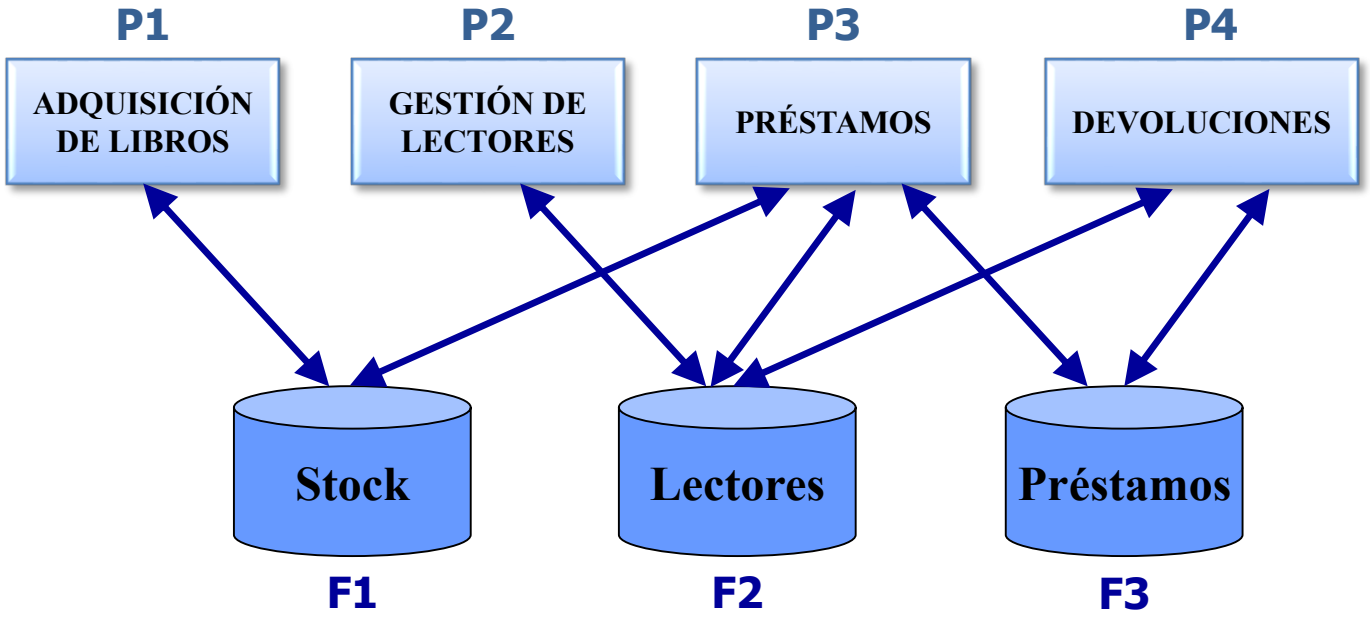
1. Origen y Evolución de las Bases de Datos

- Segunda generación (años 60)
  - Sistemas de ficheros
    - Asociación estática de los ficheros a los programas de forma individual
      - Ficheros integrados en la aplicación y el hardware
      - Ficheros a la medida de la aplicación
      - Formatos de ficheros heterogéneos

Redundancia y problemas de compartición

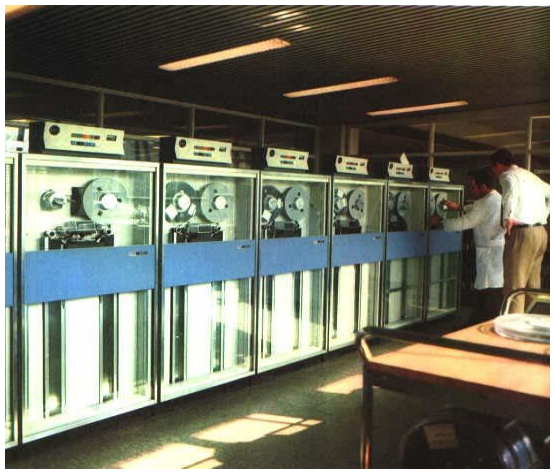
1. Origen y Evolución de las Bases de Datos

- Segunda generación (años 60)
  - Sistemas de ficheros



1. Origen y Evolución de las Bases de Datos

- Segunda generación (años 60)
  - Sistemas de ficheros
    - Problemas
      - Redundancia
      - Inconsistencia de datos
      - Dependencia de datos
      - Excesivo mantenimiento
      - Poca flexibilidad frente a cambios
      - Baja productividad
      - Limitación de recursos compartidos
      - Medidas de seguridad difíciles



1. Origen y Evolución de las Bases de Datos

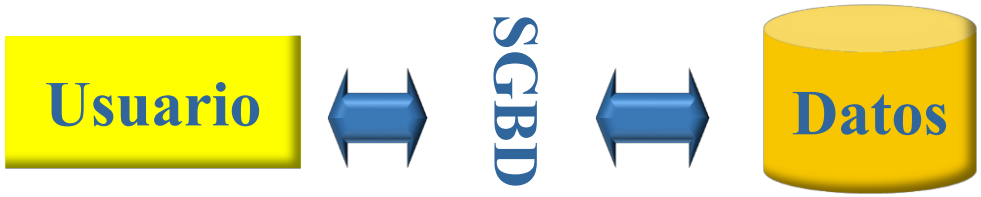
- Tercera generación (años 70, pre-relacional)
  - Distinción entre estructura lógica y física
    - arquitectura a dos niveles
  - Unificación de la información sin perder la perspectiva de usuarios
  - Distinción entre significado y valor almacenado

1. Origen y Evolución de las Bases de Datos

- Tercera generación (años 70 , pre-relacional)
  - Evolución de las bases de datos
    - En Publicaciones técnicas/militares: hechas a medida
    - Años 60: Primer software para un conjunto de aplicaciones: BOMP, CFMS, ...
    - 1962-64: Introducción de estructuras de **cadenas** y **anillos** por Bachman (IDS ⇨ Codasyl)
    - 1969 (Desde 1965): IMS/1 de IBM. Inicialmente diseñada para el proyecto Apollo
    - 1968: CODASYL A partir del modelo de Bachman establece el concepto **set** ⇨ **Modelo RED**

1. Origen y Evolución de las Bases de Datos

- Tercera generación (años 70 , pre-relacional)
  - Evolución de las bases de datos



1. Origen y Evolución de las Bases de Datos

- Cuarta generación (años 80, relacional)
  - Nuevos productos: Sistemas de Bases de Datos
  - Control centralizado ➡ evita redundancia
  - Clara distinción entre el modelo lógico y el físico
  - Alto grado de independencia de datos
  - Almacenamiento transparente al usuario
  - Lenguajes más potentes

    (*Qué* en lugar de *Cómo*)

1. Origen y Evolución de las Bases de Datos

- Cuarta generación (años 80, relacional)
  - Modelo relacional (CODD, 1969-1970)
    - Basado en el álgebra y la teoría de conjuntos
    - Prototipos
      - INGRES Universidad de Berkeley (1973-75)
      - SYSTEM-R de IBM (1974-77)
    - Sistemas comerciales
      - INGRES de RTI (1980)
      - SQL/DS de IBM (1981)
      - ORACLE de RSI (1981)
      - DB2 de IBM (1983)

1. Origen y Evolución de las Bases de Datos

- Quinta generación (años 90, post-relacional)
  - Bases de datos deductivas
  - Bases de datos orientadas a objetos
  - Bases de datos activas
  - ...

2. Concepto de Bases de Datos: Objetivos

2. Concepto de Bases de Datos: Objetivos

- Objetivos
  - Independencia de datos (Flexibilidad)
    - Los cambios en las aplicaciones no deben imponer un nuevo diseño y viceversa
  - Coste mínimo
    - Adaptación rápida y con coste mínimo a las nuevas características de la empresa
  - Consistencia y mínima redundancia (Integridad)
    - Evitar repetición innecesaria de datos y la incoherencia entre estos
  - Datos compartidos
    - Permite una mayor disponibilidad de los datos

ERP: sistema de aplicaciones para la gestión integral de una empresa
- varían los flujos de datos (enterprise resource planning)

Atributo derivado: atributo obtenido a partir de otros atributos de la BBDD

2. Concepto de Bases de Datos: Objetivos

- Objetivos
  - Versatilidad en la representación de los datos → vistas / view
    - Distintos usuarios quieren ver de forma distinta los datos
  - Capacidad de búsqueda
    - Rapidez en la exploración de la Base de Datos
  - Integridad
    - Garantizar la exactitud y veracidad de los datos
  - Tolerancia a fallos y seguridad
    - Protección de los datos ante accesos no autorizados y fallos

2. Concepto de Bases de Datos: Objetivos

- Base de datos

tienen una estructura conocida → diagrama de clases

  - Colección de datos estructurados según un modelo que refleje las entidades, relaciones y restricciones existentes en el mundo real.
  - Los datos han de ser compartidos por diferentes usuarios y aplicaciones, deben mantenerse independientes de éstas, y su definición y su descripción deben ser únicas, estando almacenadas junto a los mismos.
  - Los tratamientos habrán de conservar la integridad y seguridad de los datos.

2. Concepto de Bases de Datos: Objetivos

- Transacción
  - Conjunto de operaciones que deben realizarse de forma atómica (todas o ninguna).
  - Garantizar la consistencia de los datos
  - Debe estar aislada de otras transacciones
  - Debe perdurar en el tiempo, los cambios que produce deben quedar reflejados en la base de datos.
  - SQL
    - COMMIT/ROLLBACK

3. Independencia de Datos. Arquitectura de una Base de Datos

3. Independencia de Datos. Arquitectura de una Base de Datos

- Implica la separación lógica y física de la base de datos
  - Influencia en la arquitectura del sistema
  - Grado de dependencia
  - Fase o etapa del proceso donde se efectúa la transformación entre niveles.

Ejemplo:
- lógica: nombre de un alumno de la clase
- física: fila 12 de no sequé rollo y lo que ocupa

3. Independencia de Datos. Arquitectura de una Base de Datos

- Independencia de datos es (i)
  - La capacidad de un sistema de gestión de bases de datos para permitir que las referencias a los datos almacenados, especialmente en los programas, y sus descripciones de los datos estén aisladas de los cambios y de los diferentes usos en el entorno de los datos, como pueden ser:
    - la forma de almacenar dichos datos
    - el modo de compartirlos con otros programas
    - cómo se reorganizan para mejorar el rendimiento del sistema de bases de datos

3. Independencia de Datos. Arquitectura de una Base de Datos

- Independencia de datos es (ii)
  - La inmunidad de las aplicaciones ante cambios de la estructura de almacenamiento y de los métodos de acceso.

3. Independencia de Datos. Arquitectura de una Base de Datos

- Tipos de dependencia
  - De descripción
    - Separación de la definición de datos a nivel físico y lógico
  - De manipulación
    - Independencia respecto a los caminos de acceso y soportes físicos

3. Independencia de Datos. Arquitectura de una Base de Datos

- Inmunidad del usuario frente a
  - Atributos: Nombre, tamaño, tipo,….
  - Entidades: Nombre, nuevos atributos,…
  - Estructuras: Cambio en nombre de relaciones, cardinalidad, ...
  - Nivel interno: Tamaño de bloques, longitud de registros, accesos,….
  - Nivel físico: Tipo de soporte, tamaño de los volúmenes, sistema operativo,....

3. Independencia de Datos. Arquitectura de una Base de Datos

(3 capas)

- Arquitectura a tres niveles
  - Propuesta por el comité ANSI/X3/SPARC
  - Asocia a cada nivel un esquema
    - Esquema externo —> capa O
    - Esquema conceptual —> cómo meteremos en la base de datos la información con un lenguaje que entienda el que diseña las clases de O
    - Esquema interno —> cómo se guarda en los discos duros

3. Independencia de Datos. Arquitectura de una Base de Datos

- Arquitectura a tres niveles
  - Nivel interno
    - Es el nivel más próximo al almacenamiento físico
    - Trata los datos como registro internos (no como bloques de datos)
    - Contiene información sobre:
      - estructuras de datos usadas
      - mecanismo de acceso
      - distribución de registros en el espacio lógico

3. Independencia de Datos. Arquitectura de una Base de Datos

- Arquitectura a tres niveles
  - Nivel conceptual
    - Vista lógica general de todos los datos
    - Interrelaciones entre los datos
    - Uso de un modelo de datos que proporcione un nivel de abstracción de la vista interna
    - Uso de un lenguaje de definición de datos (DDL)

3. Independencia de Datos. Arquitectura de una Base de Datos

- Arquitectura a tres niveles
  - Nivel externo
    - Comprende las vistas individuales de los usuarios (subesquemas)
    - Extractos de las vista conceptual

[VISTAS -> CONSULTA]
pasos de un caso de uso que dan el resultado de una consulta, esa consulta será la vista

3. Independencia de Datos. Arquitectura de una Base de Datos

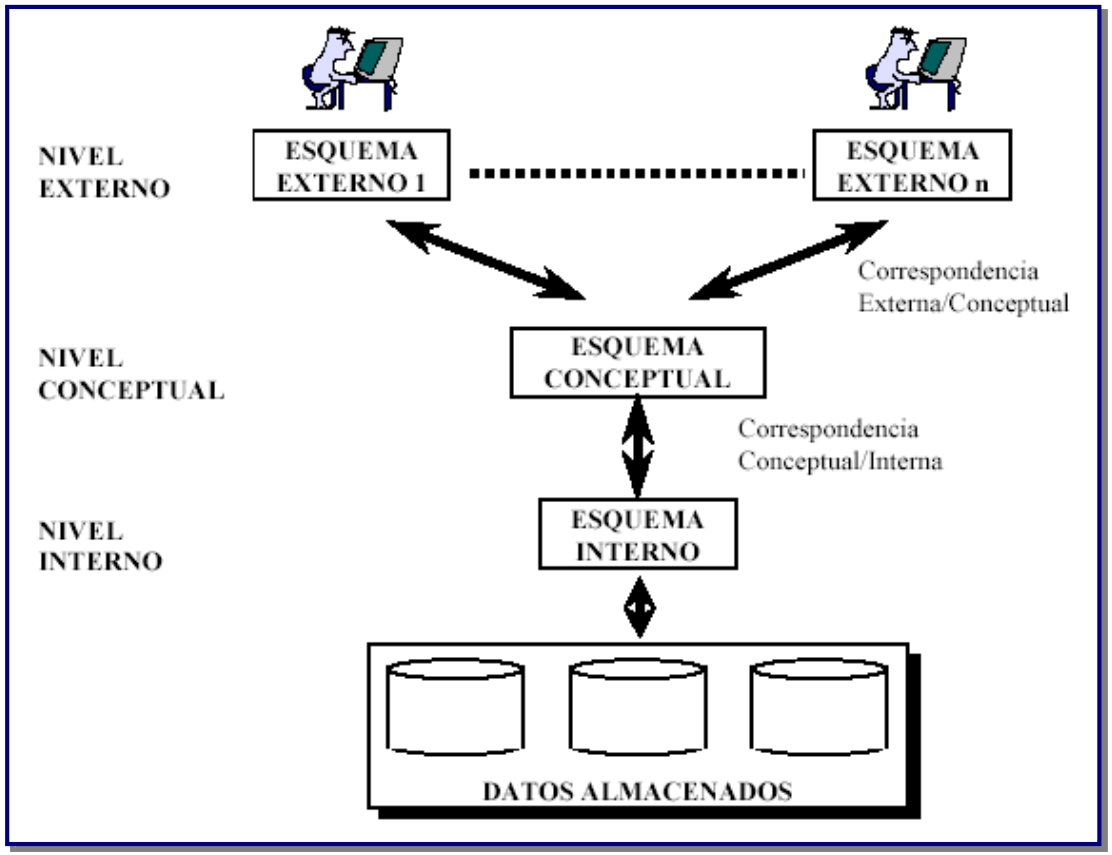
- Interfaces entre niveles
  - Los proporciona el SGBD (DBMS)
  - Transformaciones entre los esquemas
  - Consultas y actualizaciones
    - DML (Lenguaje de Manipulación de Datos)
      - Lenguaje interactivo
      - Precompilador
      - Extensión del compilador

3. Independencia de Datos. Arquitectura de una Base de Datos

- Interfaces entre niveles
  - Creación de las estructuras
    - DDL(Lenguaje de Definición de Datos)
      - DDL interno
      - DDL conceptual
      - DDL externo
  - DDL + DML = DSL (sublenguaje de datos)
  - DML ⇨ todos los usuarios
  - DDL ⇨ DBA (Administrador de la base de datos)

3. Independencia de Datos. Arquitectura de una Base de Datos

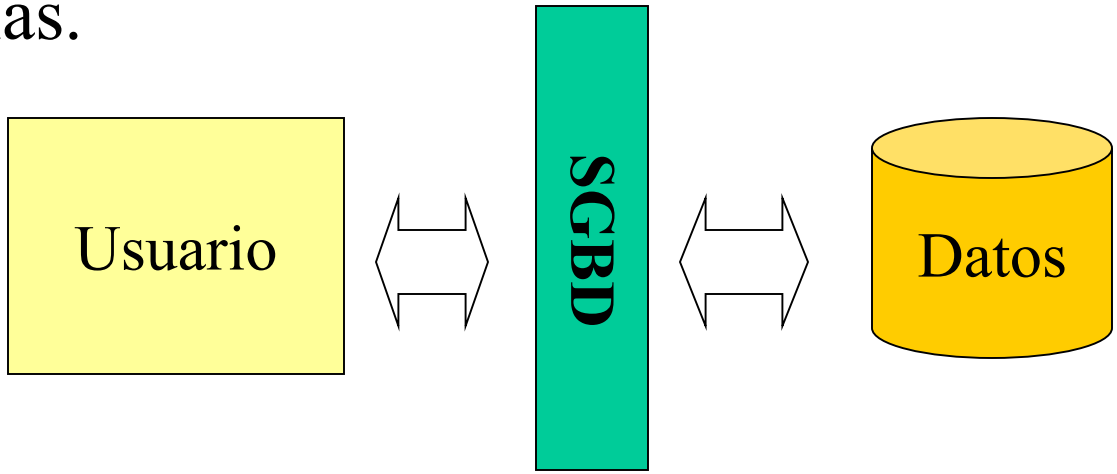
- Arquitectura a tres niveles



4. Sistemas de Gestión de Bases de Datos

4. Sistemas de Gestión de Bases de Datos

- SGBD (Sistema de Gestión de BDD)
  - Conjunto de programas que actúan como intermediario entre los datos y los usuarios.
  - Recoge las peticiones del usuario y responde a ellas.



4. Sistemas de Gestión de Bases de Datos

- Características
  - Descripción de la BDD exterior a los programas y gestionada por el SGBD.
  - Los programas no leen ni graban directamente sobre la estructura interna de la BDD.
  - Gestión de
    - Control de accesos concurrentes
    - Control de autorizaciones de acceso
  - Reconstrucción y/o restauración de la BDD

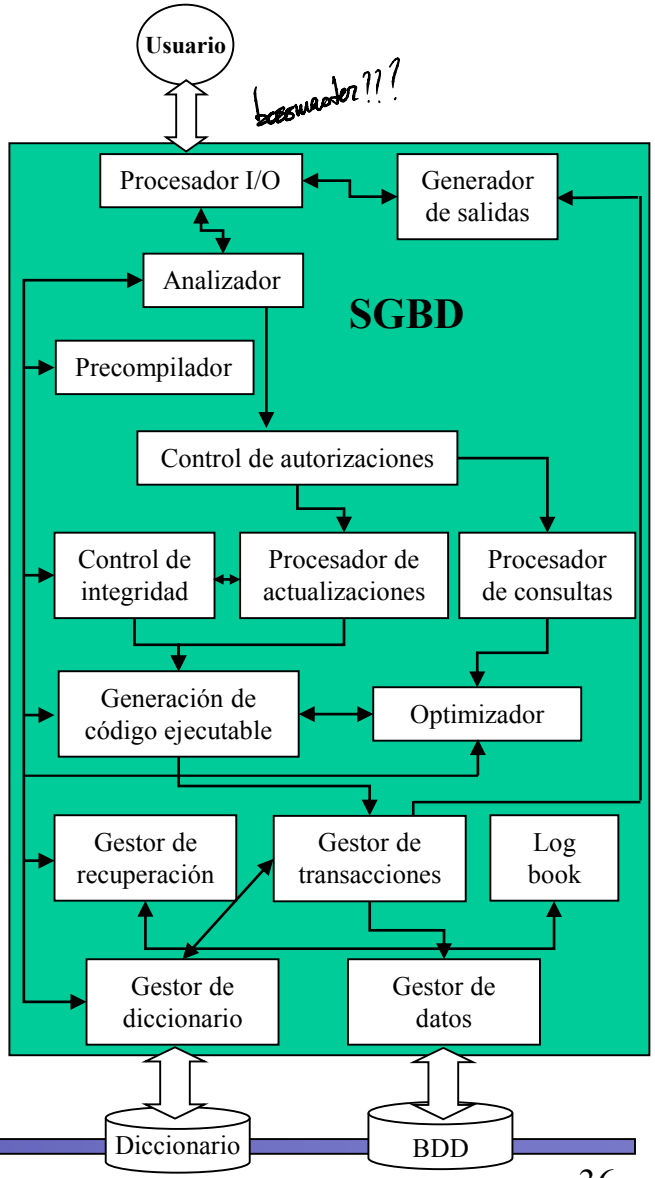
4. Sistemas de Gestión de Bases de Datos

- El Diccionario de datos
  - Es una BDD que contiene información sobre la propia BDD (metadatos)
  - Debe suministrar documentación única y actualizada de forma que pueda ser usada para obtener información sobre los tipos de datos almacenados y cómo se deben usar.

—> campo sueldo
definir metadatos: nº de dígitos, rangos y definición para toda la app.

4. Sistemas de Gestión de Bases de Datos

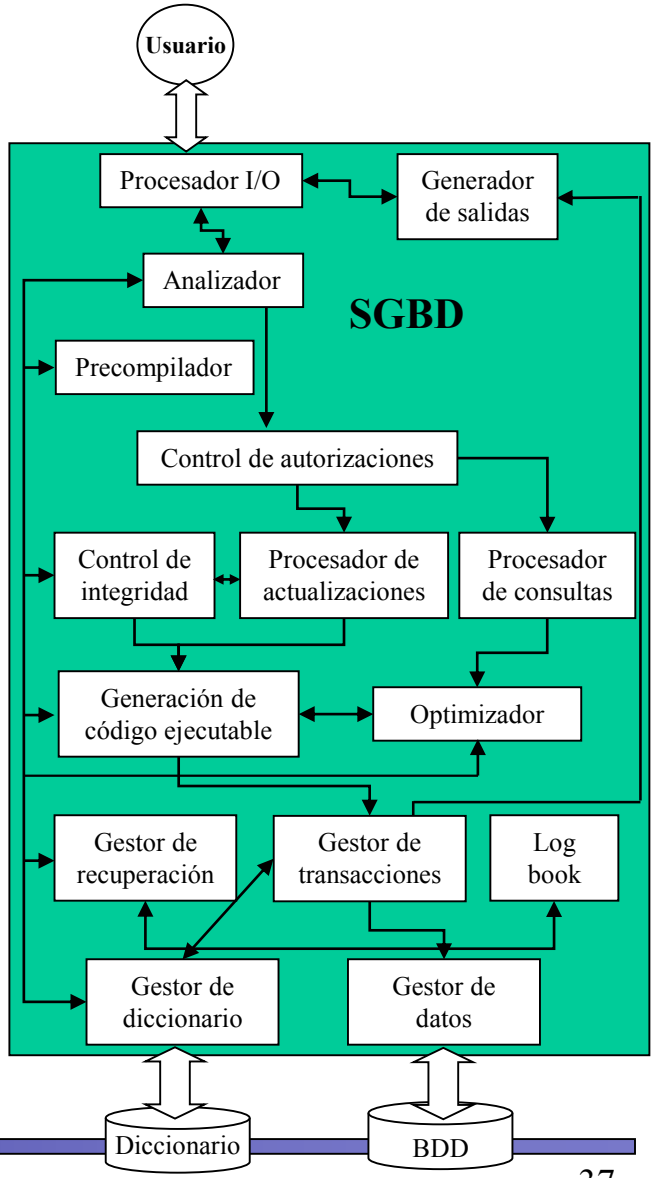
- **Componentes de un SGBD**
  - **Procesador de I/O**
    - Directamente asociado al usuario
    - Toma las órdenes y muestra la respuesta
  - **Analizador**
    - Análisis sintáctico de la orden ⇦ Diccionario
    - DDL ⇨ Actualización del Diccionario
    - Lenguaje inmerso ⇨ Precompilador
  - **Control de autorizaciones**
    - Chequeo de autorizaciones ⇦ Diccionario
    - Obtención de código intermedio



4. Sistemas de Gestión de Bases de Datos

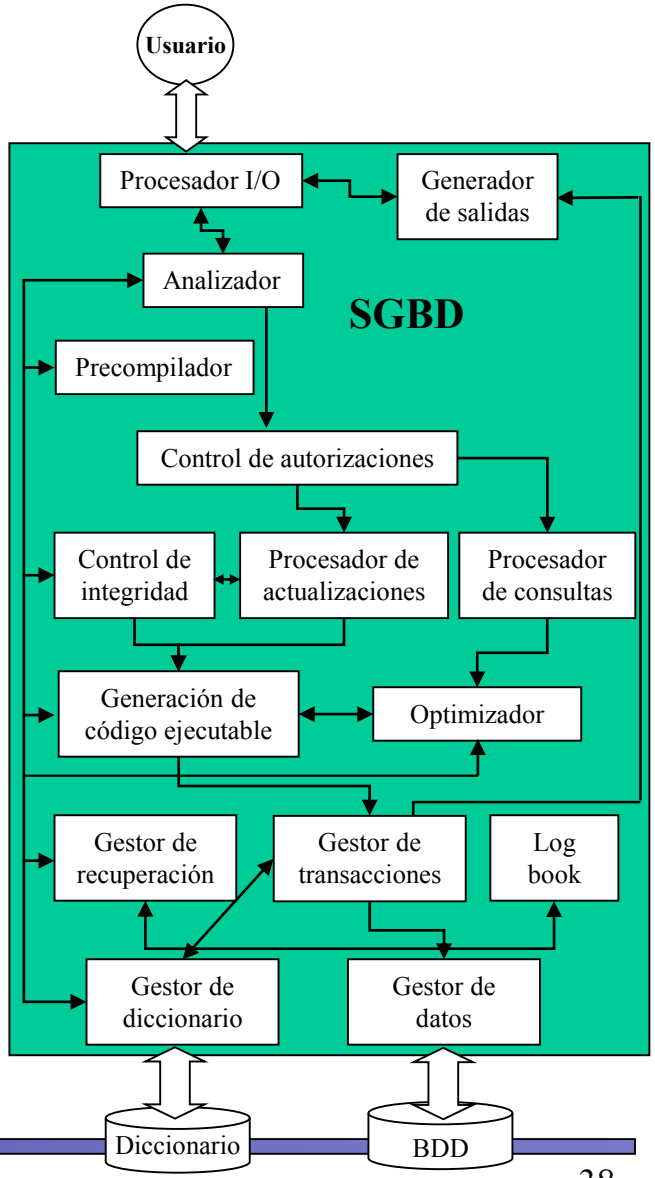
- Componentes de un SGBD
  - **Procesador de actualizaciones y control de integridad**
    - Ejecución de la actualización
    - Control de consistencia
  - **Procesador de consultas y optimizador**
    - Transforma la consulta en términos conceptuales
    - Reformulación para la optimización el acceso

      —> BBDD : álgebra relacional



4. Sistemas de Gestión de Bases de Datos

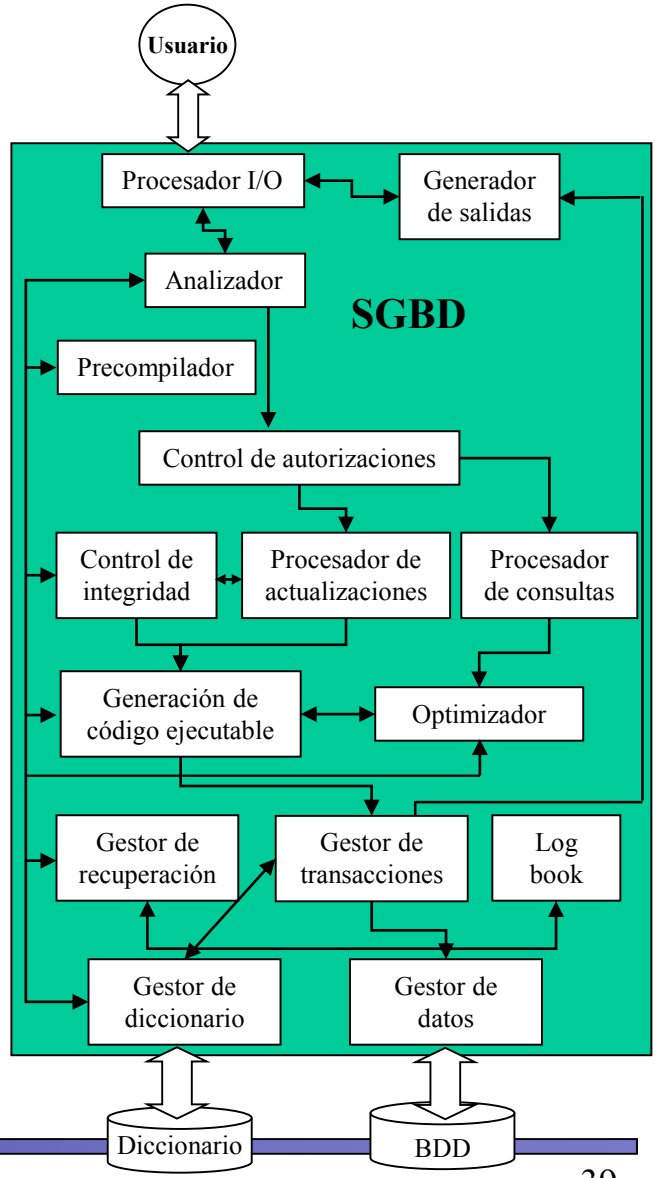
- Componentes de un SGBD
  - **Generador de código ejecutable**
    - Secuencia de lecturas y escrituras en disco
  - **Gestor de transacciones**
    - Sincronización de los accesos concurrentes
  - **Gestor de recuperación** —> rollback
    - Recuperación del estado de la BDD previo al fallo —> estado estable
  - **Gestor de datos**
    - Gestión del hardware
    - Ejecución de los accesos físicos



4. Sistemas de Gestión de Bases de Datos

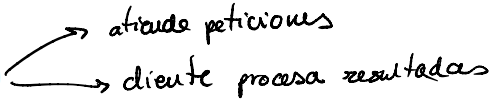
- Estructura del SGBD por interfaces

Interfaz	Componentes del SGBD asociados
Al usuario	Procesador I/O
Externo ←Conceptual	Analizador, precompilador, procesador de actualizaciones y procesador de consultas
Conceptual ← Interno	Generador de código, optimizador
Interno ← Base de datos	Gestor de transacciones, gestor de datos



4. Sistemas de Gestión de Bases de Datos

- Arquitectura de un SGBD multiusuario
  - Arquitectura cliente-servidor



    - El SGBD actúa como un **servidor** proporcionando todo el soporte de los niveles externo, conceptual e interno
    - Las aplicaciones que se ejecutan sobre SGBD actúan como **cliente** y las interfaces de consulta que interactúan con el usuario y envían consultas u otros comandos al servidor
    - **Ej.** Los sistemas relacionales utilizan el lenguaje SQL para representar peticiones del cliente ante el servidor. El servidor las procesa y envía la respuesta al cliente en forma de tabla o relación.

4. Sistemas de Gestión de Bases de Datos

- Arquitectura de un SGBD multiusuario
  - Arquitectura cliente-servidor
    - Funciones del servidor
      - Aceptar y procesar las peticiones de los clientes de la BDD
      - Comprobar autorizaciones
      - Asegurar que se cumplen las restricciones de integridad
      - Realizar los procesos de consulta/actualización y transmitir la respuesta al cliente
      - Mantener el diccionario de datos
      - Proporcionar acceso concurrente a la BDD, ...

5. Administración de la Base de Datos

5. Administración de la Base de Datos

→ gestión de la capacidad de los servidores
  → SGBD cuello de botella
    – reducir colas de peticiones ...

- Tareas de administración
  - **Gestionar la estructura de la base de datos**
  - **Descripción conceptual**. Una vez conocidos los requisitos de información, es preciso realizar el diseño conceptual
  - **Descripción física de los datos**. Encontrar una estructura interna que soporte el esquema lógico y los objetivos de diseño con la máxima eficiencia de los recursos máquina
  - Especificaciones de **vistas o subesquemas**
  - **Los estándares.** En cuanto a documentación, metodologías
  - **Procedimientos de explotación y uso**
  - **Aspectos relativos a seguridad, integridad y confidencialidad**

5. Administración de la Base de Datos

- Herramientas que utiliza el administrador
  - **Lenguajes de definición de datos(DDL)**
  - **Utilidades del SGBD** (copias de seguridad, asignación de usuarios, simulación de procesos, documentación, ...)
  - **Diccionario de datos**. Contendrá la definición y descripción de todos los datos (metadatos). Ayuda a la gestión de la información como recurso y conseguir la integración de la semántica de forma centralizada.

5. Administración de la Base de Datos

- Tipos de administradores
  - El administrador a nivel de **empresa** (N. Conceptual) → necesidades, permisos de acceso
  - El administrador de la **base de datos** (N. Interno)
  - Los administradores por **aplicaciones** (N. Externo)

5. Administración de la Base de Datos

- Tipos de administradores
  - Funciones del administrador de la **BDD** (i)
    - Fijar, a nivel orgánico, la compatibilidad entre trabajos y el orden de ejecución de los trabajos incompatibles
    - Obtener estadísticas sobre la base de datos
    - Proporcionar a los operadores del ordenador descripciones completas de lo que hay que hacer para la reconstrucción de las bases de datos
    - Establecer normas para el uso eficiente de la base de datos
    - Implementar los requisitos en cuanto a seguridad establecidos por el administrador a nivel de empresa

5. Administración de la Base de Datos

- Tipos de administradores
  - Funciones del administrador de la **BDD** (ii)
    - Escoger los valores de los parámetros físicos para la optimización de los accesos
    - Proceder a las reestructuraciones de las bases de datos exigidas por los cambios en el esquema interno
    - Seleccionar y desarrollar programas de servicios relativos a aspectos físicos de las bases de datos
    - Definir y mantener los esquemas internos, así como la documentación sobre ellos

5. Administración de la Base de Datos

- Tipos de administradores
  - Funciones del administrador por **aplicaciones**
    - Definir los esquemas externos y mantenerlos
    - Asignar autorizaciones de acceso a la base de datos a los usuarios de éstas

2. INTRODUCCIÓN AL DISEÑO DE BASES DE DATOS

1. El Problema del Diseño de una Base de Datos
2. Etapas del Diseño
   - 2.1 Formulación de requisitos y análisis
   - 2.2 Diseño conceptual
   - 2.3 Diseño lógico
   - 2.4 Diseño físico
3. El Modelo de Datos Entidad-Relación Extendido

1. El Problema del Diseño de una Base de Datos

[Metodología en Cascada]

Requisitos -> BBDD que satisfaga las necesidades de ese modelo de negocio.

1. El Problema del Diseño de una Base de Datos

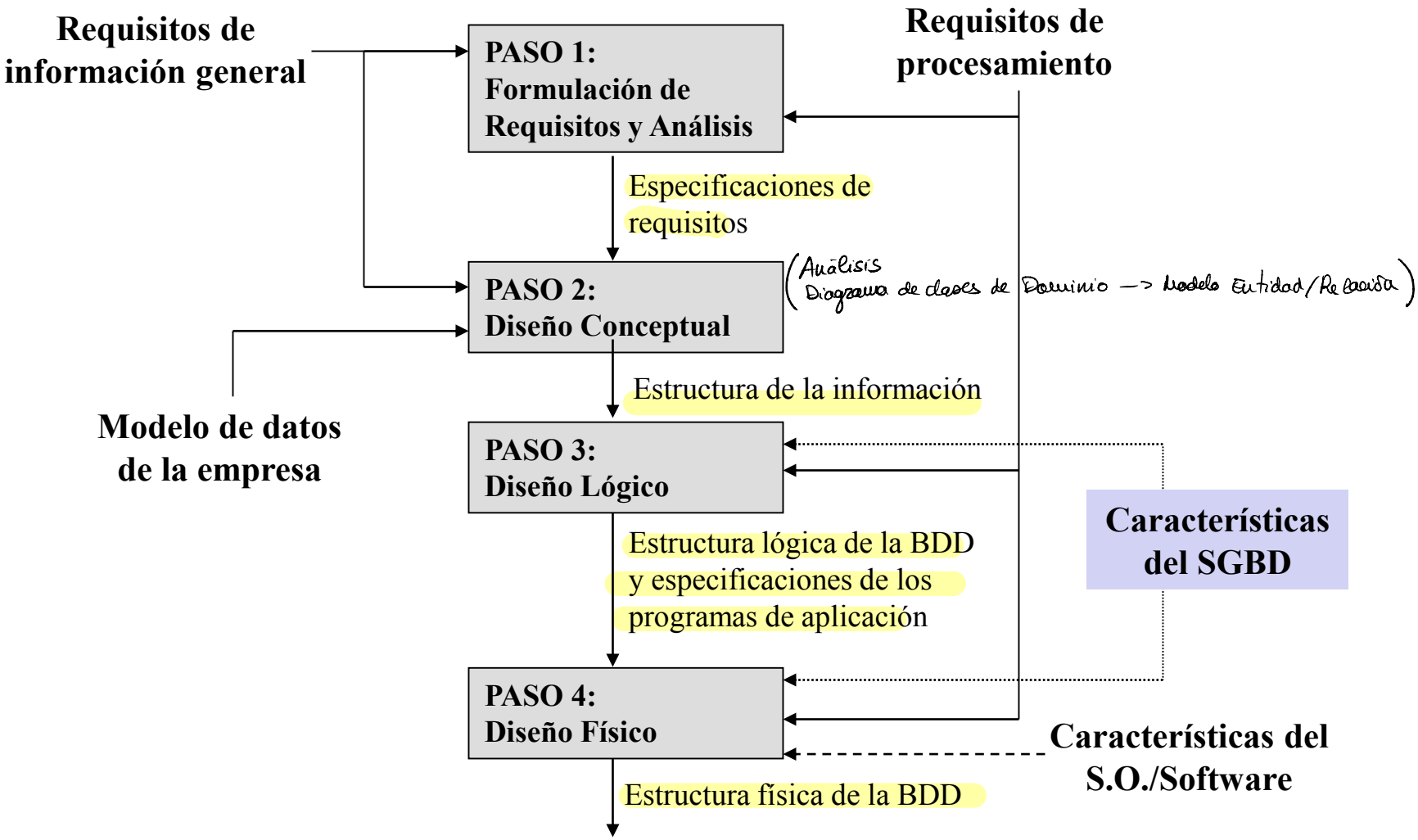
- Concepto
  - Proceso de desarrollo de las estructuras de una base de datos a partir de la utilización de los requisitos de datos
- Objetivo
  - Obtener bases de datos que representen fielmente la realidad y capaces de adaptarse fácilmente a los posibles cambios en los requisitos.

1. El Problema del Diseño de una Base de Datos

- Criterios para un diseño eficaz
  - Los datos de entrada (proporcionados por los requisitos) deben ser completos, consistentes y utilizables. → evitar cambios
  - La metodología de diseño debe ser correcta y utilizar efectivamente los datos de entrada
  - El personal técnico y los usuarios deben estar comunicados interactivamente. (Ing. de Requisitos)
  - Para conseguir una comunicación efectiva entre los desarrolladores y los usuarios es necesario utilizar un modelo conceptual (En este tema usaremos el modelo Entidad/Relación Extendido). (* hay cambios en OOD)
    ↳ diagrama de clases de análisis (?)

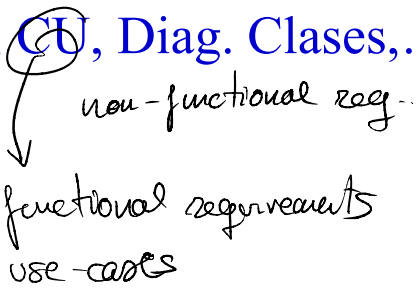
2. Etapas del Diseño de una Base de datos

2. Etapas del Diseño de una Base de datos



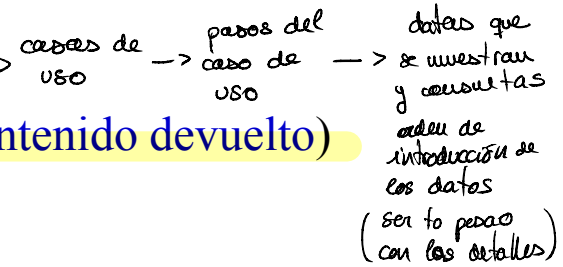
2.1 Formulación de requisitos y análisis Paso 1

- **Objetivo**
  - Identificar y describir los datos necesarios para la organización, su uso y reglas semánticas.
- **Entrada**
  - Fuentes: entrevistas con usuarios, documentos de la empresa, etc
  - Requisitos de información del usuario (datos usados y sus asociaciones). (OOD: DRS, Diag. Clases,…)
  - Requisitos de procesamiento (consultas, informes, frecuencia de informes, tiempo de respuesta, ...). (OOD: DRS, CU, Diag. Clases,…)



2.1 Formulación de requisitos y análisis

- Entrada (…)
  - Algunas posibles cuestiones a responder:
    - ¿Qué vistas de usuario se necesitan? (consultas)
    - ¿Qué elementos de datos necesitan estas vistas? (contenido devuelto)
    - Claves para identificar las entidades (filtrado)
    - Relaciones entre los datos
    - Necesidades operacionales: seguridad, integridad, tiempo de respuesta



- Salida
  - Especificaciones de Requisitos para el Diseño Conceptual.

2.1 Formulación de requisitos y análisis

- Pasos a seguir
  1. Identificar el alcance del esfuerzo de la definición de la base de datos
  2. Desarrollar una colección de metadatos estándar
  3. Identificar las vistas de usuario y requisitos de datos
  4. Construir un diccionario de datos y sus interrelaciones (metadatos)
     - Nº de elementos de datos
     - Nombre (atributo/req. info)
     - Tipo (de dato)
     - Longitud (del campo)
     - Rango (de valores posibles)
     - Frecuencia de uso
     - Autorización de actualización

2.1 Formulación de requisitos y análisis

- Pasos a seguir

5. Identificar el volumen de datos y los patrones de uso
   - Estimación del volumen de datos y su crecimiento
   - Uso de los datos: definir transacciones y accesos necesarios para cada transacción
6. Establecer los requisitos operacionales
   - Seguridad - Copias de seguridad y restauración
   - Integridad - Archivo (históricos, tiempo de vida)
   - Tiempo de respuesta - Crecimiento
7. Diagramas de flujo de Datos (DFD)

(en el Trabajo seguir las recomendaciones de "Normativa para el trabajo de asignatura")

2.2 Diseño conceptual
Paso 2

- En esta etapa, se transforma el esquema descriptivo, refinándolo y estructurándolo adecuadamente.
- Esta etapa responde a la pregunta: "¿**cómo** representar?"
- Es necesario buscar una representación basada en un modelo de datos que cumpla:
  - coherencia, completitud, no redundancia, simplicidad, fidelidad, etc.

2.2 Diseño conceptual

- Para la representación del esquema conceptual, usaremos el Modelo Entidad/Relación Extendido (EER - ME/R Extendido), además de unas plantillas que sirven de soporte documental junto al diagrama EER.

- Para generar el esquema conceptual es preciso interpretar las frases del lenguaje natural en el que está descrito el esquema percibido, convirtiéndolas en elementos del modelo EER.

2.2 Diseño conceptual

- Objetivos
  - Captar y almacenar el universo del discurso (OOD: dominio de la solución) mediante una descripción rigurosa, representando la información que describe a la organización y que es necesaria para su funcionamiento.
  - Aislar la representación de la información de los requisitos de la máquina y exigencias de cada usuario particular.
  - Independizar la definición de la información del SGBD concreto.
- En resumen, "*un esquema conceptual comprende una descripción central y única de los distintos contenidos de información que pueden coexistir en una base de datos*".

2.2 Diseño conceptual

- El ME/R Extendido permite obtener esquemas conceptuales caracterizados por su:
  - **Claridad**: su significado no sea ambiguo.
  - **Coherencia**: no existan contradicciones o confusiones.
  - **Completitud**: en cuanto a que el esquema conceptual debe representar lo esencial del fenómeno sin buscar la exhaustividad.
  - **Fidelidad**: en el sentido de que la representación del universo del discurso debe hacerse sin desviaciones ni deformaciones.
  - **Sencillez**: se debe buscar la máxima sencillez sin que ello vaya en detrimento de las características anteriores.

2.2 Diseño conceptual

- La **sencillez** del esquema conceptual debe basarse en que:
  - El número de componentes básicos sea tan reducido como sea posible.
  - Debe separar claramente conceptos distintos.
  - Independencia de los detalles de implementación.
  - La redundancia tiene que ser cuidadosamente controlada.
- Un mismo universo de discurso puede ser representado mediante múltiples esquemas EER.

2.2 Diseño conceptual

- Diseño Ascendente y diseño descendente
  - **Descendente** (*top-down*): cuya filosofía responde a que "el esquema conceptual refleje directamente la visión de la empresa que se intenta modelar en la BDD". Se parte de un modelo general de la empresa para elaborar el esquema conceptual y, posteriormente, sobre él se definen las vistas de usuario como subconjuntos de este esquema conceptual.(OOD: diagramas estructurales como el de clases, …)
  - Recomendado para BDD pequeñas.
    - Cuando no existe un OOD previo.

2.2 Diseño conceptual

- Diseño Ascendente y diseño descendente
  - **Ascendente** (*bottom-up*): este tipo de metodologías entienden el esquema conceptual como "*el resultado de la integración de las vistas de los grupos de usuarios*" - subsistemas. Por lo que se empieza construyendo las vistas de cada uno de estos subsistemas (que corresponden a las aplicaciones más importantes) y, teniendo en cuenta las restricciones entre dichas vistas, se elabora el esquema conceptual mediante un proceso de integración de vistas. (OOD: paquetes, subsistemas, etc.)
  - Recomendado para BDD medianas y grandes.
    - Más coherente con metodologías de desarrollo software evolutivas.

2.2 Diseño conceptual

- Combinar diseño Ascendente y diseño descendente (bottom-up)
  - Se pueden utilizar para contrastar los resultados del diseño ascendente y el diseño descendente.
    - No tienen por qué coincidir los resultados debido a que:
      - Los dos modelos son completados por diferentes personas en diferentes momentos.
      - El modelo de información de la empresa es más global.
      - El modelo Bottom-Up es más detallado e incremental.
      - El modelo de información de la empresa no está necesariamente normalizado.

- Salto al MER Extendido

2.3 Diseño lógico

- Objetivos
  - Transformar el modelo de datos conceptual en un modelo lógico conforme a un **SGBD concreto** y definir las especificaciones para los programas de acceso a la BDD.

2.3 Diseño lógico

- Entrada -> acelerar sólo las consultas que más se realizan (+ frecuentes)
  - Esquema conceptual
  - Cuantificación de los requisitos operacionales (¿cuál es su intensidad de uso esperada?)
  - Volumen y cuantificación de uso
  - Especificaciones de los programas de alto nivel
  - Características del SGBD

- En los sistemas que distinguen entre las bases de datos lógica y física parte de los datos de entrada pertenecen al diseño físico.

- Parte lógica / Parte física del motor de SGBD

2.3 Diseño lógico

- Salida
  - Esquema procesable para el SGBD (reducir esfuerzo computacional)
  - Vistas externas y sus restricciones de seguridad
  - Especificaciones para el diseño físico
  - Guías para el diseño de programas.
  - Guías para las operaciones sobre la base de datos (requisitos, restricciones y operaciones).

2.3 Diseño lógico

- Pasos a seguir
  1. Correspondencia al modelo lógico.
  2. Diseño de los subesquemas.
  3. Diseño de programas.
  4. Revisión del diseño.

2.4 Diseño físico

- La última etapa de la metodología de diseño de bases de datos presentada es el **diseño físico**, cuyo objetivo general es "satisfacer los requisitos del sistema optimizando la relación costes/beneficios".

2.4 Diseño físico

- Objetivos
  - Disminuir los tiempos de respuesta
  - Minimizar el espacio de almacenamiento
  - Evitar las reorganizaciones periódicas
  - Proporcionar la máxima seguridad
  - Optimizar el consumo de recursos

2.4 Diseño físico

- Entrada
  - Lista de objetivos del diseño físico con sus correspondientes prioridades y cuantificación (a ser posible).
  - Esquema lógico específico.
  - Recursos de la máquina disponibles.
  - Recursos software disponibles (Sistema Operativo,...).
  - Información sobre las aplicaciones que utilizarán la base de datos.
  - **Políticas de seguridad de datos**.
- Salida
  - Estructura interna (esquema interno).
  - Especificaciones para el ajuste (tunning) de la base de datos.
  - Normas de seguridad.

2.4 Diseño físico

- El desarrollo de SGBD se afronta desde 3 vistas:
  A. El SGBD impone una estructura interna y deja muy poca flexibilidad al diseñador. Esto suele suponer una mayor independencia físico/lógica a costa de menor eficiencia.
  B. El Administrador de la BDD (DBA) diseña la estructura interna. Esto supone más trabajo y un perjuicio para la independencia de datos, aunque puede mejorar la eficiencia.
  C. El SGBD proporciona una estructura interna inicial a partir de algunos parámetros dados por el diseñador. El DBA puede ir afinándolos (tunning) para mejorar el rendimiento.

2.4 Diseño físico

- En general, la mejor opción es la C porque:
  - La base de datos puede empezar a funcionar inmediatamente.
  - La eficiencia va aumentando al ir realizando ajustes posteriores.
  - La independencia físico/lógica se mantiene.
  - Es la estrategia de la mayoría de los SGBD actuales y también la que mejor se adapta a la metodología propuesta.

2.4 Diseño físico

- Factores importantes dentro del diseño físico
  - Determinar los índices secundarios y sus características (compresión, orden, etc.).
  - Tipos de registros físicos.
  - Uso de punteros.
  - Direccionamiento calculado (Hashing).
  - Agrupamientos (Clustering) de tablas.
  - Bloqueos (Locking) y compresión de datos.

2.4 Diseño físico

- Factores importantes dentro del diseño físico
  - Definición de tamaños de memorias intermedias (Buffers).
  - Asignación de conjuntos de datos a particiones y/o a dispositivos físicos.
  - Redundancia de datos.

- No existe un modelo formal general para el diseño físico, sino que depende mucho de cada producto comercial concreto.

3. Modelo de Datos Entidad/Relación Extendido

3.1 Introducción

- Entre los MD conceptuales, destaca el Modelo Entidad/Interrelación (ME/R) [Chen 76].
- El ME/R puede ser usado como una base para una vista unificada de los datos, adoptando el enfoque natural del mundo real que consiste en entidades e interrelaciones.
- El modelo ha sido enriquecido por distintos autores dando lugar a distintas variantes, entre las que destaca el Modelo E/R extendido.
- El ME/R ha tenido una gran difusión en la comunidad informática dedicada a las bases de datos. Prueba de ello es que ha sido el modelo más extendido en las herramientas CASE de ayuda al diseño de bases de datos.

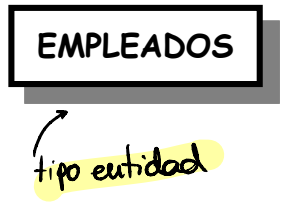
3.2 Modelo Entidad/Relación

- Objetos del ME/R:
  - Entidad (Entity)
  - Relación (Relationship)
  - Dominio (Domain) -> Rango valores de atributos
  - Atributo (Attribute)

Operaciones HO – O – O

Consultas → SGBD

3.2 Modelo Entidad/Relación
Objetos: Entidad

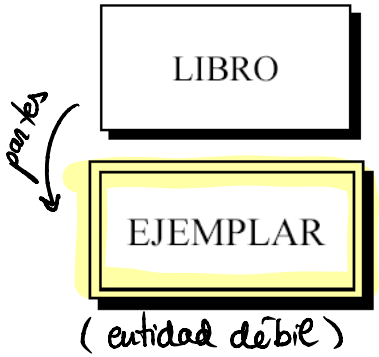


- "*Cualquier objeto (real o abstracto) que existe en la realidad y acerca del cual queremos almacenar información en la base de datos*". (**DOO: objeto = instancia de una clase**)
- Las entidades similares se agrupan dando lugar a **tipos de entidades**. (DOO: clases)
  - Ej. Los empleados de una empresa se podrían agrupar en una entidad llamada Empleados.
- Las entidades (también denominadas ejemplares) serán por lo tanto ocurrencias de los **tipos de entidades**.

3.2 Modelo Entidad/Relación
Objetos: Entidad

- En otra dimensión, también debemos distinguir entre:
  - La extensión o conjunto de ejemplares de un tipo de entidad en un momento dado;
  - La intensión que es el tipo de entidad propiamente dicho. Chen le llamó **conjunto de entidades** (entityset).
- Una entidad pertenece a un tipo de entidad si cumple el predicado asociado a ese tipo de entidad.
  - Matemáticamente, un conjunto de ejemplares de un tipo de entidad se define como:
    - E={ e : p(e) }
  - siendo e un ejemplar del tipo de entidad E y p el predicado asociado a E.
  - Ejemplo: el tipo de entidad PROFESOR, cuyo predicado asociado es "Persona que ejerce o enseña una materia o arte" tiene un ejemplar "Sánchez" que pertenece a él si cumple dicho predicado.

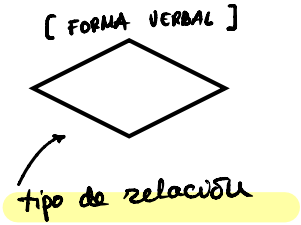
3.2 Modelo Entidad/Relación
Objetos: Entidad



- Tipos de entidades
  - **Fuertes:** son aquellas cuyas entidades (ejemplares) tienen existencia por sí mismos (como **LIBRO** y **AUTOR**).
  - **Débiles** (DOO: asociación entre clases de tipo **composición**)**:** en las cuales la existencia de una entidad (ejemplar) depende de que exista una entidad dada de otro tipo de entidad:

    Ej. **EJEMPLAR** (cada uno de los ejemplares de determinado libro) depende de **LIBRO**, y por tanto, la desaparición de un determinado libro de la base de datos hace que desaparezcan también todos los ejemplares de dicho libro.

3.2 Modelo Entidad/Relación
Objetos: Relación

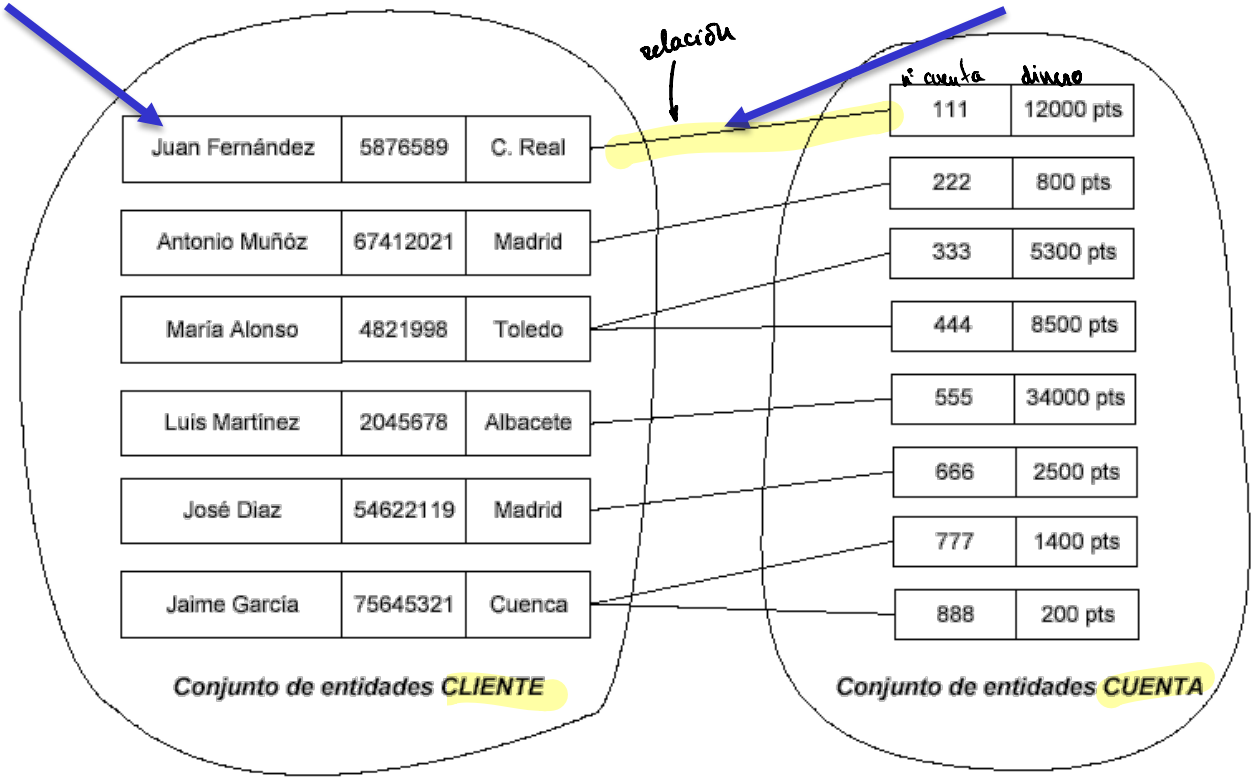


- Es una asociación, vinculación o correspondencia entre entidades. (DOO: se corresponde con una relación entre clases de los tipos: asociación, agregación y composición)
- Los **tipos de relaciones** describen conjuntos de relaciones, de forma análoga a como sucede con las entidades.
- Una relación será por lo tanto una ocurrencia de un tipo de relación.

3.2 Modelo Entidad/Relación
Objetos: Relación

Ej. de entidad: ***Juan Fernández***

Ej. de relación entre: ***Juan Fernández*** **y su cuenta** ***111***

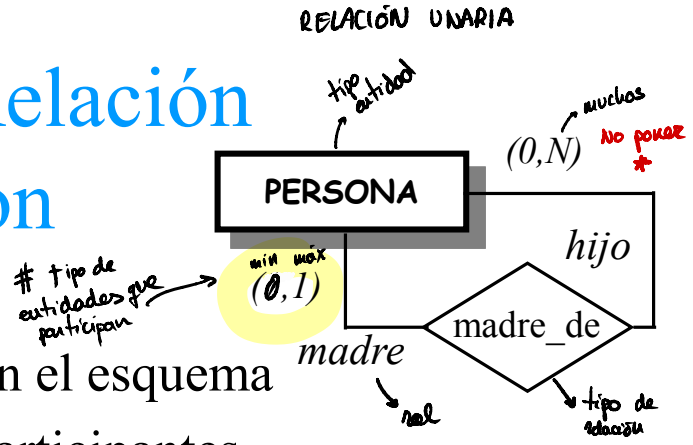


Tipo de entidad: ***CLIENTE***

Tipo de entidad: ***CUENTA***

Tipo de relación: **posee** -> 

3.2 Modelo Entidad/Relación
Objetos: Relación

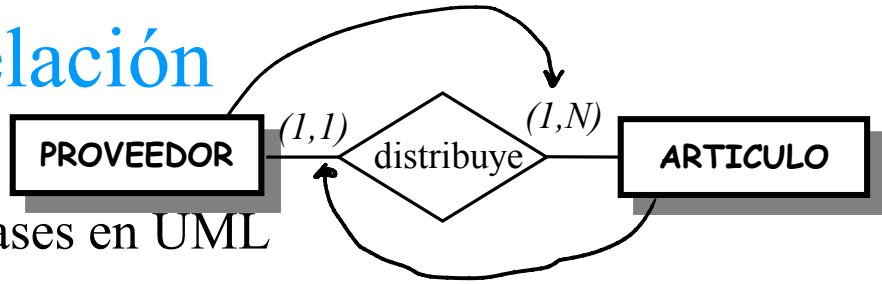


- Elementos de una relación
  - **Nombre** (madre_de): identificador único en el esquema
  - **Grado** (1): número de tipos de entidades participantes
  - **Tipo de correspondencia o conectividad de la relación** (1:N):

    1:1, 1-muchos (1:N), muchos a muchos (N:M), ...
  - **Papel (Rol)** (madre-hijo): función que desempeña cada tipo de entidad participante (importante indicarlo en las relaciones reflexivas)
  - **Clase de Pertenencia** (opcional): Especifica si cada ocurrencia de la entidad debe participar en la relación o es posible que no.

{ -> obligatorio : min 1
-> opcional : min 0

3.2 Modelo Entidad/Relación
Objetos: Relación



- Cardinalidad
  - Se lee como una asociación entre clases en UML
    - "Una entidad proveedor distribuye de 1 a muchos entidades artículo"
    - "Una entidad artículo es distribuido por un único entidad proveedor"
  - Si representamos las relaciones (I) entre proveedores (E_1) y artículos (E_2) mediante una tabla, necesitaríamos dos columnas con los identificadores principales de ambos tipos de entidades.
    - Esto se puede escribir como: $I(E_1(1,N):E_2(1,N))$ y se puede leer como:

Toda relación de I relaciona una entidad de E_1 (en la tabla se representa como uno de los valores de prov_id) con una o varias entidades de E_2 (en la tabla serían cada uno de los valores de art_id). Lo mismo se puede decir cuando se lee en sentido contrario de E_2 a E_1. Por otro lado, un artículo sólo puede relacionarse con un proveedor (un único valor de prov_id).

Valores posibles

Imposible añadir

Prov_id	Art_id
PA	A1
PA	A2
PB	A2
PB	A 3

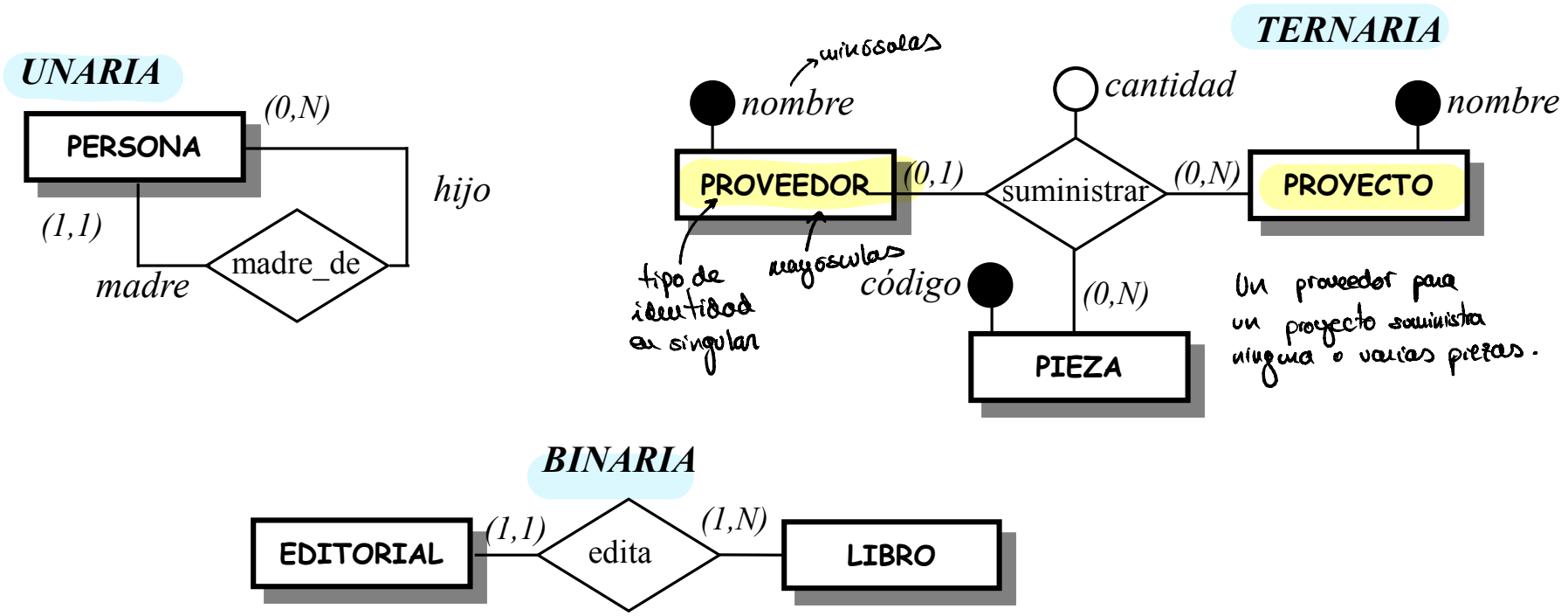
X

√√

Un artículo es distribuido por 1 y sólo 1 proveedor

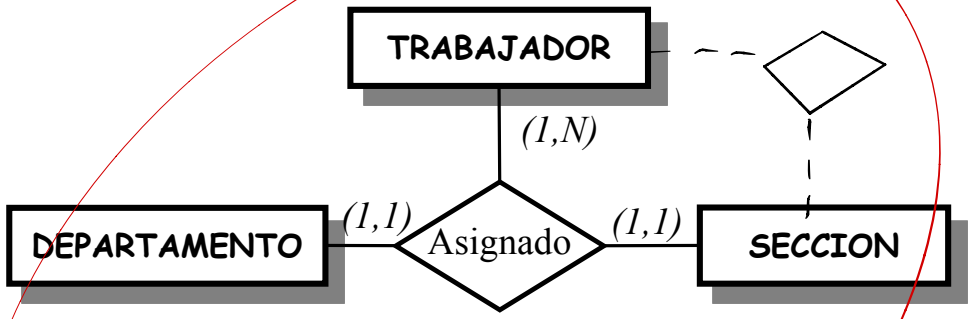
3.2 Modelo Entidad/Relación
Objetos: Relación

- Elementos de una relación
  - **Grado**: número de tipos de entidades participantes. Se utilizarán relaciones unarias (reflexivas), binarias y ternarias.



3.2 Modelo Entidad/Relación
Objetos: Relación

- Cardinalidad n-aria
  - Si representamos las relaciones (I) entre trabajador (E_1), departamento (E_2) y sección (E_3) mediante una tabla, necesitaríamos tres columnas con los identificadores principales de los tipos de entidades.
    - Esto se puede escribir como: $I(E_1(1,N):E_2(1,1):E_3(1,1))$ y se puede leer como:



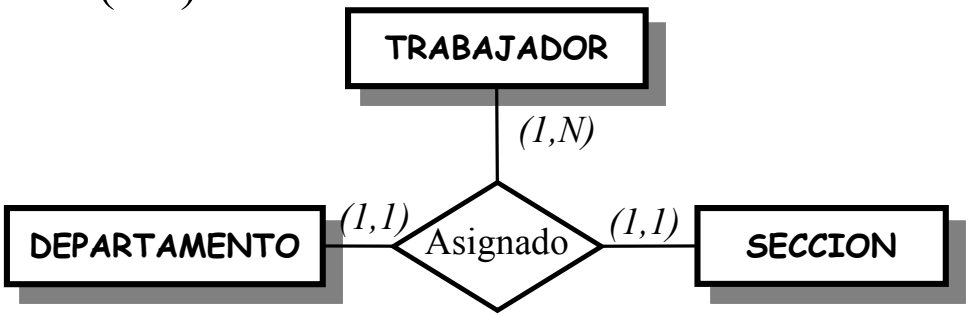
Toda relación de I relaciona una entidad de E_1 (en la tabla se representa como uno de los valores de tr_id) con una única entidad de E_2 (en la tabla serían cada uno de los valores de dp_id) y una única entidad de E_3 (se_id).

Valores posibles →

Tr-id	Dp_id	Se_id
PA	D1	S1
PA	D2	S2
PB	D2	S3

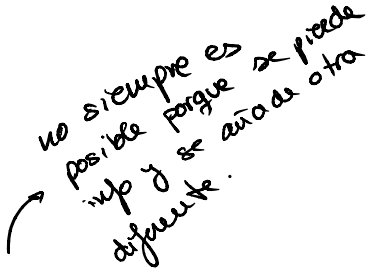
3.2 Modelo Entidad/Relación
Objetos: Relación

- Cardinalidad n-aria
  - Esto significa que es imposible añadir dos relaciones un mismo trabajador (PA) en dos secciones diferentes (S1 y S3) con el mismo departamento (D1).



	Tr-id	Dp_id	Se_id
Valores posibles	TA	D1	S1
	TA	D2	S2
	TB	D2	S3
Imposible añadir X	TA	D1	S3

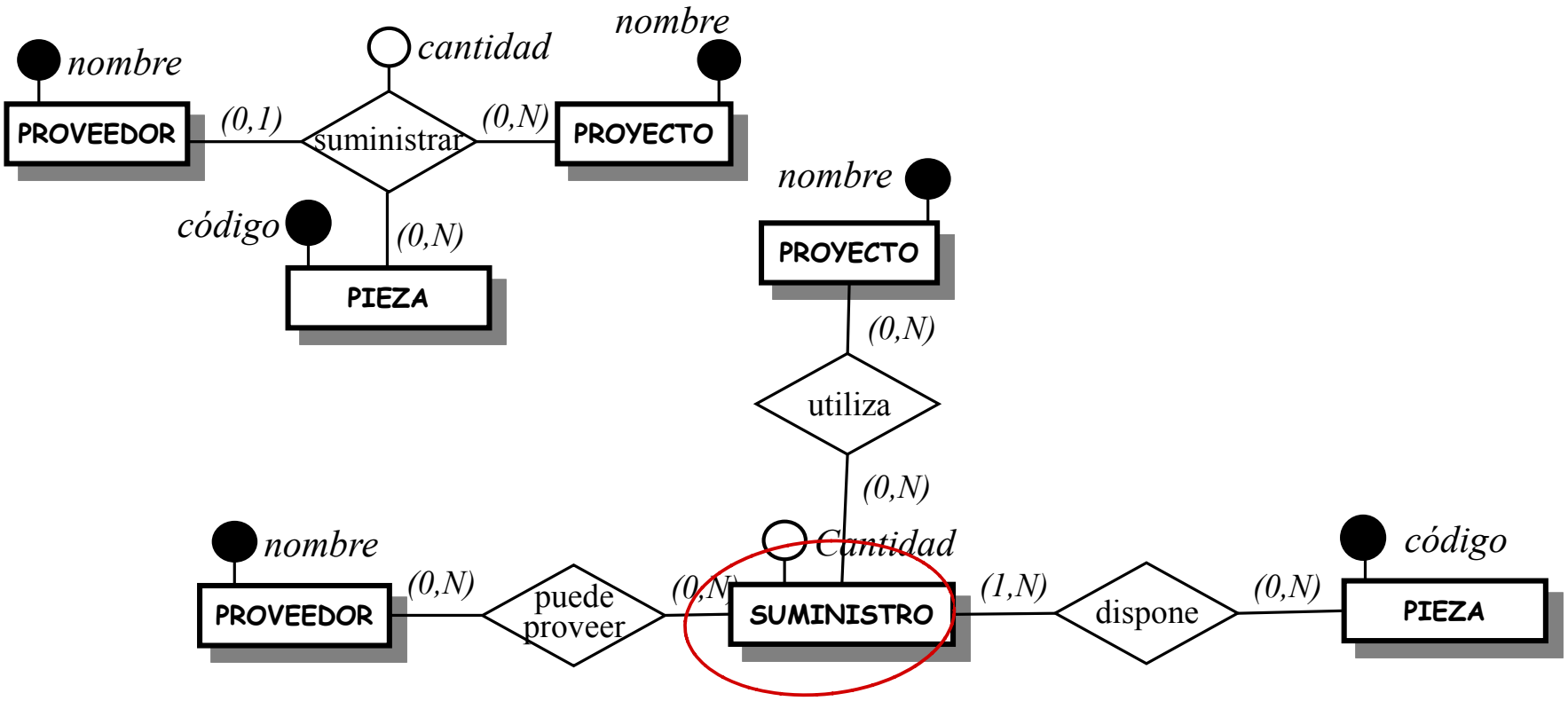
3.2 Modelo Entidad/Relación
Objetos: Relación



- En las relaciones n-arias, se suele transformar a relaciones binarias. Esta transformación permite más flexibilidad en el modelo, sobre todo para contemplar relaciones entre dos entidades.
- Pasos:
  - Transformar la relación en una entidad
  - Crear nuevas relaciones entre las entidades existentes y la nueva entidad
  - Adecuar las restricciones de las relaciones n-arias a las binarias

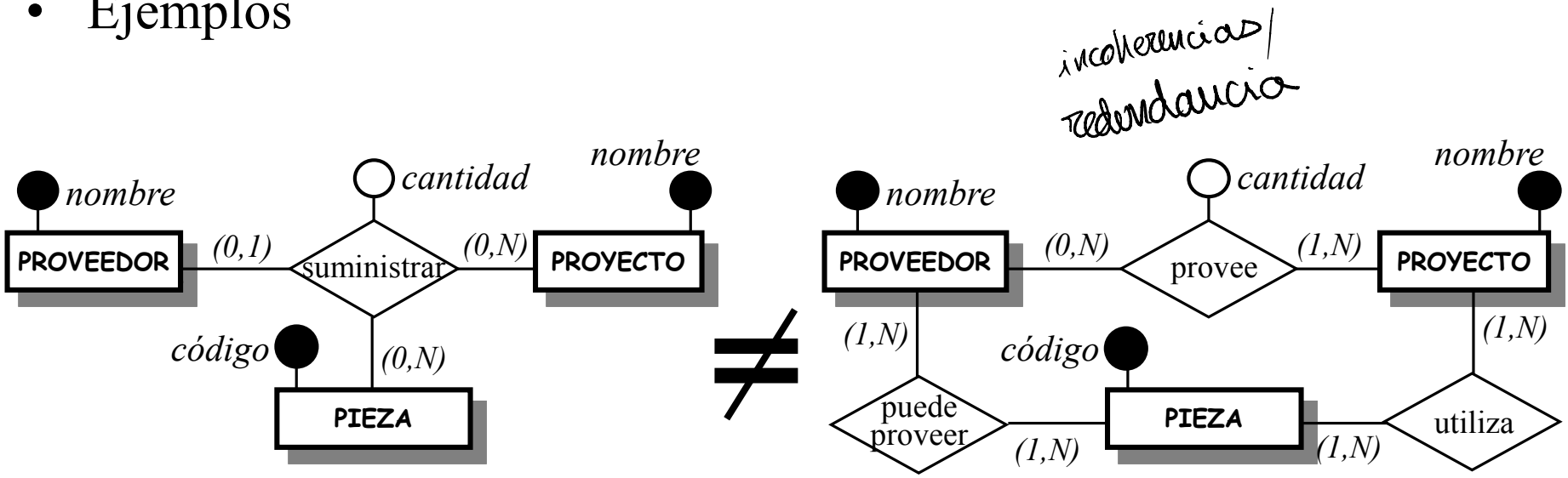
3.2 Modelo Entidad/Relación
Objetos: Relación

- En el ejemplo del proveedor, podríamos tener las piezas que provee, aunque no se hayan comprado en algún proyecto, sin necesidad de nulos



3.2 Modelo Entidad/Relación
Objetos: Relación

- Ejemplos

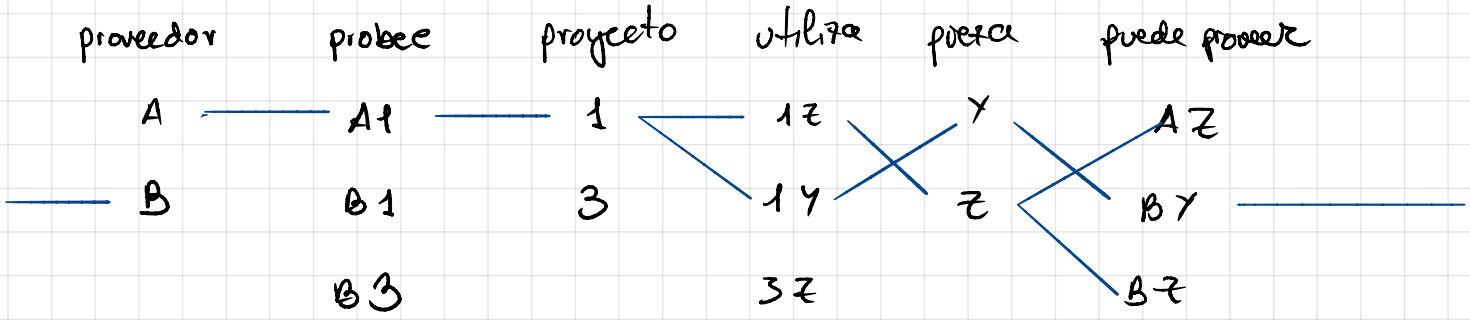


Proveedor	Proyecto	Pieza
A	1	Z
B	1	Y
B	3	Z

¿por qué?

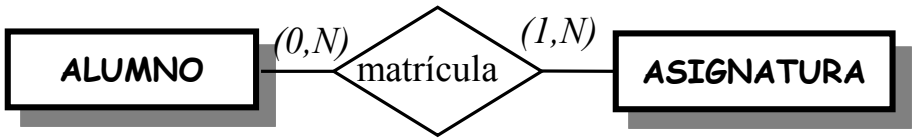
En la transformación, se podría dar:

- el caso "A1Y", que no está contemplado en la relación suministrar
- No saber qué proveedor suministró la piezas usadas en un proyecto


proveedor
provee
proyecto
utiliza
pieza
puede proveer
A
B
A1
B1
B3
1
3
1Z
1Y
3Z
Y
Z
AZ
BY
BZ

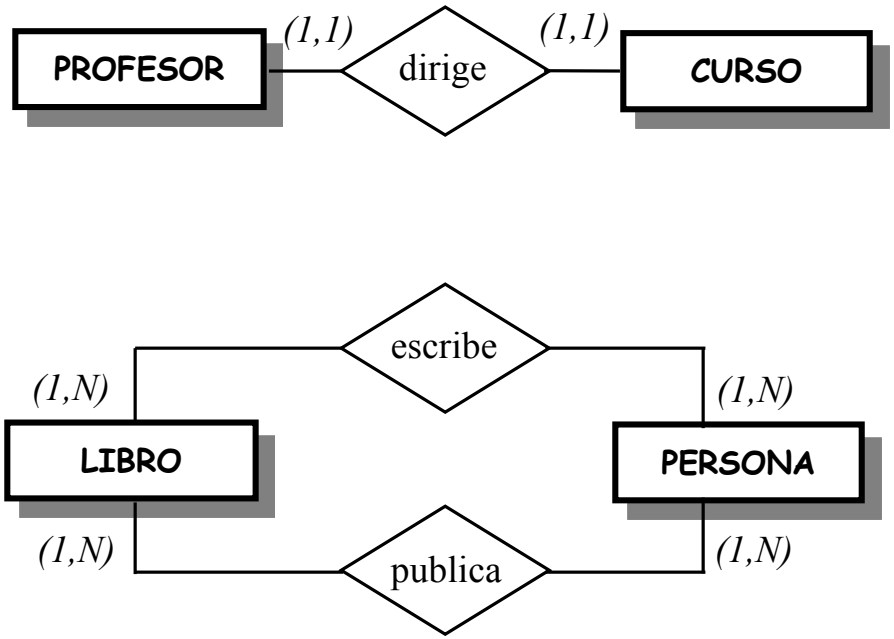
3.2 Modelo Entidad/Relación
Objetos: Relación

- Elementos de una relación
  - **Clase de Pertenencia**: Especifica si cada ocurrencia de una entidad debe participar obligatoriamente en la relación o es posible que no. Puede ser:
    - **Obligatoria**: Representada por un 1 antes de la conectividad (1,…), indica que cada ocurrencia de la entidad relacionada tiene al menos una ocurrencia de la entidad asociada.
    - **Opcional**: Representada por un 0 antes de la conectividad (0,…), indica que no es obligatoria, puede haber una ocurrencia de la entidad relacionada no asociada a una ocurrencia de la entidad.



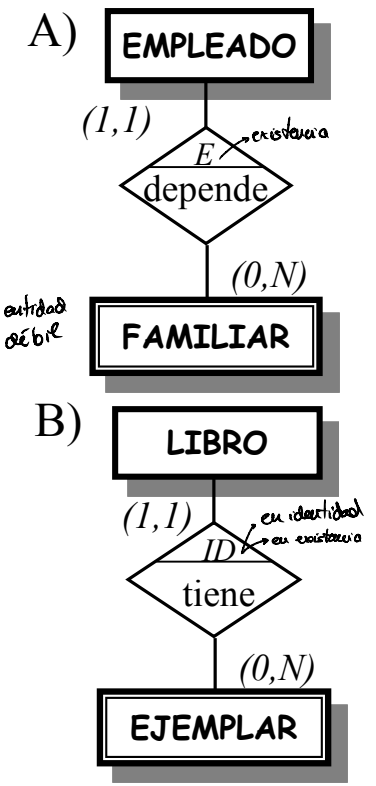
3.2 Modelo Entidad/Relación
Objetos: Relación

- Ejemplos:



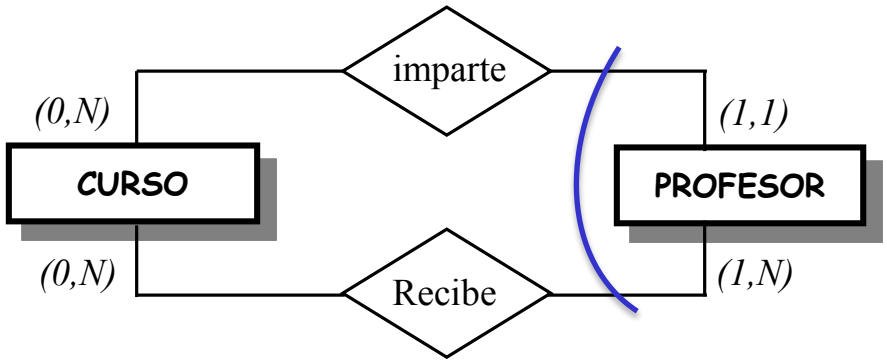
3.2 Modelo Entidad/Relación
Objetos: Relación

- Los **tipos de relación** dependiendo de los tipos de entidades que unen (débiles o fuertes) se pueden clasificar también en relaciones **fuertes y débiles**.
- Las **relaciones débiles** pueden ser:
  A. Dependencia en existencia: los ejemplares de una entidad débil no pueden existir si desaparece el ejemplar de la entidad fuerte del cual dependen.
  B. Dependencia en identificación: además de cumplir la condición anterior, los ejemplares de la entidad débil no pueden ser identificados sin recurrir a la inclusión del identificador principal de la entidad fuerte de la que dependen.



3.2 Modelo Entidad/Relación
Objetos: Relación

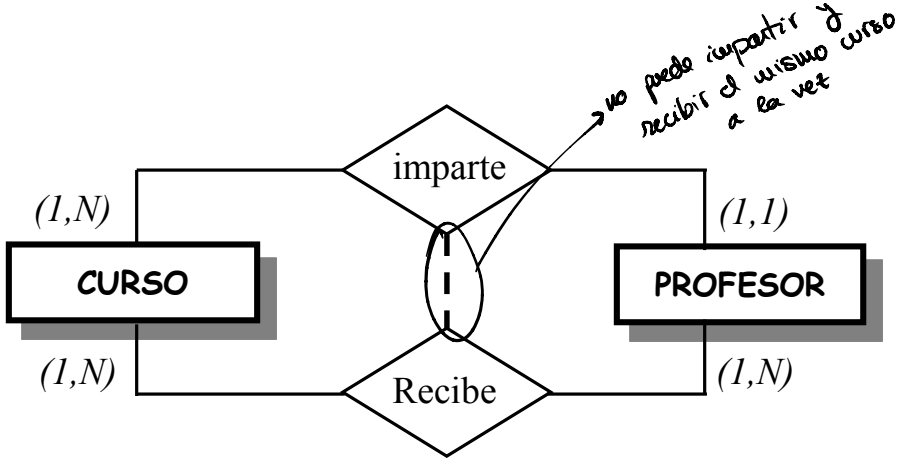
- Otras restricciones sobre relaciones
  - **Exclusividad (OR)**: dos o más relaciones tienen una restricción de exclusividad sobre una entidad que participa en todas ellas cuando una ocurrencia de esta entidad solo puede participar en uno de los tipos de relaciones.



Esta restricción no puede modelarse únicamente mediante restricciones de cardinalidad: no captura la restricción de que no podamos impartir y recibir cursos. De ahí que se recurra al símbolo de exclusividad.

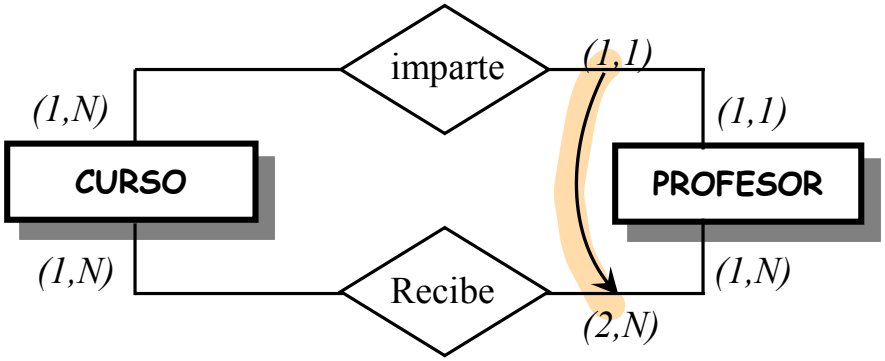
3.2 Modelo Entidad/Relación
Objetos: Relación

- Otras restricciones sobre relaciones
  - **Exclusión**: dos o más relaciones tienen una restricción de exclusión sobre una entidad que participa en todas ellas cuando una ocurrencia de esta entidad puede participar en ambas relaciones pero no simultáneamente sobre la misma ocurrencia (ejemplar) de la otra entidad.



3.2 Modelo Entidad/Relación
Objetos: Relación

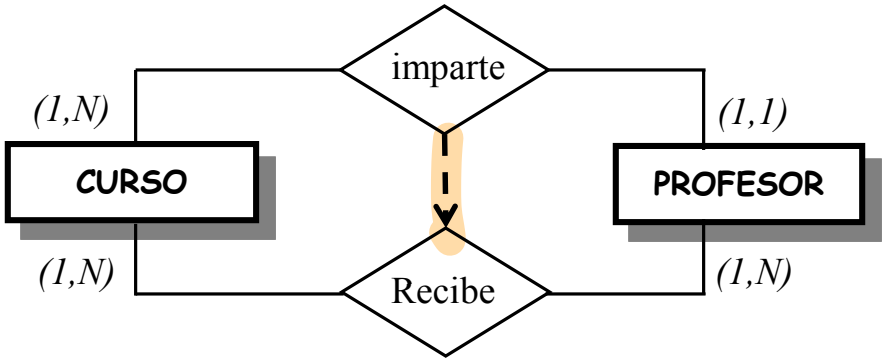
- Otras restricciones sobre relaciones
  - **Inclusividad**: dos o más relaciones tienen una restricción de inclusividad sobre una entidad que participa en todas ellas cuando una ocurrencia de esta entidad debe participar en todas las relaciones.



Para que un profesor imparta un curso debe haber recibido al menos dos cursos, aunque no tiene por qué ser el mismo.

3.2 Modelo Entidad/Relación
Objetos: Relación

- Otras restricciones sobre relaciones
  - **Inclusión**: dos relaciones tienen una restricción de inclusión sobre una entidad que participa en ambas cuando una ocurrencia de esta entidad debe participar en las dos relaciones sobre la misma ocurrencia de la entidad relacionada



Para que un profesor imparta un curso debe haber recibido al menos ese curso.

3.2 Modelo Entidad/Relación
Objetos: Dominio

→ conjunto de valores de un predicado

- se define como un conjunto de valores homogéneos (todos del mismo tipo) y atómicos (indivisibles) con un nombre, que lo identifica.
- Una cierta característica o propiedad de un objeto toma valores que pertenecen a un determinado dominio.
- Un dominio lleva siempre asociado un predicado que permite comprobar si un determinado valor pertenece al dominio.

$$D=\{v_i : p(v_i)\}$$

donde D es el dominio, v_i es el valor y p es el predicado asociado al dominio.

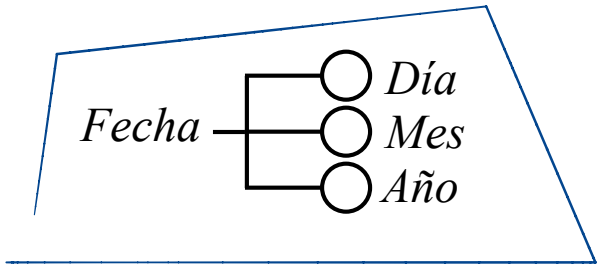
clase expositiva

3.2 Modelo Entidad/Relación
Objetos: Dominio

- Todo dominio se puede definir por
  - **Intensión**: el predicado se implementa mediante reglas/restricciones al dominio subyacente. ej. Todas las edades de personas activas como un campo entero de longitud dos, comprendido entre 18 y 65 años. Sin embargo, el dominio de los países no podría ser todo las cadenas de caracteres de 10 letras.
  - **Extensión**: listado de valores válidos. Ej. Dominio de países.

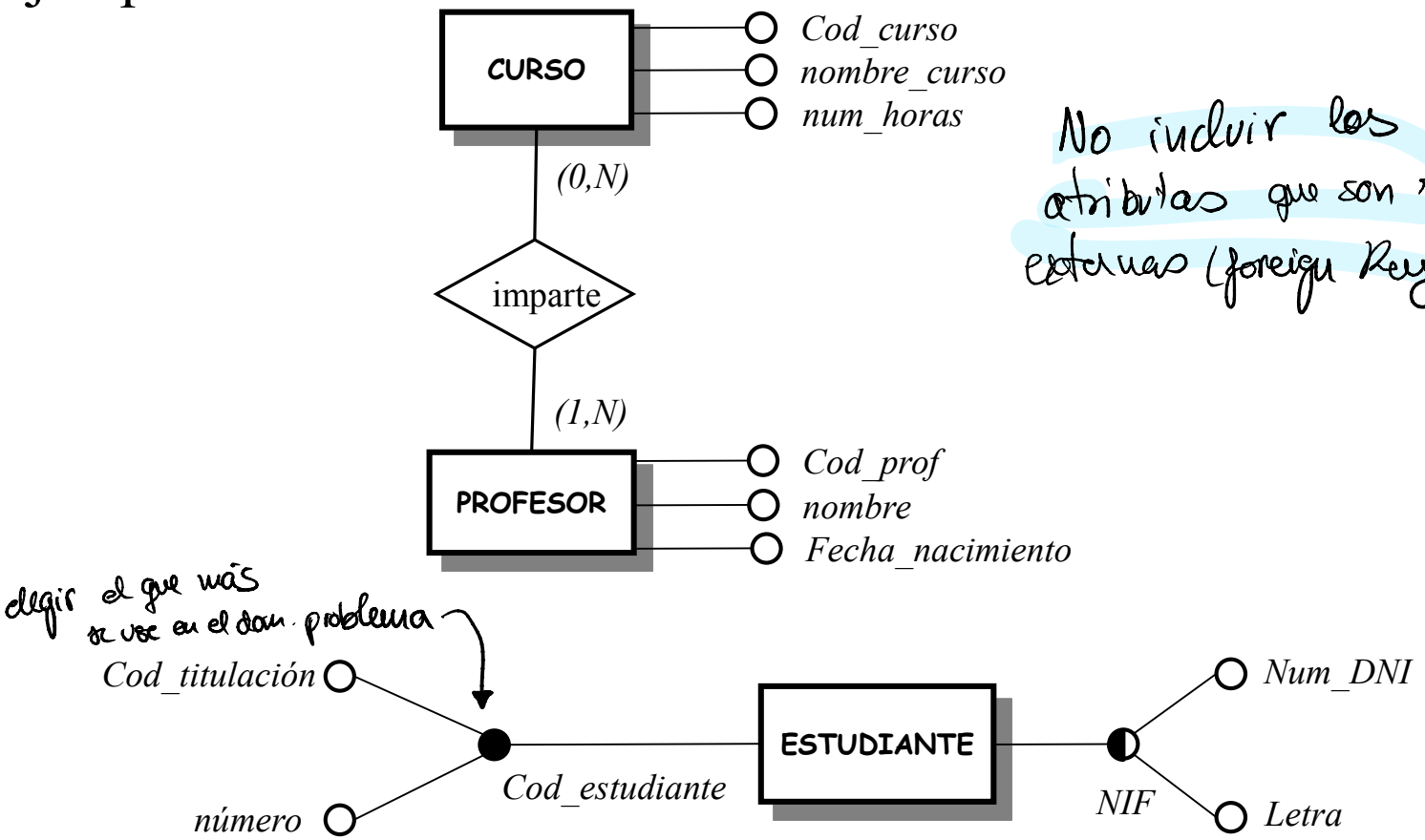
3.2 Modelo Entidad/Relación
Objetos: Atributo

- cada una de las propiedades o características que tiene un tipo de entidad (Ei) o un tipo de relación (Rj)
- formalmente, es:
  - A1: Ei $\longrightarrow$ D (D1xD2x.....xDn)
  - A2: Rj $\longrightarrow$ D (D1xD2x.....xDn)
  - Donde, para cada atributo existe un conjunto de valores permitidos (dominio).
  - puede ser un conjunto de dominios para el caso de los atributos compuestos (combinación de dominios simples).



3.2 Modelo Entidad/Relación
Objetos: Atributo

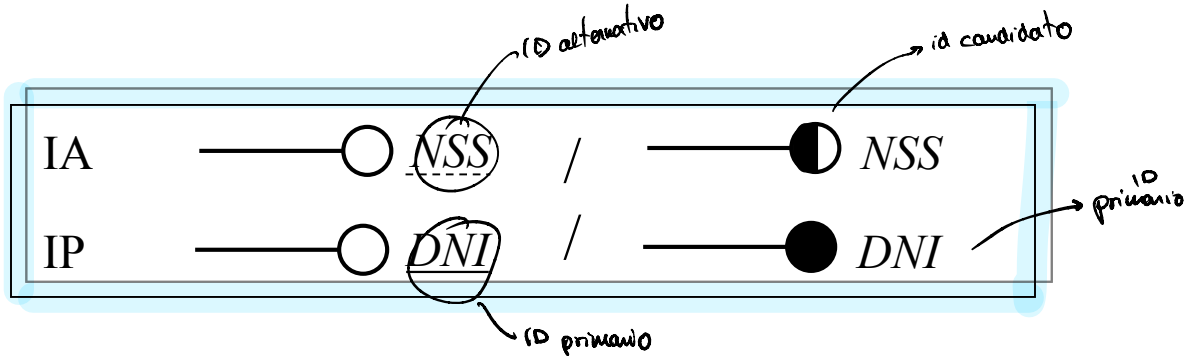
- Ejemplo



No incluir los atributos que son referencias externas (foreign Key)

3.2 Modelo Entidad/Relación
Objetos: Atributo

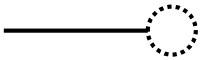
- Dentro del conjunto de atributos de una entidad debe existir uno o varios subconjuntos de esos atributos que identifiquen unívocamente esa entidad. Estos subconjuntos se denominan **identificadores candidatos**. → identifica 1 fila
- De entre los identificadores candidatos se elegirá uno que será el **identificador principal (IP)**, mientras que el resto serán **identificadores alternativos (IA).** → not null and unique



3.2 Modelo Entidad/Relación
Objetos: Atributo

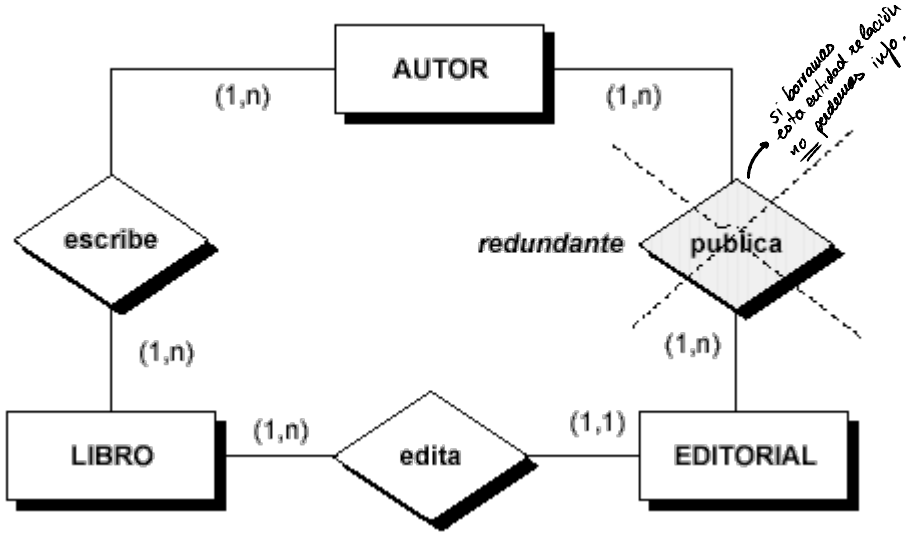
- Atributos multievaluados y unievaluados
  - Multievaluados: Pueden tomar más de un valor a la vez. (una persona puede tener más de un número de teléfono)
  - Unievaluados: toman un solo valor.
- Atributos opcionales y obligatorios
  - Obligatorio: se obliga a que ese atributo tome un valor (distinto de nulo) dentro del dominio de ese atributo.
  - Opcional: el atributo puede tomar un valor dentro del dominio o el valor nulo.

3.2 Modelo Entidad/Relación
Redundancia en el esquema

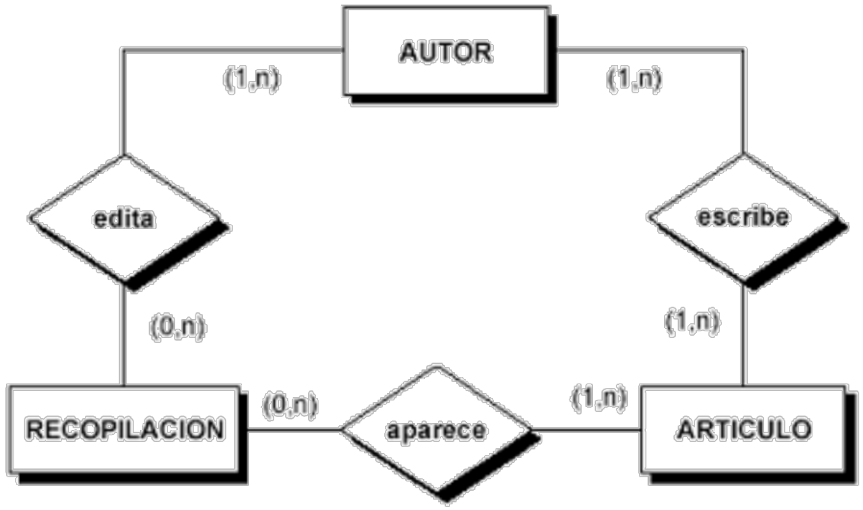
- Un **elemento** de un esquema es **redundante** si puede ser eliminado sin pérdida de semántica.
- Existen dos formas principales de redundancia:
  - En los **atributos** (atributos derivados o calculados): aunque son redundantes, no dan lugar a inconsistencias siempre que en el esquema se indique su condición de derivados y la fórmula mediante la que han de ser calculados. 

  - En las **relaciones** (también llamadas relaciones derivadas): una relación es redundante si su eliminación no implica pérdida de semántica porque existe la posibilidad de realizar la misma asociación de ejemplares por medio de otras relaciones.
    - Para ello es condición necesaria pero no suficiente, que la relación redundante forme parte de un ciclo => Hay que estudiar detenidamente los ciclos en los diagrama E/R.

3.2 Modelo Entidad/Relación
Redundancia en el esquema



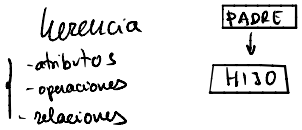
(a) Esquema con relación redundante.



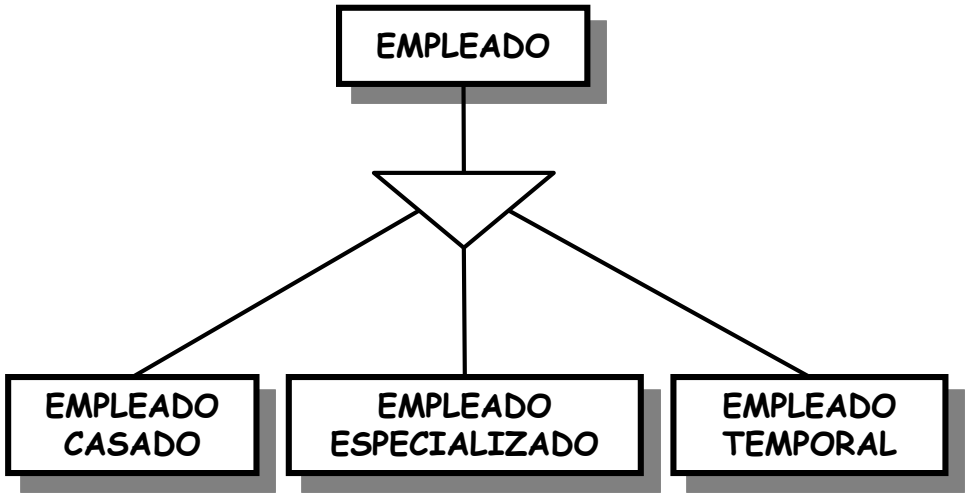
(b) Esquema sin redundancia.

3.2 Modelo Entidad/Relación
Relaciones Jerárquicas. Generalizaciones

- Proporcionan un mecanismo de abstracción que permite especializar una entidad (*supertipo*) en *subtipos* o *subentidades*, o bien, *generalizar* los *subtipos* en el *supertipo*.



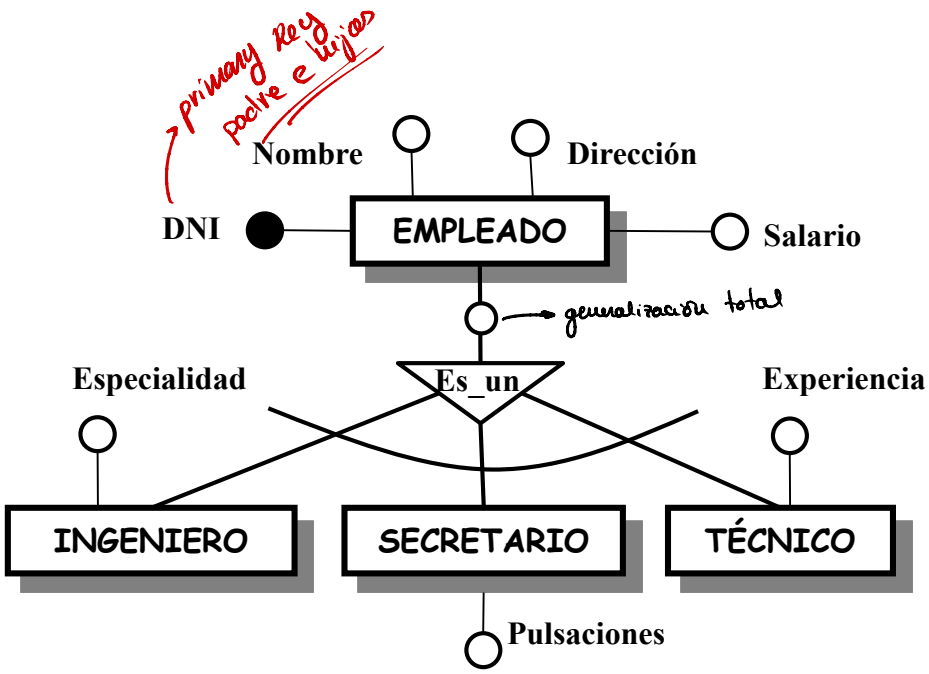
Ej: La entidad EMPLEADO puede incluir "empleados casados", "empleados especializados" o "empleados temporales".



3.2 Modelo Entidad/Relación
Relaciones Jerárquicas. Generalizaciones

- Se pueden identificar generalizaciones si encontramos una serie de atributos comunes a un conjunto de entidades. Los atributos comunes describen al supertipo y los específicos permanecen en los subtipos.

Pueden existir subtipos sin atributos propios si estos participan, por sí mismos, en relaciones con otras entidades.

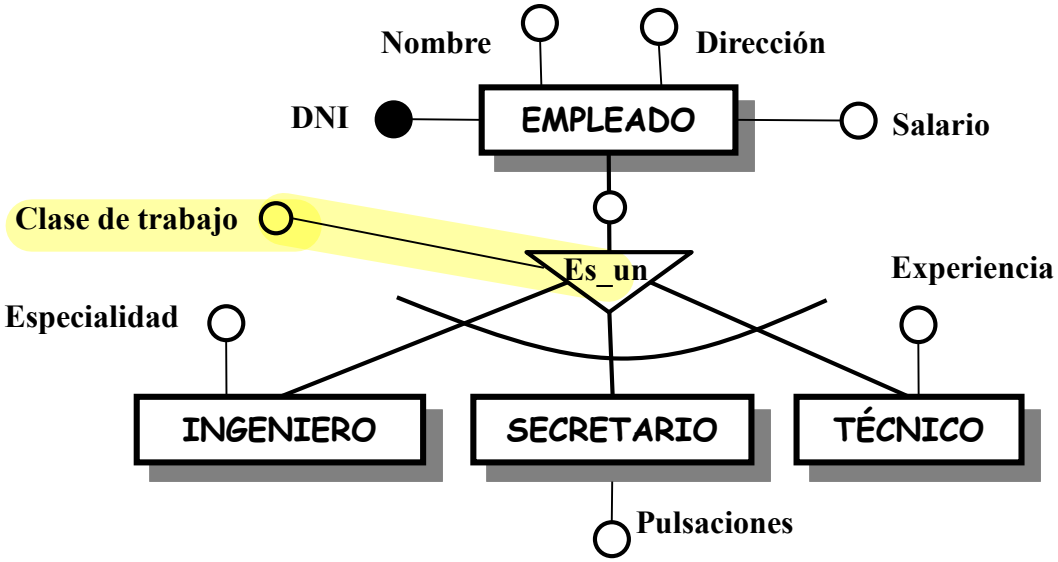


3.2 Modelo Entidad/Relación
Relaciones Jerárquicas. Tipos

- En las generalizaciones se identifican restricciones semánticas como son la totalidad /parcialidad y la exclusividad /solapamiento.
  - **Generalización total**: Las ocurrencias de los subtipos cubren al supertipo (no hay ocurrencias en el supertipo que no pertenezcan a ninguno de los subtipos) y se representa con un círculo (blanco) sobre el conector del supertipo. En otro caso es **Generalización parcial** y no lleva círculo.
  - **Generalización con solapamiento**: Pueden existir ocurrencias que pertenezcan a más de un subtipo. Si los subtipos son disjuntos se denomina **Generalización exclusiva** y se representa con un arco sobre los conectores de los subtipos.

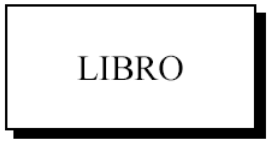
3.2 Modelo Entidad/Relación
Relaciones Jerárquicas. Tipos

- Cuando la división en subtipos (especialización) venga determinada por una **condición** predefinida (ej. Los valores de un atributo), vendrá representada como un atributo asociado ("clase de trabajo") al triángulo que representa la interrelación.



3.2 Modelo Entidad/Relación
Resumen notación gráfica

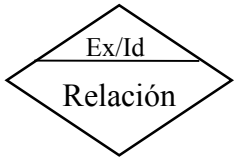
- Entidad fuerte



- Entidad débil

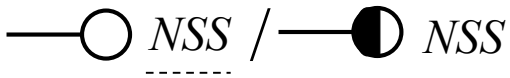
- Relación

- Relación con dependencia de Exist. o Id.

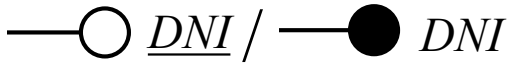


- Atributo

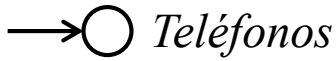
- Atributo *(identificador alternativo)*



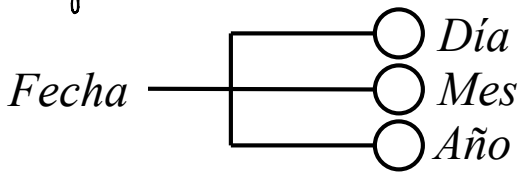
- Atributo *(identificador principal)*



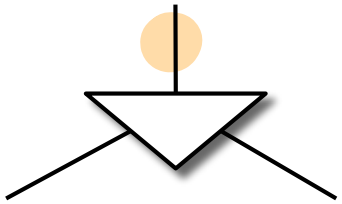
- Atributo multivaluado



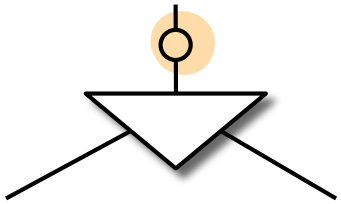
- Atributo *(compuesto)*



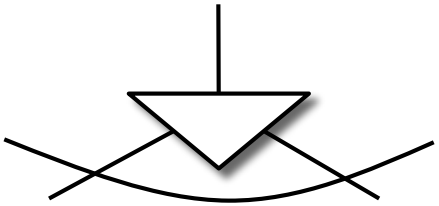
3.2 Modelo Entidad/Relación
Resumen notación gráfica



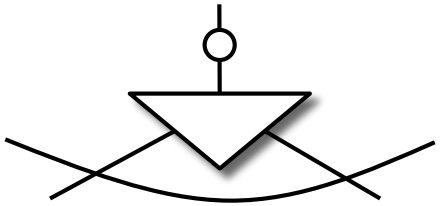
• Jerarquía solapada y parcial



• Jerarquía solapada y total

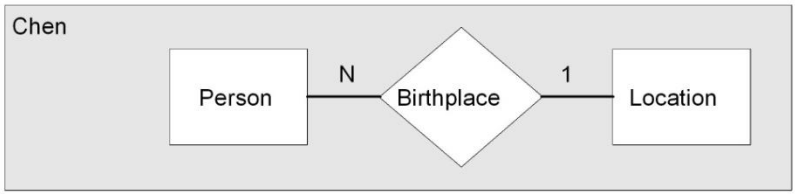


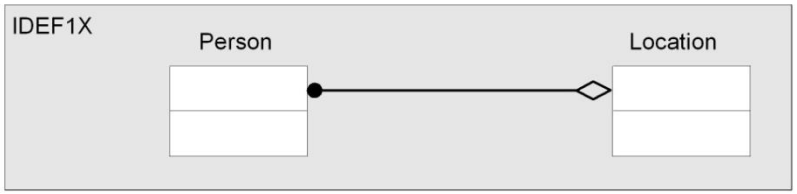
• Jerarquía exclusiva y parcial

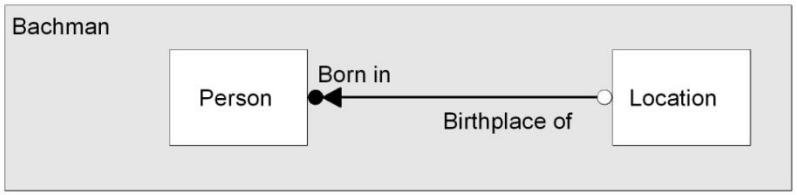


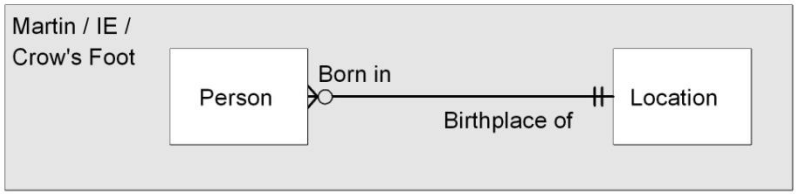
• Jerarquía exclusiva y total

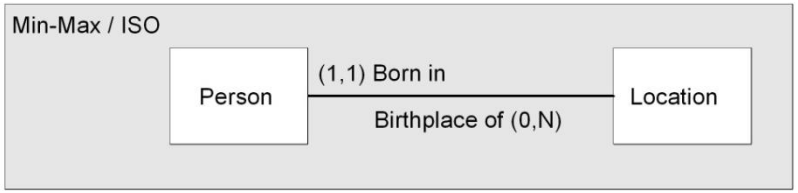
3.2 Modelo Entidad/Relación
Otras notación gráfica











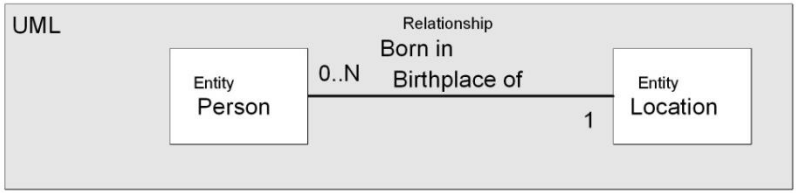


Diagrama de clases -> Entidad Relación

3.2 Modelo Entidad/Relación

Diseño del diagrama entidad/relación extendido

1) Clasificación de entidades y atributos

a) Si existe información descriptiva sobre un objeto, el objeto debe ser clasificado como ENTIDAD. Si un objeto necesita sólo de un identificador, debe ser clasificado como ATRIBUTO.

b) Si a un valor de un identificador le corresponde más de un valor de un descriptor (atributo multievaluado), entonces el descriptor se clasifica como ENTIDAD, incluso si no tiene descriptores.

c) Si un descriptor de una entidad tiene una relación N:1 con otra entidad el descriptor debe ser clasificado como ENTIDAD, incluso si no tiene descriptores.

d) Asignar los atributos a las entidades que describen más directamente.

3.2 Modelo Entidad/Relación
Diseño del diagrama entidad/relación extendido

1) Clasificación de entidades y atributos

e) Evitar, en lo posible, los identificadores compuestos:

- Si los componentes del identificador compuesto son identificadores de otras entidades, se elimina la entidad. El objeto puede definirse como una relación.
- Sino, eliminar la entidad y definir nuevas entidades con los componentes del identificador compuesto para después definir una relación.
- Mantener la entidad con el identificador compuesto si es razonablemente natural.

3.2 Modelo Entidad/Relación
Diseño del diagrama entidad/relación extendido

2) Identificar jerarquías

- Si existe una jerarquía subconjunto o generalización entre entidades, reasignamos los atributos a las entidades adecuadas.
  - **Entidad Genérica**: Identificador, Descriptores genéricos.
  - **Entidades Subconjunto/Especialización**: Identificador, Descriptores específicos.

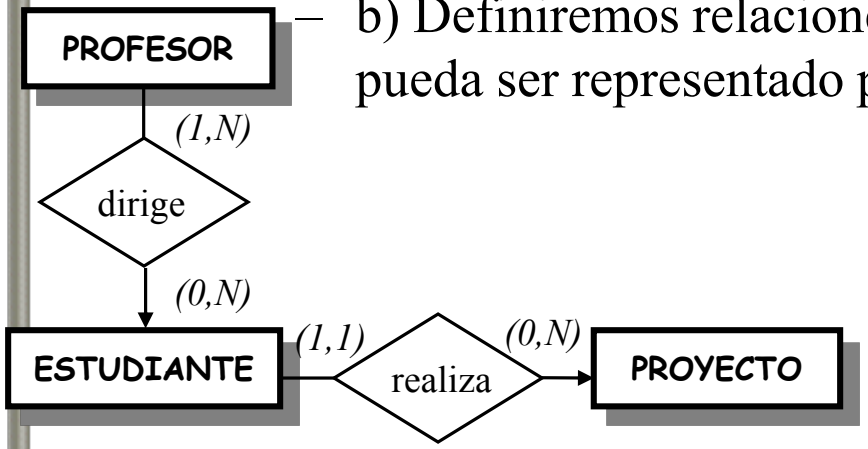
[generalización / especificación]

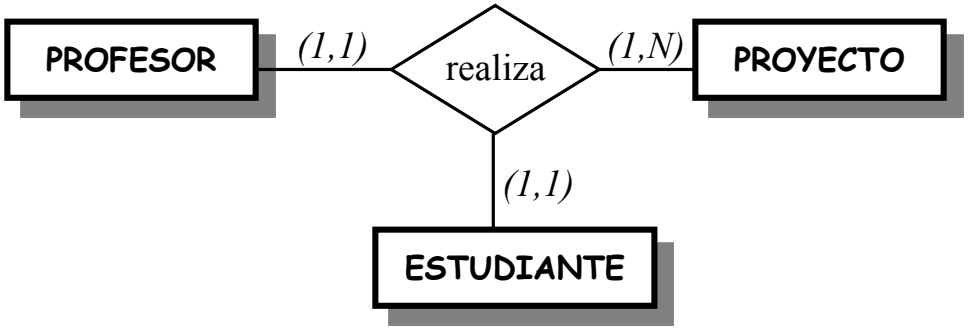
3.2 Modelo Entidad/Relación
Diseño del diagrama entidad/relación extendido

3) Definir relaciones

—> relaciones binarias

- Objetos que representan asociaciones entre objetos.
- Especificar: grado, conectividad, clase de pertenencia, atributos.
- a) Eliminación de relaciones redundantes (2 o más relaciones que representan el mismo concepto)
- b) Definiremos relaciones ternarias sólo cuando el concepto no pueda ser representado por varias binarias.





¿qué proyecto dirige cada profesor a cada estudiante?

3.2 Modelo Entidad/Relación
Diseño del diagrama entidad/relación extendido

4) Integración de vistas

- Cuando se producen múltiples vistas de datos y relaciones, éstas deben consolidarse dentro de una vista global para eliminar redundancias e inconsistencias.

3.2 Modelo Entidad/Relación
Diseño del diagrama entidad/relación extendido

Ejemplo

- Queremos construir una base de datos para una empresa de proyectos de ingeniería. Se necesita guardar información sobre todos los empleados, sus habilidades, proyectos asignados y departamentos que trabajan en ellos. Para ello se dispone de la siguiente información:
  - Cada empleado tiene asignado un número único. Se necesita su nombre y fecha_de_nacimiento.
  - Solamente si un empleado está casado con otro se necesita almacenar la fecha de la boda y con quién está casado (en caso de que el conyuge sea de la empresa).
  - Cada empleado tiene un título (ingeniero, secretario, conserje,..).
  - De los ingenieros necesitamos conocer su especialidad (eléctrico, mecánico, etc.),

3.2 Modelo Entidad/Relación
Diseño del diagrama entidad/relación extendido

Ejemplo

- y de los secretarios el número de pulsaciones.
- Un empleado realiza sólo un tipo de trabajo en un momento dado, por lo que sólo necesitamos información del trabajo que realiza en la actualidad.
- Los departamentos en que se divide la empresa tienen un nombre único. Necesitamos conocer su número de teléfono.
- Un empleado pertenece a un solo departamento.
- Para equiparse, cada departamento trata con muchos proveedores y un proveedor puede tratar con más de un departamento.
- De los proveedores se necesita conocer su dirección y la fecha del último encuentro con cada uno de los departamentos con los que trata.

3.2 Modelo Entidad/Relación
Diseño del diagrama entidad/relación extendido

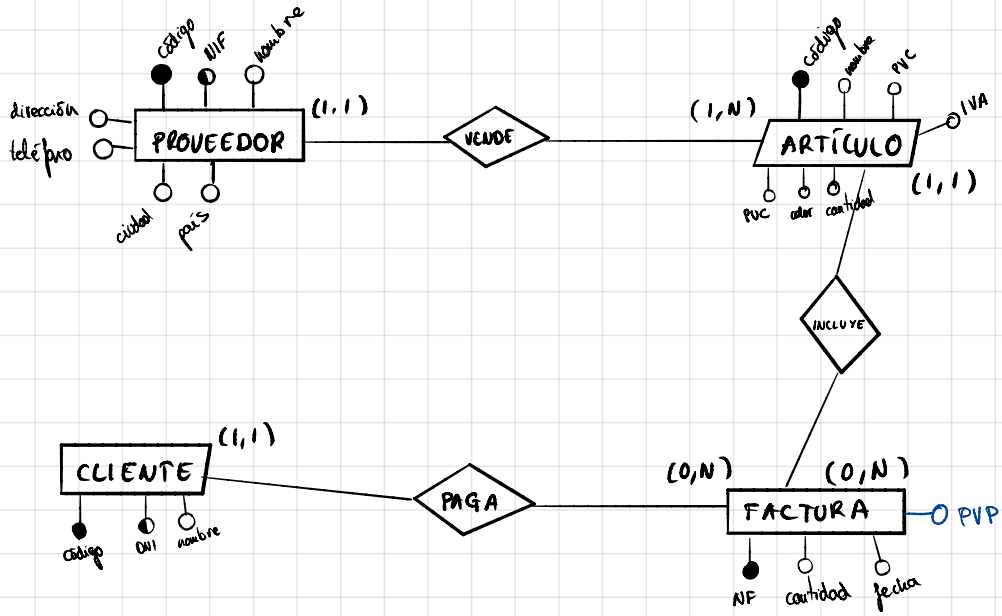
Ejemplo

- En un proyecto pueden trabajar muchos empleados y un empleado puede estar ocupado en más de un proyecto.
- Sin embargo, en una localidad, el empleado sólo puede tener asignado un proyecto.
- Para cada localidad estamos interesados en el número de habitantes y la provincia a la que pertenece.
- Cada empleado puede tener varias habilidades, pero para un determinado proyecto sólo puede usar una serie de ellas.
- Un empleado usa cada habilidad que posee en al menos un proyecto.
- Cada habilidad tiene un número asociado y necesitamos una breve descripción de ella.
- Por último, para cada proyecto necesitamos el coste estimado.

EJERCICIO 1

PROVEEDOR : código , NIF, nombre , dirección, teléfono , ciudad país

ARTÍCULO :



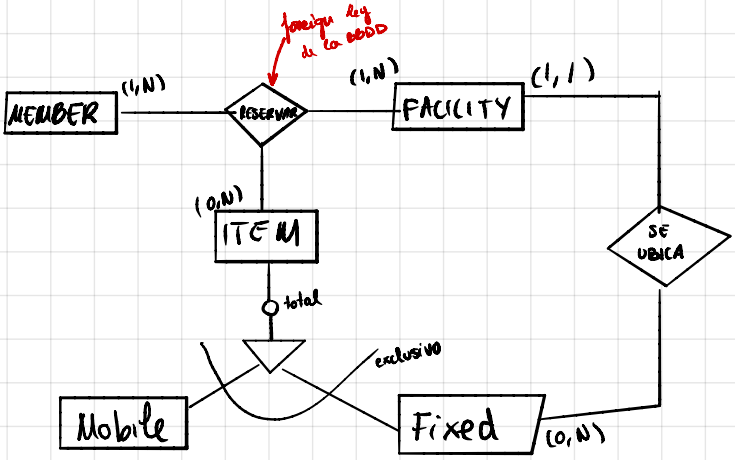
EJERCICIO 2

- FACILITY: type, city
- MEMBER : address, city, province, telephone , name, month fee
- ITEM : type
  - MOBILE : quantity
  - FIXED : (relación a facility)

especialización

◇ RESERVATION : date, start time , end time , items

En el diagrama entidad relación no se ponen las foreign key.



EJERCICIO 3

Tues 26/02/19

Entidad

- partido (siglas, nombre, dirección)
- ~~votante~~
- municipio (nombre, población, ~~censo~~, votosTotales)
  - población: enriquecido (200000 personas)
  - ~~censo~~: multievaluado
  - lista de personas que pueden votar
- mesa (blanco, nulos, numMesa,
- censo
- colegio (numDistrito)

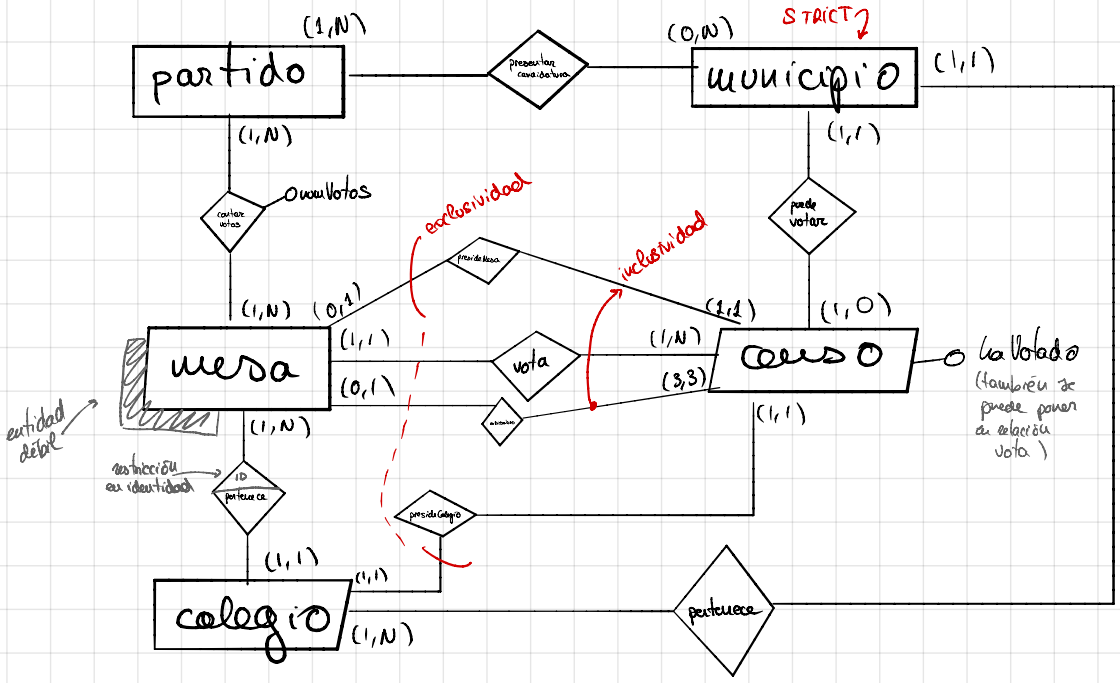
redundante

Tipo Relación

- candidatura (partido – municipio)
  (1,N) (1,N)
  (0,N) (0,N)
- votos (mesa – partido)
- municipio – censo
  (1,1) (1,N)
- mesa – censo
  (1,1) (1,N)
- colegio – mesa
  (1,1) (1,N)
- colegio – municipio
  (1,N) (1,1)
- presideColegio
  colegio – censo
  (1,1) (1,1)
- es miembro
  mesa – censo
  (0,1) (1,N)
- preside Mesa
  mesa – censo
  (0,1) (1,1)

Restricciones

- 1 candidatura partido * 1 Municipio
- T. relación
  presidirColegio ← esMiembro
- presidirColegio } exclusividad
  presidirMesa


partido
(1,N)
presentan candidaturas
(0,N)
STRICT
municipio
(1,1)
(1,N)
contar votos
0..muchos Votos
(1,1)
puede votar
(1,N)
exclusividad
inclusividad
presidente Mesa
(1,N)
(0,1)
(1,1)
(1,0)
mesa
(1,1)
(0,1)
vota
(1,N)
(3,3)
censo
la Votado
(también se puede poner en relación vota)
entidad débil
(1,N)
restricción en identidad
ID pertenece CE
presideColegio
(1,1)
(1,1)
colegio
(1,N)
pertenece

Tema 3.
Fundamentos del Modelo Relacional

3. Fundamentos del Modelo Relacional

1. Estructura de las Bases de Datos Relacionales
    - 1.1 Introducción
    - 1.2 Estructura del Modelo Relacional
2. Restricciones de Integridad
3. Teoría de las Dependencias
    - 3.1 Dependencias Funcionales. Claves
    - 3.2 Axiomas y Teoría de Cierres
4. Lenguajes Relacionales. Álgebra Relacional
    - 4.1 Introducción
    - 4.2 Álgebra Relacional

3. Fundamentos del Modelo Relacional

Referencias

- **Principales:**
  - Elmarsi, R.; Navathe,S.B.; ***Sistemas de Bases de Datos: Conceptos fundamentales*** (3ª edición). Addison-Wesley, 2002. Capítulos 7, 8 y 9
  - De Miguel, A.; Piattini, M;
    Fundamentos y Modelos de Bases de Datos (2ª edición).
    Ra-Ma, 1999. Capítulos. 6 y 7
- **Otras:**
  - Silberschatz, A.; Korth, H.F.; Sudarshan, S.; ***Fundamentos de Bases de Datos*** (4ª edición) McGraW-Hill, 2002. Capítulo 3
  - Date C.J.; ***Introducción a los Sistemas de Bases de Datos*** (7ª edición) Prentice Hall 2001. Capítulos 6 y 7.
  - Ullman, J.D.; Widom,J.; ***Introducción a los Sistemas de Bases de Datos***. Prentice Hall 1999. Capítulos 4 y 5

1. Estructura de las Bases de Datos Relacionales

1.1 Introducción

- En 1970, Codd publicó un artículo en ACM, proponiendo un nuevo modelo de datos que tenía como objetivo fundamental aislar al usuario de las estructuras físicas de los datos, consiguiendo así la **independencia** de las aplicaciones respecto de los datos.
- Este objetivo fundamental es expresado explícitamente por Codd:
  - *"... se propone un modelo relacional de datos como una base para proteger a los usuarios de sistemas de datos formateados de los cambios que potencialmente pueden alterar la representación de los datos, causados por el crecimiento del banco de datos y por los cambios en los caminos de acceso"*.
- El nuevo modelo se basa en la teoría matemática de las relaciones. Los datos se estructuran lógicamente en forma de relaciones (muy parecido al concepto de tabla).

1.1 Introducción

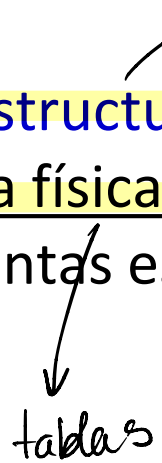
- Los avances más importantes que el modelo de datos relacional incorpora respecto a los modelos de datos anteriores son:
  - **Sencillez y uniformidad**: los usuarios ven la base de datos relacional como una colección de tablas. Al ser la tabla la estructura fundamental del modelo, éste goza de una gran uniformidad, lo que unido a unos lenguajes no navegacionales y muy orientados al usuario final, da como resultado la sencillez de los sistemas relacionales.
  - **Sólida fundamentación teórica**: al estar el modelo definido con rigor matemático, el diseño y la evaluación del mismo puede realizarse por métodos sistemáticos basados en abstracciones.
  - **Independencia de la interfaz de usuario**: los lenguajes relacionales, al manipular conjuntos de **registros**, proporcionan una gran independencia respecto a la forma en la que los datos están almacenados.

1.1 Introducción

- Las características del modelo relacional han hecho que prácticamente todos los SGBD comerciales implementen el modelo relacional.
  - Algunas de las principales empresas informáticas del mundo son, en origen, empresas de SGBD-R: ORACLE, postgres, Sybase, INFORMIX, …
- El tremendo éxito del modelo relacional ha supuesto que el cambio tecnológico a la siguiente generación esté siendo evolutivo y no revolucionario:
  - Triunfan los SGBD *Objeto-Relacionales*, y
  - Fracasan, en general, los SGBD de Objetos puros.

1.1 Introducción

- Un SGBD sólo necesita que el usuario pueda percibir la base de datos como un conjunto de tablas.
- Esta percepción sólo se aplica a la estructura lógica de la base de datos, no se aplica a la estructura física de la base de datos, que se puede implementar con distintas estructuras de almacenamiento.



1.2 Estructura del Modelo Relacional
Elementos básicos

- **Relación**: es la estructura básica del modelo relacional. Se representan utilizando tablas. } como se almacenan los datos
- **Atributo**: son las propiedades de la relación. Se representan mediante columnas en las tablas.
- **Dominio**: conjunto de valores sobre los que se define el tipo de un atributo. } conjunto de valores que puede tomar un atributo } - extensión -> enumeración - intensión -> basado en los tipos de datos definidos (cumplen un predicado)
- **Tupla**: ocurrencia de la relación. Se representa mediante filas dentro de las tablas.

↓ instancia de una relación
para cada atributo damos sus valores

1.2 Estructura del Modelo Relacional
Elementos básicos: Dominio

- El universo de discurso de una BD relacional está compuesto por un conjunto de dominios $\{D_i\}$ y un conjunto de relaciones $\{R_i\}$ definidas sobre esos dominios. -> La mayoría de los dominios son por intensión.
- Un **dominio** es un conjunto homogéneo de valores identificado por un nombre.
- Un dominio puede definirse de dos formas
  - Explícitamente (extensión):
    - *días de la semana = {lunes, martes, miércoles, jueves, viernes, sábado, domingo}*
  - usando tipos de datos (intesión):
    - *edad:entero*

1.2 Estructura del Modelo Relacional
Elementos básicos: Atributo

- Un atributo es la interpretación de un determinado dominio en una relación, es decir, la semántica de un dominio en una relación.
- Un atributo representa una propiedad de una relación.
- Un atributo tomará valores dentro de un domino.
- Distintos atributos de una relación, e incluso de distintas relaciones, pueden tomar valores dentro de un mismo dominio.

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Es el elemento fundamental del modelo relacional, y se representa usando tablas (aunque una tabla NO es una relación).

Las relaciones se modelan mediante tablas, pero una tabla no es una relación.

NOMBRE	atributo 1	atributo 2	.....	atributo n	
	XXX	XXX	.....	XXX	→ tupla 1
	XXX	XXX	.....	XXX	→ tupla 2
	.....	.....	.....	.....	.....
	XXX	XXX	.....	XXX	→ tupla m

Relación (nivel lógico):
- forma en la que guardas a nivel lógico una información

tipo de relación: conjunto de relaciones
↳ relación: conexión entre 2 tipos de entidades

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Matemáticamente, una **relación** definida sobre un conjunto de dominios $D_1 \ldots D_n$ (no necesariamente distintos) es un subconjunto del producto cartesiano de los *n* dominios, donde cada elemento de la relación (***tupla***) es una serie de *n* valores ordenados:
  - $R \subseteq D_1 \times D_2 \times \ldots \times D_n$, siendo *n* el **grado** de la relación.
- Esta definición **no** considera el concepto de atributo, por lo que dentro del contexto de las bases de datos son caracterizadas además por otros parámetros.

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Una **relación** (también llamada variable de relación) se caracteriza por:
  - Un **nombre** *R* que la identifica (algunos resultados intermedios no lo necesitan).
  - Una **cabecera de relación** que contiene *n* pares atributo-dominio ({A:D}) donde toma valores el atributo. Donde *n* es el ***grado de la relación***.
  - El **cuerpo de la relación** contiene *m* tuplas. Cada tupla estará compuesta de *n* pares atributo-valor. Donde *m* se denomina ***cardinalidad* de la relación**.

dominio de valores
nombre atributo
fila donde se guarda el atributo y su valor

↳ # tuplas de una relación
↳ # entidades que pueden participar en una relación } nivel (conceptual)

↳ # filas de la tabla (nivel lógico)

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

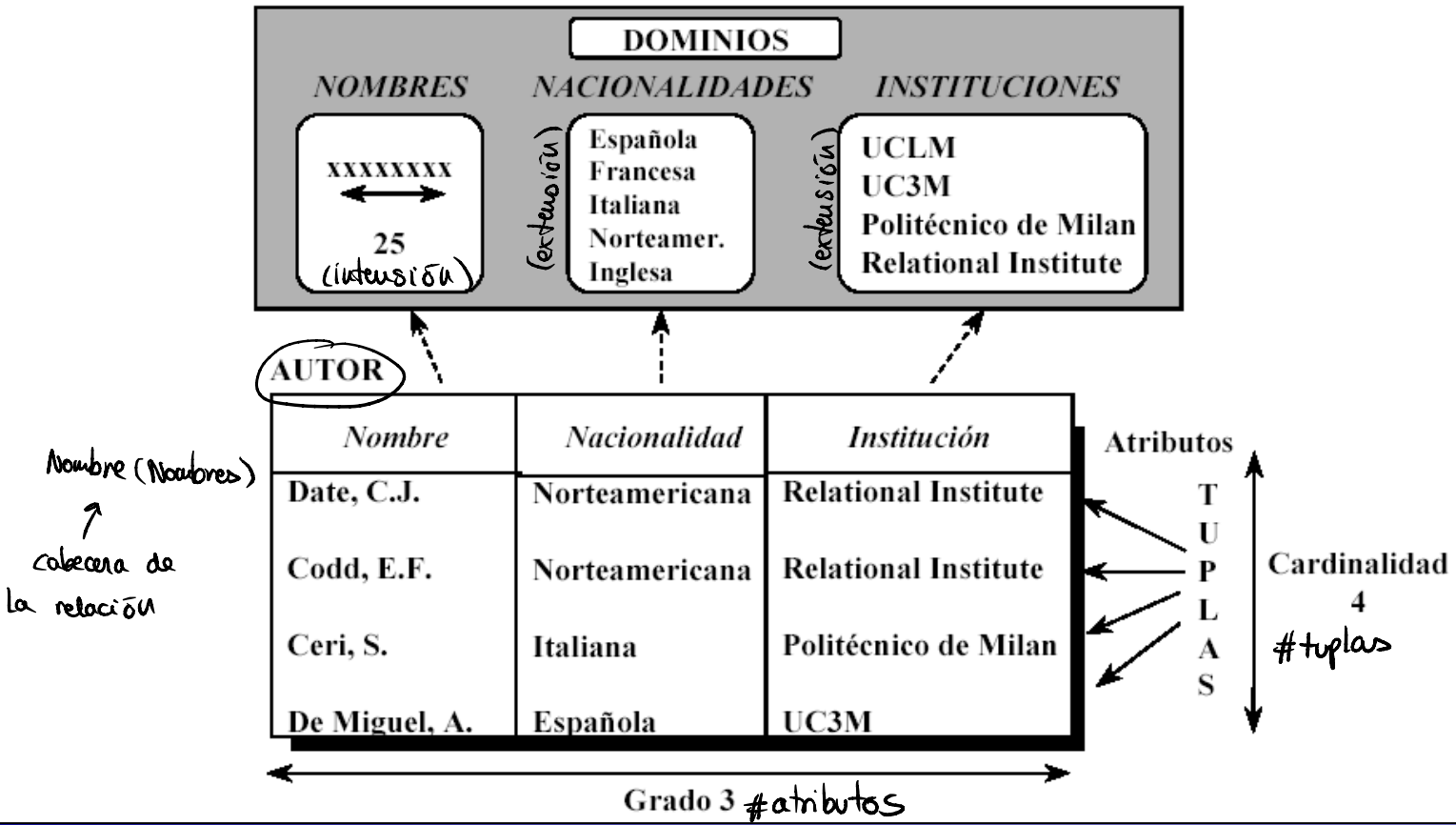
- Una relación (también llamada variable de relación) se caracteriza por:
  - El **esquema de la relación** (o intensión) está formado por el nombre *R* de la relación (si existe) y la cabecera de la relación. $R(\{A_i:D_i\}^n_{i=1})$. Es similar al concepto de *Tipo de Entidad* del modelo Entidad/Relación.
  - El **estado *r* del esquema de una relación** (o intesión) *R* (se suele denominar simplemente relación o valor de relación) se representa como *r(R)* y está constituido por el esquema y el cuerpo de la relación:

    r(R) = <esquema, cuerpo>

estado (nombre Relación) : r (R)

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Ejemplo



1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Esquema de la relación *Autor*

  AUTOR (*Nombre*: ***Nombres*****,** *Nacionalidad*: ***Nacionalidades*****,** *Institución*: ***Instituciones*****)**

- Relación *Autor*

AUTOR

Nombre	*Nacionalidad*	*Institución*
Date, C.J.	**Norteamericana**	**Relational Institute**
De Miguel, A.	**Española**	**UC3M**
Ceri, S.	**Italiana**	**Politécnico de Milan**

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Tipo de entidades vs Relaciones vs. Tablas

Tipo entidad (modelo conceptual)	Relación (modelo lógico)	Tabla (implementación)
Entidad	Tupla	Fila
Atributo	Atributo	Columna
-	Grado	Nº de columnas
Extensión	Cardinalidad	Nº de filas

Relación ≠ Tabla

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

examen

- Relaciones vs. Tablas

Relación ≠ Tabla

  - Una tabla no tiene las restricciones inherentes de una relación (como conjunto).
    - Una tabla puede tener dos filas iguales.
    - Las filas están ordenadas en el orden de grabación física, por defecto, o según el valor de la clave primaria.
    - Los atributos tienen un orden, según se ha definido en la tabla.
    - Una celda de una tabla puede contener más de un valor.

En el modelo relacional no tienen orden

solo 1 valor en el modelo relacional

Nombre	*Nacionalidad*	*Institucion*	*Idiomas*
Date, C.J.	Norteamericana	Relational Institute	*Inglés, Español*

- En el modelo relacional no podemos tener dos filas iguales.
- Es obligatorio que toda relación tenga un atributo que sea clave principal.

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Tipos de relaciones
  - No nominadas: no descritas en el esquema lógico relacional.
  - Nominadas: descritas en el esquema lógico relacional

    ↗ DROP TABLE

    - **Persistentes**: su definición (esquema) permanece en la base de datos, borrándose solamente mediante una acción explícita del usuario.
      - **Relaciones base**: existen por sí mismas, no en función de otras relaciones. Se crean especificando explícitamente su esquema de relación (nombre y conjunto de pares atributo/dominio).
      - **Vistas (*view*)**: son relaciones derivadas que se definen dando un nombre a una expresión de consulta. Lo único que se almacena es su definición en términos de otras relaciones con nombre. Se corresponden con el nivel externo de la arquitectura ANSI.

        ↳ Relación derivada: consulta SQL de mi BBDD que no guarda los datos solo guarda la consulta.

        ↳ Proporcionado al usuario en el nivel externo.

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- • Tipos de relaciones
  - No nominadas
  - Nominadas
    - **Persistentes**
      - Instantáneas (*snapshots*): son relaciones derivadas al igual que las vistas, pero tienen datos propios almacenados, que son el resultado de ejecutar la consulta especificada. Las instantáneas no se actualizan cuando cambian los datos de las relaciones sobre las que están definidas, pero se renuevan cada cierto tiempo, de acuerdo con lo indicado por el usuario en el momento de su creación. No pueden ser actualizadas por el usuario.
    - **Temporales:** a diferencia de las persistentes, una relación temporal desaparece de la BD en un cierto momento sin necesidad de una acción de borrado específica del usuario; por ejemplo, al terminar una sesión o una transacción.

una vista que guarda los datos mientras que no se le pide que la vuelva a cargar independientemente de que se actualicen los atributos de los que depende

2. Restricciones de Integridad

2.1 Restricciones de Integridad en el MR

- Las restricciones de integridad proporcionan un medio de asegurar que las modificaciones realizadas a la base de datos no provoquen la pérdida de la **consistencia** de los datos.
  - Consistencia de datos: es el estado coherente en la información o datos que contiene y que relaciona (**No hay contradicciones**), en el cual la información cumple las necesidades o expectativas de quien la requiera.
  - Una BD está en un estado consistente si:
    - Obedece todas las restricciones de integridad definidas sobre ella.
    - Eliminando o controlando las redundancias de datos. -> trigger
- Las restricciones del modelo relacional teórico han sido recogidas dentro del estándar SQL92, aunque con ciertas modificaciones.

2.1 Restricciones de Integridad en el MR
Tipos de restricciones

- **Restricciones inherentes**: Restricciones que impone el modelo de datos al no admitir ciertas estructuras. No son definidas por el usuario.
  - No hay dos tuplas iguales (obligatoriedad de la clave primaria)
  - El orden de las tuplas no es significativo
  - El orden de los atributos no es significativo
  - Cada atributo sólo puede tomar un único valor del dominio sobre el que está definido; no se admiten grupos repetitivos como valores de los atributos de una tupla. (Primera forma normal) -> NO atributos multivaluados
  - Regla de integridad de entidad: Ningún atributo que forme parte de la clave primaria de una relación puede tomar el valor nulo.

2.1 Restricciones de Integridad en el MR
Tipos de restricciones

- **Restricciones semánticas (**también llamadas de **usuario):** son facilidades que el modelo ofrece a los usuarios con el fin de estos puedan reflejar en el esquema, la semántica del mundo real.
  - Clave primaria (PRIMARY KEY)
  - Unicidad (UNIQUE)
  - Obligatoriedad (NOT NULL)
  - Integridad Referencial (FOREIGN KEY)
  - Verificación (CHECK)
  - Aserción (ASSERTION)
  - Disparador (TRIGGER)

2.1 Restricciones de Integridad en el MR
Tipos de restricciones: semánticas

- Clave primaria (PRIMARY KEY)
  - De la definición de relación se deduce que siempre existe, como mínimo, un conjunto de atributos que identifican de forma unívoca cada una de las tuplas de una relación, al que se denomina clave **candidata**.
  - La clave **primaria** es la clave candidata que el usuario escoge por motivos ajenos al modelo relacional.
    - Los valores del atributo/s que componen la clave primaria no pueden repetirse.
    - Los valores del atributo/s que componen la clave primaria no admiten valores nulos.

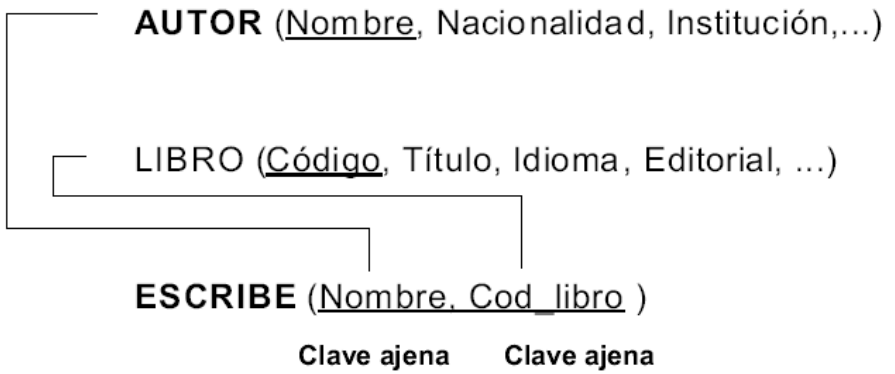
2.1 Restricciones de Integridad en el MR
Tipos de restricciones: semánticas

- Clave Candidata: Unicidad (UNIQUE)
  - Permite la definición de conjuntos de atributos cuyos valores no pueden repetirse dentro de la relación (claves alternativas).
  - Permite valores nulos si no se especifica lo contrario.

- Obligatoriedad (NOT NULL)
  - Indica que un conjunto de atributos no admite valores nulos.

2.1 Restricciones de Integridad en el MR
Tipos de restricciones: semánticas

- Integridad referencial (FOREIGN KEY)
  - Se llama clave foránea/ajena (foreign key) de una relación R2 a un conjunto de atributos cuyos valores deben coincidir con los valores de una clave candidata de una relación R1 (donde R1 y R2 no necesitan ser necesariamente distintas - unarias) o contener el valor nulo.
  - La clave candidata y la clave foránea implicadas deben estar definidas sobre el mismo dominio.



2.1 Restricciones de Integridad en el MR
Tipos de restricciones: semánticas

- Además de definir las claves foráneas, hay que determinar las consecuencias que pueden tener las operaciones de borrado y modificación realizadas sobre las tuplas de la relación referenciada:
  - **Operación restringida** (NO ACTION)

    El borrado de tuplas de la relación con la clave referenciada (o la modificación de la clave) sólo se permite si **NO** existen tuplas con este valor en la relación que contiene la clave foránea (Opción por omisión)
  - **Operación con transmisión en cascada** (ON DELETE/UPDATE CASCADE)

    El borrado de tuplas de la relación con la clave referenciada (o la modificación de la clave) lleva consigo el borrado (o modificación) en cascada de las tuplas de la relación que contiene la clave foránea.

2.1 Restricciones de Integridad en el MR
Tipos de restricciones: semánticas

- **Operación con puesta a nulos** (ON DELETE SET NULL)

  El borrado de tuplas de la relación con la clave referenciada (o la modificación de la clave) lleva consigo poner a nulos los valores de las claves foráneas de la relación que referencia.

- **Operación con puesta a valor por defecto** (SET DEFAULT)

  El borrado de tuplas de la relación con la clave referenciada (o la modificación de la clave) lleva consigo poner el valor por omisión a la clave foránea de la relación que referencia.

2.1 Restricciones de Integridad en el MR
Tipos de restricciones: semánticas

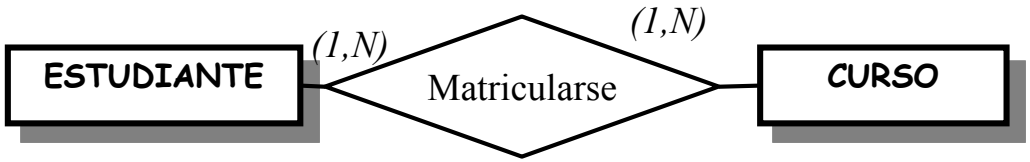
- Verificación (CHECK)
  - Permite especificar una condición que deben cumplir todas las tuplas de la relación. Esta condición se comprueba siempre que se actualiza o se añade una nueva tupla. CHECK (N_HORAS > 30).
- Aserción (ASSERTION)
  - La aserción funciona de forma similar a la verificación, pero en este caso la condición puede afectar a varios elementos, incluso a varias relaciones.

```
CREATE ASSERTION CONCEDE_SOLICITA AS
  CHECK (SELECT Cod_estud, Cod_Beca FROM CONCEDE) IN
  (SELECT Cod_estud, Cod_Beca FROM SOLICITA));
```

  - **PostgreSQL does not implement assertions at present**.

2.1 Restricciones de Integridad en el MR
Tipos de restricciones: semánticas

- Disparador (TRIGGER)
  - Permite que el usuario especifique cómo debe reaccionar el sistema cuando se satisface una condición.




```
CREATE TABLE estudiante (
    DNI         CHAR(8) PRIMARY KEY,
    nombre      VARCHAR2(20),
    apellidos VARCHAR2(20),
    localidad       VARCHAR2(30),
    telefono        CHAR(9)  );
```

2.1 Restricciones de Integridad en el MR
Tipos de restricciones: semánticas

- Disparador (TRIGGER)
  - Permite que el usuario especifique cómo debe reaccionar el sistema cuando se satisface una condición.

```
CREATE TABLE curso (
    CODIGO          CHAR(5) PRIMARY KEY,
    nombre          VARCHAR2(20),
    especialidad    VARCHAR2(20),
    localidad       VARCHAR2(30),
    duracion        NUMBER(2)  );

CREATE TABLE matricula (
    CODIGO          CHAR(5) REFERENCES CURSO(CODIGO),
    DNI             CHAR(8) REFERENCES ESTUDIANTE(DNI),
    PRIMARY KEY (CODIGO,DNI) );
```

2.1 Restricciones de Integridad en el MR
Tipos de restricciones: semánticas

RESTRICCION: Un estudiante no se puede matricular en cursos que no se impartan en su localidad.

```
CREATE OR REPLACE TRIGGER MISMALOC BEFORE
   INSERT ON MATRICULA
   FOR EACH ROW
   DECLARE
      LOC_EST ESTUDIANTE.LOCALIDAD%TYPE;
      LOC_CUR CURSO.LOCALIDAD%TYPE;
  BEGIN
      SELECT LOCALIDAD INTO LOC_EST
      FROM ESTUDIANTE
      WHERE DNI = :NEW.DNI;
      SELECT LOCALIDAD INTO LOC_CUR
      FROM CURSO
      WHERE CODIGO = :NEW.CODIGO;
      IF LOC_EST != LOC_CUR THEN
          RAISE_APPLICATION_ERROR(-20112, 'El curso no se imparte en la localidad del estudiante');
      END IF;
  END MISMALOC;
```

Esquema de relación y esquema relacional

- **Redefinición**: Un **esquema de relación completo** debe especificar los atributos y dominios sobre los que se define la relación, así como las restricciones de integridad que se deben cumplir para que la relación constituya una ocurrencia válida del esquema.

  $R(\{A_i:D_i\}^n_{i=1}, S)$

  Donde se ha añadido **S**, representando a las restricciones de integridad intraelementos. Lo más habitual es omitir S.

- El **esquema relacional de una BD** es la colección de esquemas de relación y de restricciones de integridad intraelementos.

  $E<(\{R_i\}, \{I_i\})>$

  Donde $\{R_i\}$ es el conjunto de esquemas e $\{I_i\}$ es el conjunto de restricciones.

3. Teoría de las Dependencias

3.1 Dependencias Funcionales. Claves.

- Las **dependencias funcionales** (DF) representan vinculaciones entre los distintos atributos que componen una relación. (semántica)
- Una dependencia funcional es una restricción inherente a la **semántica** de los atributos que se expresa en la forma: X→Y (X e Y son **descriptores**, esto es, conjuntos de atributos) y se lee "X implica Y"

3.1 Dependencias Funcionales. Claves.

- Las dependencias funcionales permiten determinar posibles claves de una relación.

- Las dependencias funcionales sirven como base para la teoría de la normalización de las bases de datos propuesta por Boyce y Codd. Esta teoría de la normalización permite eliminar ciertas anomalías en el diseño de las bases de datos relacionales (ver Tema 4: Diseño en el Modelo Relacional).

3.1 Dependencias Funcionales. Claves.

- Dada la relación *R* y *X* e *Y* subconjuntos de los atributos de *R*, decimos que ***Y* depende funcionalmente de *X***, si para dos tuplas cualesquiera de R *u* y *v*, si *u[X]=v[X]* entonces *u[Y]=v[Y]*.
- Se dice que F (un conjunto de DF) **implica lógicamente** X → Y, notado por F |= X→Y si cada ocurrencia r de R que satisface las dependencias funcionales en F también satisface X → Y.

  Ejemplo: {A → B, B → C} |= A → C.

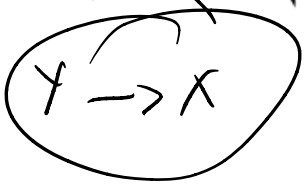
3.1 Dependencias Funcionales. Claves.

X			...	Y	
A	B	C	...	D	E
...	...	...	...	...	...
a1	b2	c3	...	d1	e2
...	...	...	...	...	...
a1	b2	c3	...	d1	e2
...	...	...	...	...	...

Vale X → Y

X			...	Y	
A	B	C	...	D	E
...	...	...	...	...	...
a1	b2	c3	...	d1	e2
...	...	...	...	...	...
a1	b2	c3	...	d2	e1
...	...	...	...	...	...

No vale X → Y



3.1 Dependencias Funcionales. Claves.

- Determinar las DF

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

D → B B ↛ C

A → C, se satisface:

- Las dos tuplas con valor a1 en A tienen el mismo valor en C, c1.
- Las dos tuplas con valor a2 en A tienen el mismo valor en C, c2.
- No existen otros pares de tuplas distintos que tengan el mismo valor en A.

C → A, no se satisface.

- Sean t1=(a2, b3, c2, d3) y t2=(a3 ,b3, c2, d4)
- tienen el mismo valor en C, c2 y distintos valores en A, a2 y a3, respectivamente.
- hemos encontrado un par de tuplas t1 y t2 tales que t1 [C] = t2 [C] pero t1 [A] ≠ t2 [A].

r satisface muchas otras DF.

- Por ejemplo:
  - AB → D

3.1 Dependencias Funcionales. Claves.

- Ejemplo 1:
  - *DNI→Nombre* (en la relación *Personal(DNI, nombre)*).
- Ejemplo 2:
  - *F = { CP→Ciudad, Ciudad Direccion →CP }* (en la relación *Código Postal(Direccion, Ciudad, CP)*).
- Una relación se podrá describir como: *(R, F)*, donde R es el conjunto de atributos de la relación y *F* el conjunto de dependencias funcionales entre dichos atributos.
  - Recordar el concepto de esquema de la relación

3.1 Dependencias Funcionales. Determinante y descriptores equivalentes

- Determinante:
  - Un **determinante o implicante** es un conjunto de atributos del que depende funcionalmente otro conjunto de atributos al que llamamos **determinado o implicado**.
  - Ejemplo:
    - El código de estudiante determina el nombre del estudiante:

      Cod_Estudiante→Nombre
- Descriptores equivalentes:
  - Dos descriptores X e Y se dice que son equivalentes si
    - X→Y ∧ Y→X
  - también se puede representar como:
    - X↔Y

3.1 Dependencias Funcionales. Claves.

- Una dependencia funcional *f* es **consecuencia lógica** de un conjunto de dependencias funcionales *F* (F ⊨ *f*) si se verifica en toda ocurrencia *r* de *(R,F)*.
- El **cierre de F (*F*⁺)** de un conjunto de dependencias funcionales *F* es el conjunto de dependencias funcionales que son consecuencia lógica de F ⇒ $F^+ = \{ f / F \vDash f \}$.
- *F* es una **familia completa** de dependencias funcionales si $F^+ = F$.
- Es necesario calcular F^+ para entender las implicaciones lógicas entre las dependencias, poder determinar las claves y decidir si una dependencia funcional *f* pertenece a F^+ o no. Para ello es necesario disponer de reglas de inferencia, como los axiomas de Armstrong.

3.1 Dependencias Funcionales. Claves.

X,Y,Z subconjuntos de atributos

- Axiomas de Armstrong
  - A1. Reflexividad

    $Y \subseteq X \Rightarrow X \rightarrow Y$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

se satisface:

A → C,

AB → D

Si Y=A y X= AB $\Rightarrow$ A $\subseteq$ AB $\Rightarrow$ Y $\subseteq$ X $\Rightarrow$ X $\rightarrow$ Y

X		y
A	B	A
a1	b1	a1
a1	b2	a1
a2	b2	a2
a2	b3	a2
a3	b3	a3

Cod_estud, nombre_est → Cod_estud
Porque: Cod_estud $\subset$ Cod_estud, nombre_est

3.1 Dependencias Funcionales. Claves.

X,Y,Z subconjuntos de atributos

- Axiomas de Armstrong
  - A2. Aumento

    $X \rightarrow Y$ y $Z \subseteq W \Rightarrow XW \rightarrow YZ$

    $X \rightarrow Y \Rightarrow XZ \rightarrow YZ$ (caso particular cuando Z=W)

se satisface: $A \rightarrow C$ y $AB \rightarrow D$

Si Y=C y X= A $\Rightarrow$ $X \rightarrow Y$

Si Z=B y W= AB $\Rightarrow$ $B \subseteq AB \Rightarrow Z \subseteq W \Rightarrow XW \rightarrow YZ$

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

X W → Y Z				
X	A	Z	Y	Z
A	A	B	C	B
a1	A1	b1	c1	b1
a1	A1	b2	c1	b2
a2	A2	b2	c2	b2
a2	A2	b2	c2	b2
a3	a3	b3	c2	b3

Como: Cod_estud → nombre_est
Cod_estud, cod_beca, duración → nombre_est, cod_beca
Porque: Cod_beca ⊂ Cod_beca, duración

3.1 Dependencias Funcionales. Claves.

X,Y,Z subconjuntos de atributos

- A3. Transitividad

  $X \rightarrow Y$ y $Y \rightarrow Z \Rightarrow X \rightarrow Z$

Si Cod_estud → cod_beca y cod_beca → duración
Entonces: Cod_estud → duración

- Una DF f se deriva de F ($F \models f$) si existe una secuencia $f_1, f_2, \ldots\ldots, f_n$ / $f_n = f$, donde cada f_i o bien pertenece a F o bien se deriva de las dependencias precedentes mediante la utilización de los axiomas de Armstrong.

3.1 Dependencias Funcionales. Claves.

X,Y,Z subconjuntos de atributos | Cod_estud → nombre_est | nombre_est → Cod_estud

- Reglas derivadas de los axiomas de Armstrong
  - *R1. Unión*
    - $\left.\begin{array}{l} X \to Y \\ X \to Z \end{array}\right\} \Rightarrow X \to YZ$

    Si Cod_estud → nombre_est y Cod_estud → cod_beca
    Entonces: Cod_estud → nombre_est, cod_beca
  - *R2. Pseudotransitividad*
    - $\left.\begin{array}{l} X \to Y \\ YW \to Z \end{array}\right\} \Rightarrow XW \to Z$

    Si Cod_estud → DNI y DNI, cod_beca → fecha_concesión
    Entonces: Cod_estud,cod_beca → fecha_concesión
  - *R3. Descomposición o proyectividad*
    - $\left.\begin{array}{l} X \to Y \\ Z \subseteq Y \end{array}\right\} \Rightarrow X \to Z$

    Si Cod_estud → nombre_est, cod_beca
    Entonces: Cod_estud → cod_beca
    Ya que: Cod_beca ⊂nombre_est, Cod_beca

3.1 Dependencias Funcionales. Claves.

- *Corolario:* $X \rightarrow A_1 A_2 \ldots A_n \Leftrightarrow X \rightarrow A_i, \ \forall i \in \{1, \ldots, n\}$
- Una dependencia funcional X → *Y* se dice **trivial** si $Y \subseteq X$
- *Y* **depende funcionalmente en forma elemental** de *X* si
  - $X \rightarrow Y \wedge \nexists X' \subset X \ / \ X' \rightarrow Y$
- Un subconjunto de atributos *X* de una relación $(\{A_1 A_2 \ldots A_n\}, F)$ es **clave** candidata de dicha relación si
  - $X \rightarrow A_1 A_2 \ldots A_n \in F^+ \wedge \nexists Y \subset X \ / \ Y \rightarrow A_1 A_2 \ldots A_n \in F^+$
  - **X implica a todos los atributos de R**
- Un subconjunto de atributos es **superclave** si es o contiene una clave

3.1 Dependencias Funcionales.

Importante

- El **cierre** de un subconjunto de atributos X (X^+) en una relación (R,F) donde R = $\{A_1 A_2 \ldots A_n\}$, es
  - $X^+ = \{ A \in A_1 A_2 \ldots A_n / F \vdash X \rightarrow A\}$

- *Lema:* $X \rightarrow Y$ se deriva de los axiomas de Armstrong a partir de F $\Leftrightarrow$ $Y \subseteq X^+$.

- El cálculo del cierre de un subconjunto de atributos permitirá determinar si una DF pertenece o no al cierre de ese subconjunto de atributos, lo que permite determinar si un subconjunto es clave o no.

3.1 Dependencias Funcionales. Comprobar si X → Y se deriva de DF *Importante*

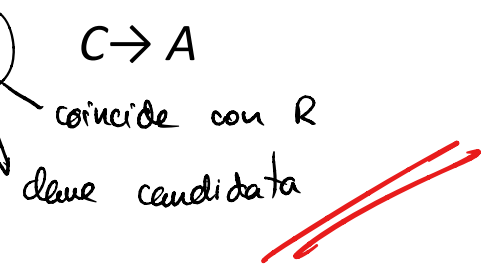
- Comprobar si X → Y se deriva de un conjunto de dependencias F equivale a comprobar si X → Y pertenece a F^+
- El algoritmo de comprobación es el siguiente:
  - Calcular el cierre X^+_F de X
  - Si $Y \subseteq X^+_F$ la dependencia X → Y ∈ F^+ (o lo que es igual F ⊢X → Y), en caso contrario X → Y ∉ F^+.

3.1 Dependencias Funcionales. Claves.

- Cálculo del cierre de un subconjunto de atributos *X*
  - Entrada
    - $U = \{ A_1 A_2 \ldots A_n \}$ (Atributos de la relación *R*)
    - *F* (Conjunto de dependencias funcionales en *U*)
    - $X \subseteq U$
  - Salida
    - X^+ (Cierre de *X* respecto a *F*)
  - Método (se calcula una secuencia de conjuntos $X^{(0)}$, $X^{(1)}$, …)
    - ***Paso 1.*** $X^{(0)} = X$
    - ***Paso 2.*** $X^{(i+1)} = X^{(i)} \cup \{ A \in U / (\exists Y \rightarrow Z \in F) \wedge (A \in Z) \wedge (Y \subseteq X^{(i)}) \}$
    - *Repetir el paso 2 hasta que* $X^{(i)} = X^{(i+1)} = X^+$.
      - R tiene un número finito de atributos $\Rightarrow$ el proceso termina.

3.1 Dependencias Funcionales. Claves.

- Cálculo del cierre de un subconjunto de atributos *X*
  - *Ejemplo*
    - *(R, F)*
      - *R = { A, B, C, D, E, G }*
      - *F = { AB→C, C→A, BC→D, ACD→B, D→EG, BE→C, CG→ BD, CE→AG }*
    - *X = BD = $X^{(0)}$ (queremos saber si BD podría ser clave candidata en R)*
    - *$X^{(1)}$ = BDEG* *D→EG*
    - *$X^{(2)}$ = BCDEG* *BE→C*
    - *$X^{(3)}$ = ABCDEG = X^{+}* *C→ A*



3.1 Dependencias Funcionales. Claves.

- Cálculo del cierre de un subconjunto de atributos *X*
  - *Ejemplo*
    - *(R, F)*
      - *R = { A, B, E, G,H,I,J }*
      - *F = {*AB→E, AG→J, BE→I, E→G,GI →H}
    - ejemplos de cierre de atributos respecto a F: **COMPROBARLO**
      - $A^+ = A$. √ no sería clave candidata
      - ✗ $AB^+ = ABEIGH$. AG → J
      - $AG^+ = AGJ$. √
      - $BE^+ = BEIGH$. ✓

$X^0 = AB$

$X^1 = ABE$ $\quad AB \rightarrow E$

$X^2 = ABEI$ $\quad BE \rightarrow I$

$X^3 = ABEIG$ $\quad E \rightarrow G$

$X^4 = ABEIGH$ $\quad IG \rightarrow H$

$X^5 = ABEIGHJ = X^+ = R$ $\quad AG \rightarrow J$

2.1) Relación de ejercicios Fundamentos MR

a) Cierta, no se permiten 2 tuplas iguales

b) Falsa, no tiene por qué $X \rightarrow Y$

c) Falsa, no tiene por qué ser cierto (normalmente falso)

d) Axioma aumentativo (Cierto)

e) Clave foránea -> relación desde la que se apunta

↳ Admite nulos, pero el atributo referenciado no puede admitir nulos

Falsa

f) Verdadera => Descomposición y también transitividad.

3.1 Dependencias Funcionales.
2 conjuntos de dependencias F y G son equivalentes

- Dos conjuntos de dependencias funcionales *F* y *G* son **equivalentes** si
  - $F^+ = G^+ \iff F \sim G$
- Es fácil saber si dos conjuntos de dependencias funcionales *F* y *G* son equivalentes comprobando que
  - $F \subseteq G^+ \wedge G \subseteq F^+$
- ***Lema***: todo conjunto de dependencias funcionales *F* es equivalente a un conjunto de dependencias funcionales *G* en el que no existen dependencias funcionales con más de un atributo en el lado derecho.
- ***Teorema***: todo conjunto de dependencias funcionales *F* tiene al menos una cobertura minimal (aunque no es única).

3.1 Dependencias Funcionales. Algoritmo de equivalencia

Importante

- Para evitar el coste computacional del cálculo de los cierres F^+ y G^+, se puede comprobar si cada dependencia de F se encuentra en G^+ y, viceversa, si cada dependencia de G se encuentra en F^+.
- Algoritmo:
  1. Si para toda dependencia X→ Y de G se cumple $Y \subseteq X^+_F$
     - significa que toda dependencia de G está en F^+ ($G \subseteq F^+$) y, por tanto, F es un recubrimiento de G.
  2. Recíprocamente, si para toda dependencia Z→ W de F, se cumple $W \subseteq X^+_G$
     - significa que toda dependencia de F está en G^+ ($F \subseteq G^+$) y, por tanto, G es un recubrimiento de F.
  3. Si se cumplen 1 y 2, F y G son mutuamente recubrimientos y, por tanto, son equivalentes.

3.1 Dependencias Funcionales. Algoritmo de equivalencia: Ejemplo

- Determinar si son equivalentes los siguientes conjuntos de dependencias:
  - R={A,B,C,D} F = {A → B, B → A, A → C, A → D}
  - G = {A → B, B → A, B → C, B → D}
- Solución:
  - Las dependencias A → B y B → A están en ambos conjuntos, por lo que las únicas dependencias de F que no están en G son A → C y A → D. Por tanto, debe calcularse el cierre de A con respecto al conjunto G:
    - A^+_G = ABCD
  - como C y D están contenidos en el cierre, queda demostrado que todas las dependencias de F están en G, luego G es un recubrimiento de F.
  - Análogamente, el cierre de B con respecto a F es:
    - B^+_F = ABCD
  - y por tanto, las dependencias B → C y B → D de G están contenidas en F^+, por lo que F es un recubrimiento de G.
  - Como conclusión, F y G son equivalentes.

3.1 Dependencias Funcionales. Minimizar el número de DF.

- Minimizar el número de DF permite
  - Reducir la complejidad algorítmica, al reducir el número de DF de partida.
  - Reducir el número de restricciones de integridad que debe controlar el SGBD.

3.1 Dependencias Funcionales. Claves.

- Un conjunto de dependencias funcionales *F* es **minimal (canónico, recubrimiento o cobertura irredundante)** si:

  a) Todos los atributos de la parte derecha de las dependencias funcionales son simples (tienen un solo atributo).

  Generar implicados únicos por el axioma de la unión.

  b) $(\nexists X \to A \in F) / ((F - \{ X \to A \}) \cup \{ Z \to A \}) \sim F$ y $Z \subset X$

  no se dan casos tales que existe $X \to A$ y $XY \to A \Rightarrow X \to A$, *se dice que los atributos del* ***conjunto Y son externos o extraños****.*

  Regla 1 de Reducción: $XY \to Z$ *y* $X \to Z \Rightarrow$ Y es extraño
  Regla 2 de Reducción: $XY \to Z$ *y* $X \to Y \Rightarrow$ Y es extraño

  c) $(\nexists X \to A \in F) / (F - \{ X \to A \}) \sim F$

  no existen dependencias ***redundantes****, es decir, no existen dependencias que pueden ser deducidas a partir de otras utilizando los axiomas de Armstrong.*

  (recordar: *Lema:* $X \to Y$ se deriva de los axiomas de Armstrong a partir de F $\Leftrightarrow$ $Y \subseteq X^+$. *De la transparencia 3.1 Dependencias Funcionales. Claves.)*

3.1 Dependencias Funcionales. Algoritmo Ullman y Atkins de minimal

Importante

- **Entrada**: DF (conjunto de dependencias elementales)
- **Salida**: H (recubrimiento minimal de DF)
- **Proceso** :

1. Obtener implicados únicos por el axioma de la unión.
2. Eliminación de atributos extraños.

   2.1) Repetir para cada dependencia X → B de DF:

   - L = X
   - Repetir para cada atributo A de X:
     - Si B ∈ $(L - A)^+$ entonces L = L - A
   - Reemplazar X → B por L → B
3. Eliminación de dependencias redundantes.
   - H = F
   - Repetir para cada dependencia X → A de DF:
     - G = H - { X → A }
     - Si A pertenece a X^+_G entonces H = G

Ejemplo 1

$R(A, B, C, D, E, G, H)$

$F = \{ GD \to H,\ G \to A,\ DA \to C,\ BE \to D,\ D \to B \}$

$L = X = GD \qquad GD \to H$

$\{D\}^+ = \{DB\}, \quad H \notin D^+$

$\{G\}^+ = \{A\}, \quad H \notin G^+$

Atributos externos o extraños (superfluos)

$L = X = DA \qquad DA \to C$

$\{D\}^+ = \{DB\} \qquad C \notin D^+$

$\{A\}^+ = \{A\} \qquad C \notin A^+$

$L = X = BE \qquad BE \to D$

$\{B\}^+ = \{B\} \qquad D \notin B^+$

$\{E\}^+ = \{E\} \qquad D \notin B^+$

· Dependencias funcionales redundantes

$GD \to H$

$J = H - \{GD \to H\}$

$X^+_J = \{GD\}^+ = \{G, D, A, B, C\} \qquad H \notin X^+_J$ no es redundante

$G \to A$

$J = H - \{G \to A\}$

$X^+_J = \{G\}^+ = \{G\} \qquad A \notin X^+_J$

$DA \to C$

$J = H - \{DA \to C\}$

$X^+_J = \{DA\}^+ = \{DA\} \qquad C \notin X^+_J$

$BE \rightarrow D$

$J = H - \{BE \rightarrow D\}$

$X^+_J = \{BE\}^+ = \{BE\}$ $\quad D \notin X^+_J$

$D \rightarrow B$

$J = H - \{D \rightarrow B\}$

$X^+_J = \{D\}^+ = \{D\}$ $\quad B \notin X^+_J$

↓ (como implicante sólo D)

Ejemplo 2

$R = \{DNIC, NombreC, TLf, CODP, Descripción, NF, pvp, NumLinea, ctd)$

$F = \{NF \rightarrow DNIC, \underline{NumLinea}\ \underline{NF} \rightarrow \underline{CODP}\ \underline{ctd}\ \underline{pvp}, DNIC \rightarrow \underline{NombreC}\ \underline{TLf},$

$CODP \rightarrow Descripción, NF \rightarrow NombreC\}$

con los nombres más cortos (quitamos implicados compuestos)

① $F' = \{NF \rightarrow DNI, NLNF \rightarrow C, NLNF \rightarrow ctd, NLNF \rightarrow pvp, DNI \rightarrow N,$

$DNI \rightarrow T, C \rightarrow D, NF \rightarrow N\}$

② con las implicantes compuestas (atributos externos / extraños)

$NLNF \rightarrow C$

$L = X = NLNF$

$\{NL\}^+ = \{NL\}$ $\quad C \notin \{NL\}^+$

$\{NF\}^+ = \{NF, DNI, N, T\}$ $\quad C \notin \{NL\}^+$

$NLNF \rightarrow ctd$

$L = X = NLNF$

$ctd \notin \{NL\}^+$

$ctd \notin \{NF\}^+$

$NLNF \rightarrow pvp$

$L = X = NLNF$

$pvp \notin \{NL\}^+$

$pvp \notin \{NF\}^+$

(3) redundancias

NF -> DNI

$J = H - \{NF \to DNI\}$

$X^+_J = \{NF\}^+ = \{NF, N\}$, DNI $\notin NF^+_J$

NLNF -> C

$J = H - \{NLNF \to C\}$

$X^+_J = \{NLNF\}^+ = \{NL, NF, ctd, pvp, DNI, N, T\}$, $C \notin \{NLNF\}^+$

NLNF -> ctd

$J = H - \{NLNF \to ctd\}$

$X^+_J = \{NLNF\}^+ = \{NL, NF, C, pvp, D, DNI, N, T\}$ ctd $\notin \{NLNF\}^+$

NLNF -> pvp

$J = H - \{NLNF \to pvp\}$

$X^+_J = \{NLNF\}^+ = \{NL, NF, C, ctd, D, DNI, N, T\}$, pvp $\notin \{NLNF\}^+$

DNI -> N

$J = H - \{DNI \to N\}$

$X^+_J = \{DNI\}^+ = \{DNI, T\}$, $N \notin \{DNI\}^+$

DNI -> T

$J = H - \{DNI \to T\}$

$X^+_J = \{DNI\}^+ = \{DNI, N\}$, $T \notin \{DNI\}^+$

$C \rightarrow D$

$J = H - \{C \rightarrow D\}$

$X^+_J = \{C\}^+ = \{C\}$, $D \in \{D\}^+$

$NF \rightarrow N$ => se puede quitar

$J = H - \{NF \rightarrow N\}$

$X^+_J = \{NF\}^+ = \{NF, DNI, N, T\}$, $N \in \{NF\}^+$

$F' = \{NF \rightarrow DNI, NLNF \rightarrow C, NLNF \rightarrow ctd, NLNF \rightarrow pvp, DNI \rightarrow N, DNI \rightarrow T, C \rightarrow D\}$ => solución

solo implicados = {ctd, pvp, N, T, D}

solo implicantes = {NF, NL}

ambos = {DNI, C}

$\{NFNL\}^+ = \{NF, NL, DNI, C, ctd, pvp, N, T, D\}$ ✓

3.1 Dependencias Funcionales. Claves.

- Practicar los algoritmos con los ejercicios de la “relación de ejercicios.pdf”
- Prueba a realizar estos ejercicios aplicando los algoritmos.
- Ejemplo 1 de conjunto de dependencias funcionales minimal
  - $F = \{ A \rightarrow B, B \rightarrow A, B \rightarrow C, A \rightarrow C, C \rightarrow A \}$
    - Solución 1. Eliminar $\{ B \rightarrow A, A \rightarrow C \}$

      $(B \rightarrow C \wedge C \rightarrow A) \Rightarrow B \rightarrow A$

      $(A \rightarrow B \wedge B \rightarrow C) \Rightarrow A \rightarrow C$
    - Solución 2. Eliminar $\{ B \rightarrow C \}$

      $(B \rightarrow A \wedge A \rightarrow C) \Rightarrow B \rightarrow C$

3.1 Dependencias Funcionales. Claves.

- Ejemplo 2 de conjunto de dependencias funcionales minimal
  - R = { A, B, C, D, E, G }
  - F = { AB →C, C →A, BC→D, ACD→B, D→E , D→G , BE→C
    , CG→B , CG→D , CE→A , CE→G }

    1. Implicados compuestos
    2. Atributos externos o extraños
    3. Redundancias

    - Atributos externos: { CE→A}
      - C → A ∧ CE → A ⇒ C→A (atributo externo)
    - Atributos externos: ACD → B puede sustituirse por CD →B
      - CD ⊂ ACD; Si X={CD} entonces X+ =¿? y ¿B ⊂ X+?.
    - redundantes: CG→B puede derivarse de F – {CG → B}
      - CG → D — hipótesis
      - CG →C — reflexiva
      - CG → DC — unión
      - CG →DC ∧ CD → B ⇒ CG→B — Transitiva

3.1 Dependencias Funcionales. Claves.

Cálculo de las **claves candidatas** de un (R,F)

Importante

1. Hacemos H=F conjunto de dependencias funcionales
2. Si existen atributos en R que no forman parte de ninguna dependencia funcional de F, estos pertenecen a todas las claves candidatas del esquema
3. Eliminamos de H los atributos de 2), pero al final del proceso se añaden a las claves obtenidas.
4. Todos los atributos no implicados (sólo pueden ser implicantes) en las dependencias funcionales de H pertenecen a todas las claves candidatas de (R,F).
5. Se comprueban si son claves (por ejemplo, calculando su cierre) todos los posibles conjuntos que contengan los atributos de 4) partiendo de menos a más cardinalidad. (si un conjunto es clave, ya no es necesario comprobar los conjuntos que lo contienen)
6. Se añaden los atributos de 2) a todas las claves obtenidas.

3.1 Dependencias Funcionales. Claves. Cálculo de las claves candidatas de un (R,F)

Importante

- Otra versión
  1. Si existen atributos en R que no forman parte de ninguna dependencia funcional de F, estos pertenecen a todas las claves candidatas del esquema
  2. Los descriptores equivalentes dan lugar a varias claves. (ver 3.1 Dependencias Funcionales. Determinante y descriptores...)
     - $(X \rightarrow Y \wedge Y \rightarrow X)$ o $(X \leftrightarrow Y)$
  3. Ningún atributo implicado que no es implicante forma parte de ninguna clave.
  4. Todo atributo implicante pero no implicado forma parte de todas las claves (siempre que no tenga otros equivalentes).
  5. Aquellos atributos que son implicantes e implicados pueden formar parte de alguna clave.

[TEMA 3]

4. Lenguajes Relacionales. Álgebra Relacional

4.1 Introducción a los lenguajes relacionales

- El Modelo relacional es un modelo lógico de datos que lleva asociado, además de una parte estática (estructura y restricciones) que especifica cómo se representarán los datos, una parte dinámica que permite la transformación entre estados de la base de datos. Esta transformación de un estado origen a un estado destino se realiza aplicando un conjunto de operadores.
- A través de estos operadores podemos llevar a cabo operaciones como:
  - Inserción, borrado y modificación de tuplas
  - Consultas

4.1 Introducción a los lenguajes relacionales

- Los lenguajes relacionales operan sobre conjuntos de tuplas y nos permitirán la manipulación de las relaciones, a través de ellos podremos expresar las consultas a la base de datos. El resultado de cada operación será una nueva relación que puede ser manipulada de nuevo.
- Tanto el estado origen como el estado final deben satisfacer las restricciones de integridad estáticas y la transformación debe cumplir las restricciones de integridad dinámicas.

4.1 Introducción a los lenguajes relacionales

- Podemos distinguir dos tipos de lenguajes relacionales
- **Lenguajes Algebraicos**:
  - Las consultas se expresan aplicando ciertos operadores sobre las relaciones y cuyo resultado es otra relación (Álgebra Relacional)
- **Lenguajes de cálculo de predicados**:
  - Las consultas describen el conjunto de tuplas que se desea obtener, mediante un predicado que indica la condición que deben satisfacer. Especificamos qué queremos obtener pero no cómo.
  - Se divide en dos subtipos:
    - Cálculo Relacional Orientado a tuplas
    - Cálculo Relacional Orientado a dominios.

4.2 Álgebra Relacional

Operadores

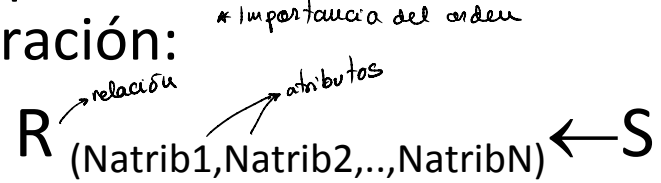
- Básicamente es un conjunto de operadores de alto nivel (expresión) que toman relaciones como sus operandos y retornan una relación como resultado.
- Codd definió lo que generalmente se conoce como álgebra "original" que constituían un conjunto de 8 operadores:
  - El Conjunto tradicional de operadores de conjuntos:
    - **Unión** **Intersección** **Diferencia** **Producto Cartesiano**
  - Operadores relacionales especiales:
    - **Selección** **Proyección** **Reunión** **División**

Operadores

- Los operadores del álgebra relacional también son clasificados en ocasiones de otras formas:
  - Por la completitud del lenguaje:
    - **Primitivos**: operadores esenciales que no pueden obtenerse a partir de otros
      - **Selección (σ)** **Proyección (Π)** **Producto** (x)
        Unión (U) **Diferencia** (-)
    - **Derivados**: se pueden obtener a partir de los operadores primitivos
      - **Intersección** (∩) **Reunión (⋈)** **División** (÷)
    - Operación auxiliar: el resultado de una expresión se asigna a una nueva relación
      - Asignación (←)
  - Por el número de operandos:
    - **Unarios**: actúan sobre una única relación.
    - **Binarios**: el operador tiene dos relaciones como operandos.

Operación auxiliar de asignación (←)

- Utilizaremos la **asignación** para indicar que el resultado de una expresión lo asignamos a una nueva relación. También será útil para dividir una operación compleja en una secuencia de operaciones más simples.
- ASIGNACIÓN Y RENOMBRADO (←): Podemos usarlo para asignar relaciones, pudiendo cambiar el nombre de sus atributos en la operación:



Siendo **R** y **S** relaciones

- Junto con la proyección también lo podemos utilizar para cambiar el orden de los atributos de una relación:

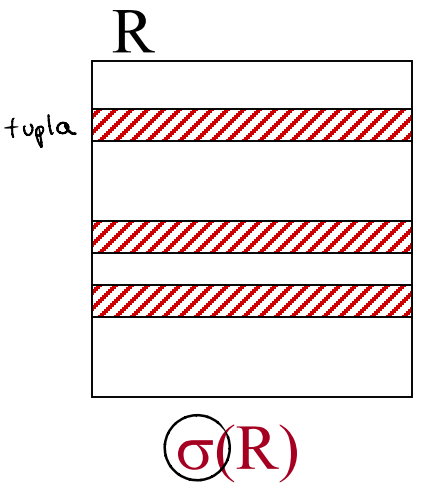
$$R \leftarrow \Pi_{(atrb_3,atrib_2,atrib_1)} S$$

Π: operador proyección que se ve más adelante

4.2.1 Operadores primitivos **unarios**
El operador Selección (σ) (Selección o restricción)

- Selecciona un conjunto de las tuplas de una relación que cumplen una condición lógica F. WHERE / HAVING

- $\sigma_p(R)$ → conjunto de tuplas de R que cumplen la fórmula P
  - Todas las tuplas de r'(R)={$t_i \in$ r(R) / p(t_i)= 'verdadero'}
- **La fórmula puede incluir**:
  - Nombres de atributos y, entre comillas, constantes.
  - Operadores de comparación θ: (=, ≠, >,<, ≥, ≤)
    - Condiciones Ai θ Aj ó Ai θ "a";
    - siendo Ai, Aj atributos y "**a**" una constante
  - Conectores lógicos (∧ (AND), ∨ (OR), ¬ (NOT))
  - El orden de precedencia de los operadores es: ¬, ∧, ∨



4.2.1 Operadores primitivos unarios

El operador Selección (σ) (Selección o restricción)

grado 3

Jugador

nombre	país	equipo
Deco	**Portugal**	**F.C. Barcelona**
Raúl	**España**	**R. Madrid**
Salgado	**España**	**R. Madrid**

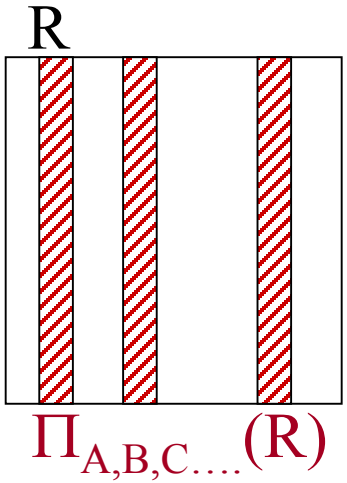
tupla

```
SELECT * FROM Jugador
WHERE país = 'España'
```

$\sigma_{país=\text{“España”}}(Jugador)$

nombre	país	equipo
Raúl	**España**	**R. Madrid**
Salgado	**España**	**R. Madrid**

4.2.1 Operadores primitivos unarios
El operador Proyección (Π)



- Operador de proyección (Π) SELECT

DISTINCT → filas no duplicadas

  - Selecciona ciertas columnas de una tabla:
    - $\Pi_{<lista\_atributos>}$ (R)
    - <Lista atributos>: Nombres de atributos separados por comas.

  - Permite Obtener y ordenar sólo los atributos que nos interesan de una relación así como organizarlos.
  - Las tuplas **NO** pueden repetirse.

4.2.1 Operadores primitivos unarios

El operador Proyección (Π)

Jugador

Nombre	País	Equipo
Deco	**Portugal**	**F.C. Barcelona**
Raúl	**España**	**R. Madrid**
Salgado	**España**	**R. Madrid**

SELECT equipo, país
FROM jugador
¿ DISTINCT ?

No se repiten tuplas

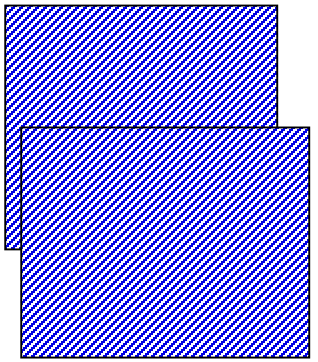
$\Pi_{\text{Equipo, País}}$ **(Jugador)**

Equipo	País
F.C. Barcelona	**Portugal**
R. Madrid	**España**

4.2.2 Operadores Primitivos de Conjuntos
Unión (U)

- **Compatibilidad de Unión**:
  - Dos relaciones son compatibles de unión si tienen el mismo grado (número de atributos) y cada par de atributos correspondientes por posición en ambas relaciones tienen el mismo dominio.

- Sean R, S relaciones, R y S compatibles de unión:
  - **UNIÓN** : R ∪ S, relación que incluye todas las tuplas que están en R(Z), en S(Y) o en ambas.
  - Donde Z e Y, el conjunto de atributos de R y S respectivamente, deben ser compatibles



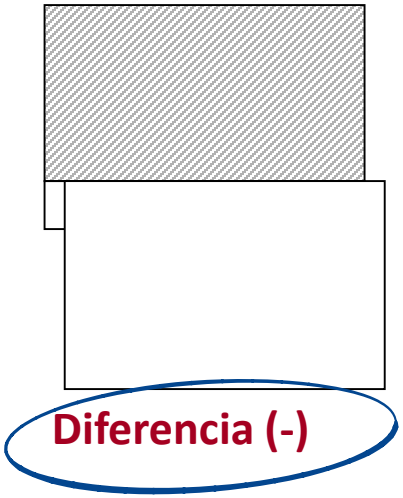
Unión (∪)

4.2.2 Operadores Primitivos de Conjuntos Diferencia (-) y Producto cartesiano (x)

– **DIFERENCIA: R–S**, relación que incluye las tuplas que están en **R(Z)** pero no en **S(Y).**

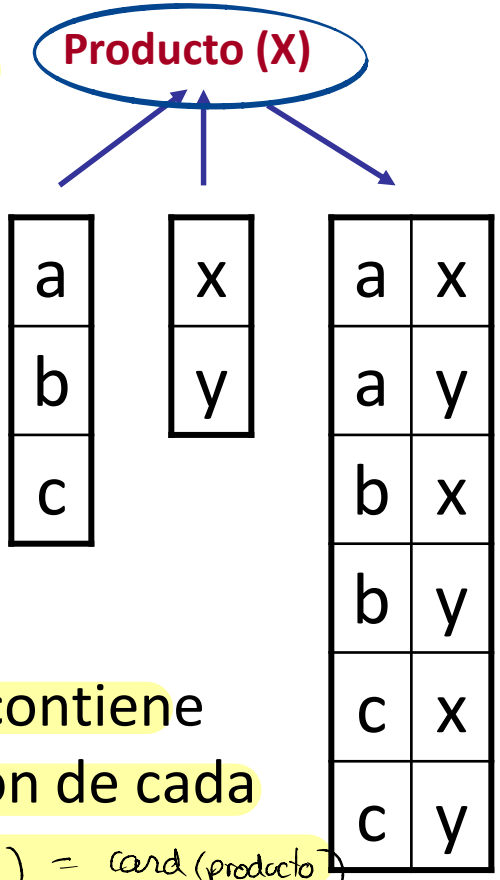
Hola

Donde Z e Y deben ser compatibles



EXCEPT -> SQL

- subconsulta en alguno de los filtros



– # filas = cardinalidad

– **PRODUCTO CARTESIANO: RxS**, relación que contiene todas las tuplas formadas por la concatenación de cada tupla de **R** con todas las de **S**

card(R) · card(S) = card(producto)

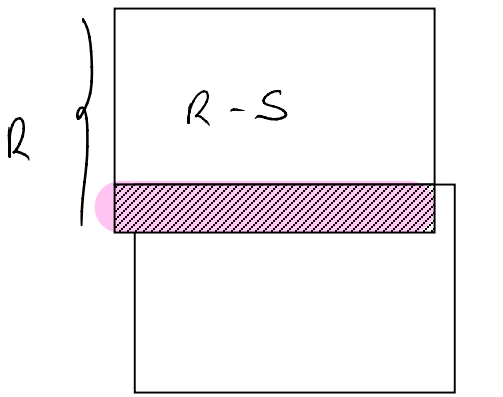
4.2.3 Operadores derivados
Intersección (∩)

- INTERSECCIÓN: R ∩ S, relación que incluye las tuplas que están en R(Z) y en S(Y). (a la vez)
  - Su significado es el mismo que en la teoría de conjuntos
  - Donde Z e Y deben ser compatibles
  - La intersección se puede definir en función de la diferencia:

$$R \cap S \equiv R - (R - S) \quad ¡!$$

$$R \cap S \equiv S - (S - R)$$

$$R \cap S \equiv (R \cup S) - ((R - S) \cup (S - R))$$



Intersección (∩)

4.2.3 Operadores derivados
División (÷)

–*DIVISIÓN:* R ÷ S → s'n equivalente en POSTGRES

Se aplica a dos relaciones R(Z) ÷ S(Y) donde Y ⊂ Z

(Z, Y conjunto de atributos)

Sea X= Z-Y (Conjunto de atributos de R que no son atributos de S).

La relación R puede considerarse como un conjunto de pares <x,y>.

- **El resultado de la división es una relación T(X) que contiene los valores x tales que <x,y> aparece en R para todos los valores y de S.**
- **$R \div S \equiv T1 \leftarrow \Pi_x (R) ; T2 \leftarrow \Pi_x ((T1 \times S) - R); DIV \leftarrow (T1 - T2)$**

qp

combinaciones que no están en R (incumplen la condición)

4.2.3 Operadores derivados
División (÷)

Dividendo R(Z)

Suministra

P#	C#
P1	C1
P1	C2
P1	C3
P1	C4
P1	C5
P1	C6
P2	C1
P2	C2
P3	C2
P4	C2
P4	C4
P4	C5

Divisores S(Y)

Com2

C#
C2
C4

subconjunto de atributos de R

Com3

C#
C1
C2
C3
C4
C5
C6

Resultados divisiones: T(X)

Suministra ÷ com2

P#
P1
P4

Suministra ÷ com3

P#
P1

- Al realizarse el producto entre el resultado y el divisor todas las tuplas resultantes se encuentran en el dividendo.
- El operador de división suele utilizarse en consultas que incluyen de alguna forma una condición que se debe cumplir "para todos", Ej. proveedores que suministran **todos** los componentes.

- **$R \div S \equiv T1 \leftarrow \Pi_x(R)\ ;\ T2 \leftarrow \Pi_x((T1 \times S) - R);\ DIV \leftarrow (T1 - T2)$**

q

combinaciones que no están en R (incumplen la condición)

$T_1 \leftarrow \Pi_A(R)$

P1
P2
P3
P4

$T2 \leftarrow \Pi_x((T_1 \times S) - R)$

~~P1C1~~	P3C1
~~P1C2~~	~~P3C2~~
~~P1C3~~	P3C3
~~P1C4~~	P3C4
~~P1C5~~	P3C5
~~P1C6~~	P3C6
~~P2C1~~	P4C1
~~P2C2~~	~~P4C2~~
P2C3	P4C3
P2C4	~~P4C4~~
P2C5	~~P4C5~~
P2C6	P4C6

cardinalidad 24

grado 2

$T_2 \Rightarrow$ P2 P3 P4

$T1 - T2 =$ (P1)

SQL ÷

```
SELECT DISTINCT X FROM R

  EXCEPT
SELECT X FROM (

SELECT DISTINCT * FROM

    (SELECT DISTINCT X FROM R), S

    EXCEPT

    SELECT * FROM R

)
```

4.2.3 Operadores derivados
Reunión (⋈) y combinación

- Combina dos relaciones según una cierta condición dada. El resultado es una relación que contiene las tuplas del producto cartesiano que cumplen la condición expresada.

  R ⋈<condición de reunión> S

- Si la condición de reunión es una condición simple de igualdad se denomina **EQUIRREUNIÓN**.
  - Suele ser la más utilizada para procesar vínculos entre relaciones. (clave externa – clave primaria)
- Relación $T(A1_1, \ldots A1_{n1}, A2_1, \ldots, A2_{n2})$ resultante:
  - $R \bowtie_{\text{condición}} S \equiv \sigma_{\text{condición}} (R \times S)$
  - Cardinalidad $\leq m1 \times m2$
  - $r(T)=\{<V1_{i1}, \ldots V1_{in1}, V2_{j1}, \ldots, V2_{jn2}> / <V1_{i1}, \ldots V1_{in1}> \in r(R) \wedge < V2_{j1}, \ldots, V2_{jn2}> \in r(S) \wedge V1_{ik} \; \Theta \; V2_{jl}\}$

4.2.3 Operadores derivados
Reunión (⋈)

Película

CodP	Título	Director	Duración
P1	EL SEÑOR DE LOS ANILLOS	Peter Jackson	178
P2	MEMENTO	Christopher Nolan	113
P3	LOS OTROS	Amenábar	114

Cartelera

Cine	Sala	CodP
Ábaco	1	P1
Ábaco	2	P2
Albacenter	1	P1
Albacenter	2	P3
Albacenter	3	P2

Cartelera ⋈ $_{\text{Cart.CodP = Peli.CodP}}$ Película

Cine	Sala	Cart.CodP	Peli.CodP	Título	Director	Duración
Ábaco	1	P1	P1	EL SEÑOR DE LOS ANILLOS	Peter Jackson	178
Ábaco	2	P2	P2	MEMENTO	Christopher Nolan	113
Albacenter	1	P1	P1	EL SEÑOR DE LOS ANILLOS	Peter Jackson	178
Albacenter	2	P3	P3	LOS OTROS	Amenábar	114
Albacenter	3	P2	P2	MEMENTO	Christopher Nolan	113

4.2.3 Operadores derivados
Reunión Natural (❋)

- **REUNIÓN NATURAL (R ❋ S) / (R ⋈S)**
  - Equirreunión en la que se eliminan los atributos superfluos (duplicados).
  - Se utiliza para relacionar tablas con atributos comunes.
  - Cuando los atributos comunes tienen el mismo nombre en ambas relaciones se omite la condición de combinación.

Cartelera ❋ Película

Cine	Sala	CodP	Título	Director	Duración
Ábaco	1	P1	EL SEÑOR DE LOS ANILLOS	Peter Jackson	178
Ábaco	2	P2	MEMENTO	Christopher Nolan	113
Albacenter	1	P1	EL SEÑOR DE LOS ANILLOS	Peter Jackson	178
Albacenter	2	P3	LOS OTROS	Amenábar	114
Albacenter	3	P2	MEMENTO	Christopher Nolan	113

4.2.3 Operadores derivados
Reunión Externa

– Reunión Externa

- Al combinar dos relaciones, nos permite que en el resultado aparezcan todas las tuplas que hay en una u otra relación, aunque no cumplan la condición de reunión.

- **Reunión Externa derecha R1 ⟖ R2, Conserva todas las tuplas de R2**
- **Reunión Externa Izquierda R1 ⟕ R2, Conserva todas la tuplas de R1**
- **Reunión Completa R1 ⟗ R2, Conserva la tuplas de ambas relaciones**

- Los atributos que no tengan valor definido para una determinada tupla se ponen con valor nulo.
- En el caso de la reunión natural lo expresaríamos como:

  Derecha: R1 ✱/ R2 Izquierda: R1 /✱ R2 Completa: R1 /✱/ R2

4.2.3 Operadores derivados
Reunión Externa

Película

CodP	Título	Director	Duración
P1	EL SEÑOR DE LOS ANILLOS	Peter Jackson	178
P2	MEMENTO	Christopher Nolan	113
P3	LOS OTROS	Amenábar	114
P4	TESIS	Amenábar	122
P5	EPISODIO IV	George Lucas	111

Cartelera

Cine	Sala	CodP
Ábaco	1	P1
Ábaco	2	P2
Albacenter	1	P1
Albacenter	2	P3
Albacenter	3	P2

reunión natural por la izq.

$(\Pi_{(CodP,\ Título)}$ Película) /✱ Cartelera ≡ Películas y donde se proyectan

CodP	Título	Cine	Sala
P1	EL SEÑOR DE LOS ANILLOS	Ábaco	1
P1	EL SEÑOR DE LOS ANILLOS	Albacenter	1
P2	MEMENTO	Ábaco	2
P2	MEMENTO	Albacenter	3
P3	LOS OTROS	Albacenter	2
P4	TESIS		
P5	EPISODIO IV		

4.2.4 Operaciones agregadas

- Algunas peticiones que se hacen comúnmente a las bases de datos no se pueden expresar con las operaciones estándar del álgebra relacional.
- La mayoría de los leguajes comerciales de consulta para SGBD relacionales sí cuentan con mecanismos para atender dichas peticiones.
- En los siguientes apartados veremos algunas operaciones que podríamos agregar al álgebra para ampliar su poder expresivo.

4.2.4 Operaciones agregadas

Funciones Agregadas y de Agrupación

– Funciones Agregadas y de Agrupación

La agrupación sobre valores comunes de determinados atributos para luego aplicar una función matemática suele ser una operación de consulta usual.

- Podemos expresarlo así:

  <atributos de agrupación> $f_{<\text{lista de funciones}>}$ (< Nombre de la relación>)

  separadas con comas

- Podemos tener funciones como: SUMA, MEDIA, VARIANZA, FRECUENCIA, MÍNIMO, MÁXIMO, etc.)
- La relación resultante tendrá los atributos de agrupación más un atributo por cada elemento de la lista de funciones.
- Si no se especifican atributos de agrupación, las funciones se aplicarán a los valores de los atributos de todas las tuplas de la relación

Ej: En la tabla SUMINISTROS(CODP, Componente, Maquina, CTDAD) Cantidad total de componentes suministrados por cada proveedor:

C (CODP, CTTOTAL) ← CODP f suma CTDAD (SUMINISTROS)

4. Lenguajes Relacionales. Álgebra Relacional Ejemplo

Ejemplo de consulta en Álgebra Relacional

Empresas (NombreE, Ramo, NºEmpleados)

Trabajo (NombreT, Edad, NombreE, Puesto_Trabajo)

Familia (Nombre1, Nombre2, Parentesco)

- Matrimonios que trabajan en puestos distintos de dos empresas hoteleras distintas

❶ $H \leftarrow \sigma_{Ramo=\text{“Hotel”}}(Empresas)$ ❷ $HOTELES \leftarrow \Pi_{NombreE}(H)$

❸ $TH \leftarrow \Pi_{NombreT,NombreE,Puesto\_Trabajo}(Trabajo * HOTELES)$

❹ $M \leftarrow \Pi_{Nombre1,Nombre2}(\sigma_{Parentesco=\text{“matrimonio”}}(Familia))$

❺ $MH \leftarrow TH \bowtie_{NombreT = Nombre1} M$

❻ $THM_{(NombreT1,NombreE1,Puesto\_trabajo1,Nombret2,.....)} \leftarrow TH \bowtie_{th.NombreT = Nombre2} MH$

❼ $SOLUCION \leftarrow \sigma_{NombreE1 \neq NombreE2 \,\vee\, Puesto\_trabajo1 \neq Puesto\_trabajo2}(THM)$

Operadores (Resumen)

- **Primitivos**:
  - Selección (σ) Proyección (Π) Producto (x)
  - Unión (U) Diferencia (-)
- **Derivados**:
  - Intersección ($\cap$) Reunión ($\bowtie$) División ($\div$)
    Reunión natural (*)

  reunión externa: Derecha: R1 */ R2 (⟖) Izquierda: R1 /* R2 (⟕)
  Completa: R1 /*/ R2 (⟗)
- **Operación auxiliar**: Asignación ($\leftarrow$)
- Operadores:
  - De comparación θ: ($=, \neq, >, <, \geq, \leq$)
  - Conectores lógicos ($\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT))
  - El orden de precedencia: $\neg, \wedge, \vee$

Operadores (Resumen)

- Funciones Agregadas y de Agrupación
  - Podemos expresarlo así:
    - <atributos de agrupación > $\int_{\text{<lista de funciones>}}$ (< Nombre de la relación>)
    - funciones:
      - SUMA, MEDIA, VARIANZA, FRECUENCIA, MÍNIMO, MÁXIMO, etc.
    - La relación resultante tendrá los atributos de agrupación más un atributo por cada elemento de la lista de funciones.
    - Si no se especifican atributos de agrupación, las funciones se aplicarán a los valores de los atributos de todas las tuplas de la relación.

4.3 Traducción del algebra relaciona a SQL

Operadores implementados en SQL (Resumen)

- **Selección (σ)**
  - Select * from …. Where predicado
- **Proyección (Π)**
  - Select distinct <atributos de la proyección> from ….; cláusula Select
- **Producto (x)**
  - Select * from tableA INNER/CROSS JOIN tableB ON ….
- **Unión (U)**
  - select * from TABLE_A

    UNION [ALL **| DISTINCT]** -- ALL duplica tuplas, DISTINCT sólo un ejemplar

    select * from TABLE_B

Operadores implementados en SQL (Resumen)

- **Diferencia (-)**
  - select * from TABLE_A
    where A_KEY **not in** (select A_KEY from TABLE_B)
  - select * from TABLE_A
    EXCEPT
    select * from TABLE_B
- **Intersección (⋂)**
  - select * from TABLE_A
    where TABLE_A.KEY **in** (select TABLE_B.KEY from TABLE_B)
  - select [distinct] * from TABLE_A
    INTERSECT
    select [distinct] * from TABLE_B

Operadores implementados en SQL (Resumen)

- **Reunión (⋈)**
  - Select distinct * from tableA **INNER JOIN** tableB **ON <condiciones>** ….
- **Reunión Natural (∗)**
  - Select distinct * from tableA **INNER JOIN** tableB **USING** ….
  - Select distinct * from tableA **NATURAL INNER JOIN** tableB ….

Operadores implementados en SQL (Resumen)

- **División (÷)**

  Se aplica a dos relaciones R(Z) ÷ S(Y) donde Y ⊂ Z ∧ X= Z-Y

  La relación R puede considerarse como un conjunto de pares <x,y>.

  - **$R \div S \equiv T1 \leftarrow \Pi_x (R) ; T2 \leftarrow \Pi_x ((T1 \times S) - R); DIV \leftarrow (T1 - T2)$**
  -
```
SELECT DISTINCT x FROM R
EXCEPT
SELECT x FROM (
    SELECT *
            FROM (SELECT DISTINCT x FROM R) NATURAL INNER JOIN S
    EXCEPT
    SELECT *  FROM R
)
```

Operadores implementados en SQL (Resumen)

- **División (÷)**

Para el ejemplo de la transparencia 4.2.3 Operadores derivados División (÷) .

- **$R \div S \equiv T1 \leftarrow \Pi_x(R)$; $T2 \leftarrow \Pi_x((T1 \times S) - R)$; $DIV \leftarrow (T1 - T2)$**
- SELECT DISTINCT P FROM R
  EXCEPT
  SELECT P FROM (
  SELECT * FROM (SELECT DISTINCT P FROM R) AS t1 , S
  EXCEPT
  SELECT * FROM R
) t2
  order by p

Data Output | Explain | M

p character (2)
P1
P4

Operadores implementados en SQL (Resumen)

- **División (÷)**
  - select a_.id
    from
      (select a.id, array_agg(a.b_id) as b_ids from a group by a.id) as a_
    join
      (select array_agg(b.id) as b_ids from b) as b_
    on b_.b_ids <@ a_.b_ids

Operadores implementados en SQL (Resumen)

- **División (÷)**
  -
```
select distinct A_KEY
from TABLE_C C
where not exists (
        select B_KEY
        from TABLE_B B
        where not exists (
                select *
                from TABLE_C CC
                where A.A_KEY = CC.A_KEY
                      and B.B_KEY = CC.B_KEY ))
```

4.4 Creación y gestión de Dominios en Postgres

1.2 Estructura del Modelo Relacional
Elementos básicos: dominio

- Ejemplo implementación de un dominio en postgres
  - Creación de un domino (CREATE DOMAIN)

```
CREATE DOMAIN name [ AS ] data_type
    [ COLLATE collation ]
    [ DEFAULT expression ]
    [ constraint [ ... ] ]

where constraint is:

[ CONSTRAINT constraint_name ]
{ NOT NULL | NULL | CHECK (expression) }
```

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Ejemplo implementación de un dominio en postgres

Dominio por Extensión

```
CREATE DOMAIN eye_color AS TEXT
CONSTRAINT valid_eye_colors CHECK (
   VALUE IN ( 'blue', 'green', 'brown' )
);

CREATE TABLE faces (
   face_id SERIAL PRIMARY KEY,
   name TEXT NOT NULL DEFAULT '',
   eye_color eye_color NOT NULL);
```

Dominio por intensión

```
CREATE DOMAIN us_postal_code AS TEXT
CHECK(
   VALUE ~ '^\d{5}$'
OR VALUE ~ '^\d{5}-\d{4}$'
);

CREATE TABLE us_snail_addy (
  address_id SERIAL PRIMARY KEY,
  street1 TEXT NOT NULL,
  street2 TEXT,
  street3 TEXT,
  city TEXT NOT NULL,
  postal us_postal_code NOT NULL);
```

POSIX-style regular expression

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Ejemplo implementación de un dominio en postgres
  - Dominio como un Tipo ENUMERATIVO.

```
CREATE TYPE eye_color AS ENUM ( 'blue', 'green', 'brown' );

CREATE TABLE faces (
   face_id SERIAL PRIMARY KEY,
   name TEXT NOT NULL DEFAULT '',
   eye_color eye_color NOT NULL
);
```

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Ejemplo implementación de un dominio en postgres
  - Dominio como valores de una tabla

```
CREATE TABLE eye_colors (
   eye_color TEXT PRIMARY KEY
);

INSERT INTO  eye_colors VALUES( 'blue' ), ('green'), ('brown' );

CREATE TABLE faces (
   face_id SERIAL PRIMARY KEY,
   name TEXT NOT NULL DEFAULT '',
   eye_color TEXT REFERENCES eye_colors(eye_color)
);
```

VARCHAR(20)

VARCHAR(20)

dominio sin join

1.2 Estructura del Modelo Relacional
Elementos básicos: Relación

- Ejemplo implementación de un dominio en postgres
  - Dominio como una restricción de valores de la columna

```
CREATE TABLE faces (
   face_id SERIAL PRIMARY KEY,
   name TEXT NOT NULL DEFAULT '',
   eye_color TEXT NOT NULL,
   CONSTRAINT valid_eye_colors CHECK (
      eye_color IN ( 'blue', 'green', 'brown' )
   )
);
```

columna

4. DISEÑO EN EL MODELO RELACIONAL

Contenidos

1. Introducción
2. Diseño Lógico. Del Modelo E/R al Modelo Relacional.
3. Normalización por medio de Dependencias Funcionales
    1. Descomposición de Esquemas
    2. Formas Normales de Codd
    3. Formas Normales Avanzadas

Referencias

- **Principales:**
  - Elmarsi, R.; Navathe,S.B.; ***Sistemas de Bases de Datos: Conceptos fundamentales*** (3ª edición). Addison-Wesley, 2002. Capítulos 14 y 15
  - De Miguel, A.; Piattini, M;

    Diseño de Bases de Datos Relacionales.

    Ra-Ma, 1999. Capítulos. 4, 5, 6, 7 y 10

    Tecnología y diseño de Bases de Datos

    Ra.Ma, 2006, Capítulo 16
- **Otras:**
  - Silberschatz, A.; Korth, H.F.; Sudarshan, S.; ***Fundamentos de Bases de Datos*** (4ª edición) McGraW-Hill, 2002.
  - Date C.J.; ***Introducción a los Sistemas de Bases de Datos*** (7ª edición) Prentice Hall 2001.
  - Ullman, J.D.; Widom,J.; ***Introducción a los Sistemas de Bases de Datos***. Prentice Hall 1999.

1. Introducción

El reto

- Podemos enfrentarnos al diseño de una base de datos relacional de dos formas:
  1. Aplicando inicialmente al Mundo Real un modelo semántico (p. e. E/R) para obtener un esquema conceptual, y transformar éste en un modelo relacional.
  2. Podemos plasmar directamente en el modelo relacional nuestra concepción del Mundo real.
- En general la primera forma produce un esquema relacional estructurado y con poca redundancia, pero pueden presentar algunos problemas derivados de fallos en:
  - La percepción del mundo real
  - El diseño del esquema E/R
  - El paso al modelo relacional

El reto

- Efectos de un mal diseño
  - Incapacidad para almacenar ciertos **hechos**.
  - Aparición de tuplas espúreas (Pérdida de información).
  - Pérdida de DF.
  - Valores Nulos (inaplicables)
  - Anomalías de inserción
    - **Redundancia** → Inconsistencia
    - Imposibilidad de insertar tuplas, existencia de valores nulos
  - Anomalías de modificación
    - Necesidad de propagar modificaciones por un diseño redundante.
  - Anomalías de borrado
    - Pérdida de datos
    - Necesidad de borrar varias tuplas para eliminar un sólo elemento.

El reto

- Estos posibles fallos de diseño hacen conveniente (necesario si optamos por diseñar directamente en el modelo relacional) aplicar un conjunto de **reglas** que nos permitan asegurar que un esquema relacional cumple ciertas propiedades:

Reglas → Formas Normales → Teoría de la normalización

1. Introducción

– **Ejemplo de diseño inadecuado:**

CÓDIGO PROV.	NOMBRE PROV.	CÓDIGO ART.	DESCRIPCIÓN	PRECIO
P2742	Juan López	A1267	Pack 100 CD	30
P2742	Juan López	A1298	Pack 10 DVD/DL	60
P1259	Martín García	A1267	Pack 100 CD	30
P1259	Martín García	A1298	Pack 10 DVD/DL	60
P1259	Martín García	A2356	Repr. MP3 USB	120
P1259	Martín García	A0012	500 Folios	3
P1259	Martín García	A0435	10 Papel Foto	6
P7789	Ramón Ramírez	A1267	Pack 100 CD	30
P7789	Ramón Ramírez	A0435	10 Papel Foto	6
•	•	•	•	•

Reglas -> Formas normales -> Teoria de normalización

El reto

- Problemas:
  - **Gran cantidad de redundancia**: Los datos del proveedor se repiten por cada producto que venda y los datos de los artículos se repiten por cada proveedor que los suministre.
  - **Anomalías de modificación**: Se puede cambiar el nombre de un proveedor en una tupla y por error no hacerlo en las otras (inconsistencia de datos)
  - **Anomalías de inserción**: No es posible incluir información sobre un proveedor del que no se tiene información sobre los artículos que suministra.
  - **Anomalías de Borrado**: Si se quiere eliminar un proveedor, también hay que eliminar el artículo que suministra si es el único que lo suministra.

2. Diseño Lógico. Del Modelo E/R al Modelo Relacional.

Reglas para el diseño Lógico

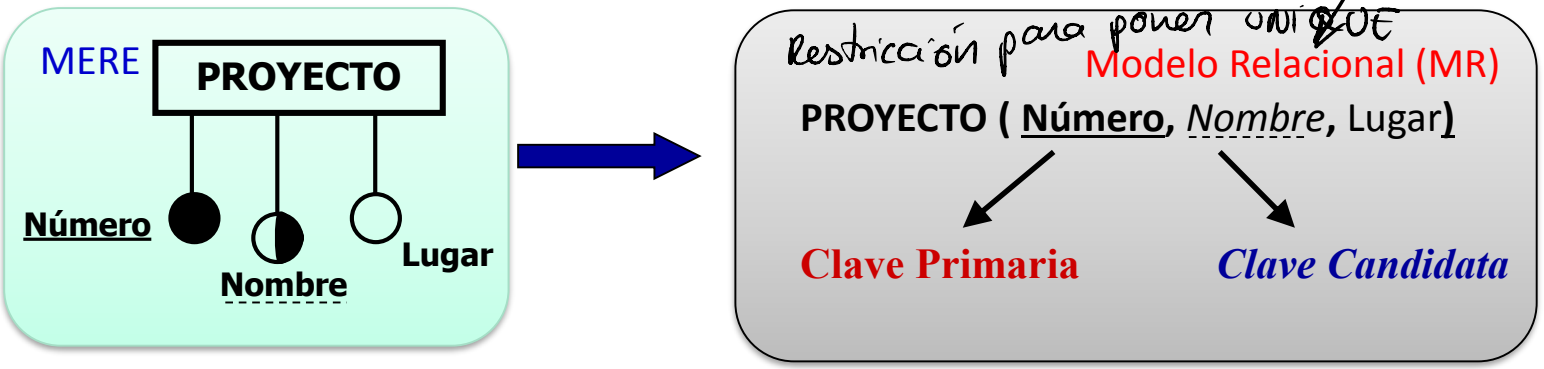
- R1: Transformación de dominios.
- R2: Transformación de entidades.
- R3: Transformación de atributos de entidades.
  - R3.1 Atributos identificadores principales.
  - R3.2 Atributos identificadores alternativos.
  - R3.3 Atributos no identificadores.
  - R3.4 Atributos compuestos.
  - R3.5 Atributos multivaluados
- R4: Transformación de Relaciones.
  - R4.1 Relaciones binarias 1:1.
  - R4.2 Relaciones binarias 1:N.
  - R4.3 Relaciones binarias N:M.
  - R4.4 Relaciones unarias 1:1.
  - R4.5 Relaciones unarias 1:N.
  - R4.6 Relaciones unarias N:M.
  - R4.7 Relaciones de grado > 2.
- R5: Transformación de Jerarquías
- R6: Transformación de Atributos de Relaciones.

Reglas para el diseño Lógico

- R1. Transformación de Dominios
  - Se crean los dominios y/o tipos necesarios para la creación de relaciones
- R2. Transformación de los tipos de entidades
  - Cada tipo de entidad se convierte en una relación. La relación se llamará igual que el tipo de entidad de donde proviene.
- R3. Transformación de Atributos:
  - Cada atributo de un tipo de entidad se transforma en un atributo de la relación a la que ha dado lugar.
  - Teniendo en cuenta que existen atributos identificador principal, otros que son identificadores alternativos y el resto de atributos que no son identificadores –atributos no identificadores-, establecemos las reglas:
    - R3.1 Atributos Identificadores:
      - Los atributos que son identificadores principales (AIP) pasan a formar la clave primaria de la relación (subrayados) → PRIMARY KEY

Reglas para el diseño Lógico

- R3. Transformación de Atributos:
  - R3.2 Atributos Identificadores Alternativos:
    - Los atributos que son identificadores alternativos pasan a formar parte de los atributos de la relación de los que hay que mantener la unicidad (UNIQUE). Subrayado discontinuo
  - R3.3 Atributos no identificadores:
    - Se representan como atributos de la relación correspondiente
  - R3.4 Atributos compuestos:
    - Se incluyen sólo los atributos simples que los componen

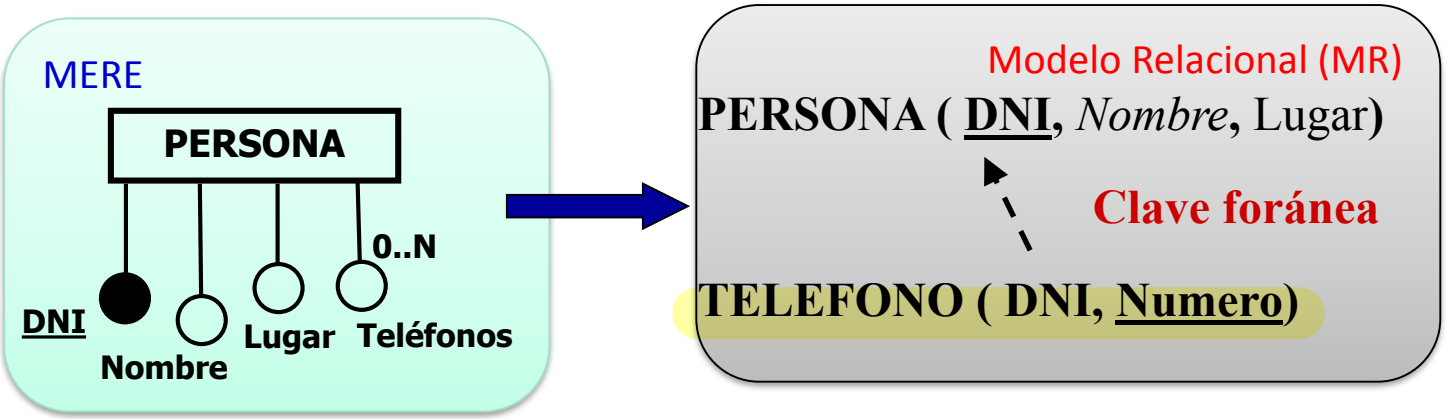


Reglas para el diseño Lógico

- R3. Transformación de Atributos:

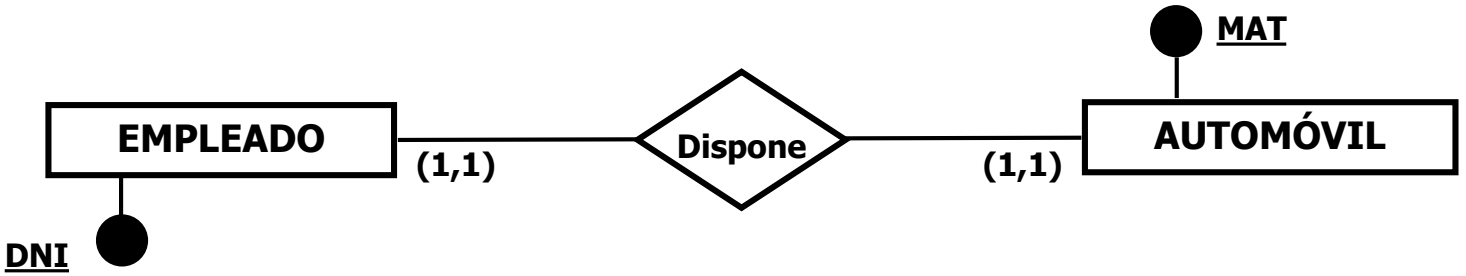
el modelo relacional no permite atributos multivaluadas

  - R3.5 Atributos Multivaluados:
    - Puesto que el Modelo Relacional **NO** permite atributos multivaluados, debe crearse una nueva relación cuyos atributos serán:
      - la clave primaria de la entidad original: como clave foránea y
      - el atributo multivaluado: como clave primaria.



Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.1. Tipos de relaciones Binarias 1:1;**
  - Si ambas clases de pertenencia son obligatorias (cardinalidad mínima >0), cada entidad se transforma en una relación y la clave de una de ellas aparece en la otra entidad como clave foránea sin admitir nulos.



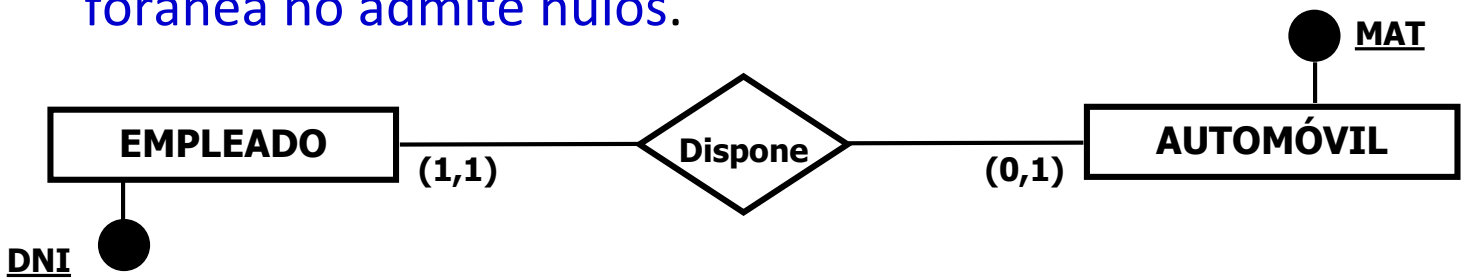
EMPLEADO (DNI, …… MAT) **AUTOMÓVIL (MAT ……..)**

unique

NO admite NULOS y UNIQUE **Clave foránea**

Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.1. Tipos de relaciones Binarias 1:1;**
  - Si una clase de pertenencia es opcional, la clave de la entidad con clase de pertenencia obligatoria, estará, como clave foránea, en la entidad con clase de pertenencia opcional. De esta forma evitamos los valores nulos en la clave foránea ya que la clave foránea no admite nulos.



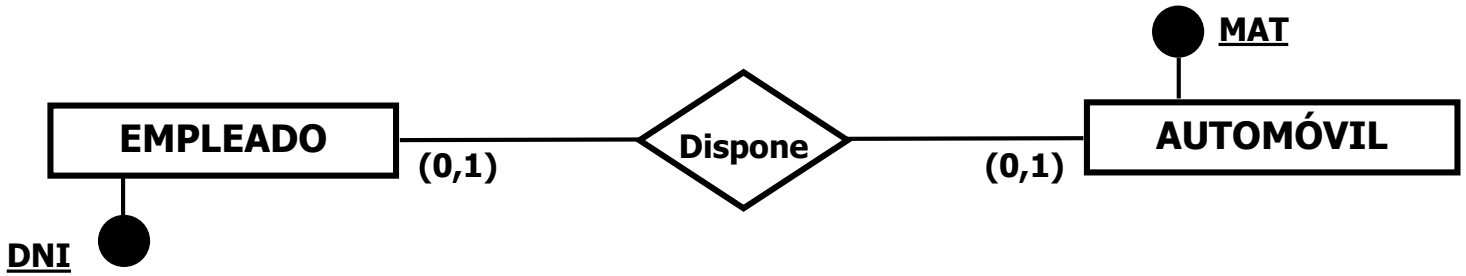
EMPLEADO (DNI, ……) **AUTOMÓVIL (MAT ……..... DNI)**

Clave foránea

NO admite NULOS Y UNIQUE

Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.1. Tipo de Relaciones Binarias 1:1**
  - Si ambas clases de pertenencia son opcionales (cardinalidad mínima >=0), una de las entidades contendrá la clave foránea asociada a la clave primaria de la otra, permitiendo valores nulos.



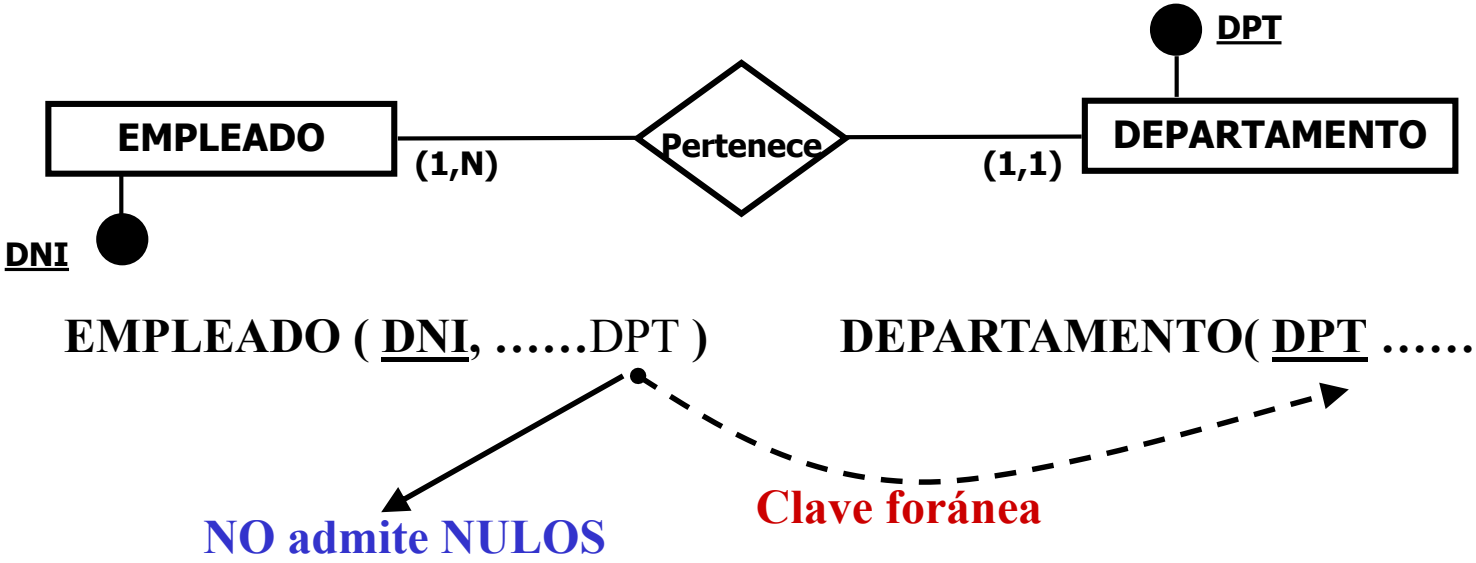
EMPLEADO (DNI, ……) **AUTOMÓVIL (MAT …….. DNI)**

Clave foránea

Admite NULOS y UNIQUE

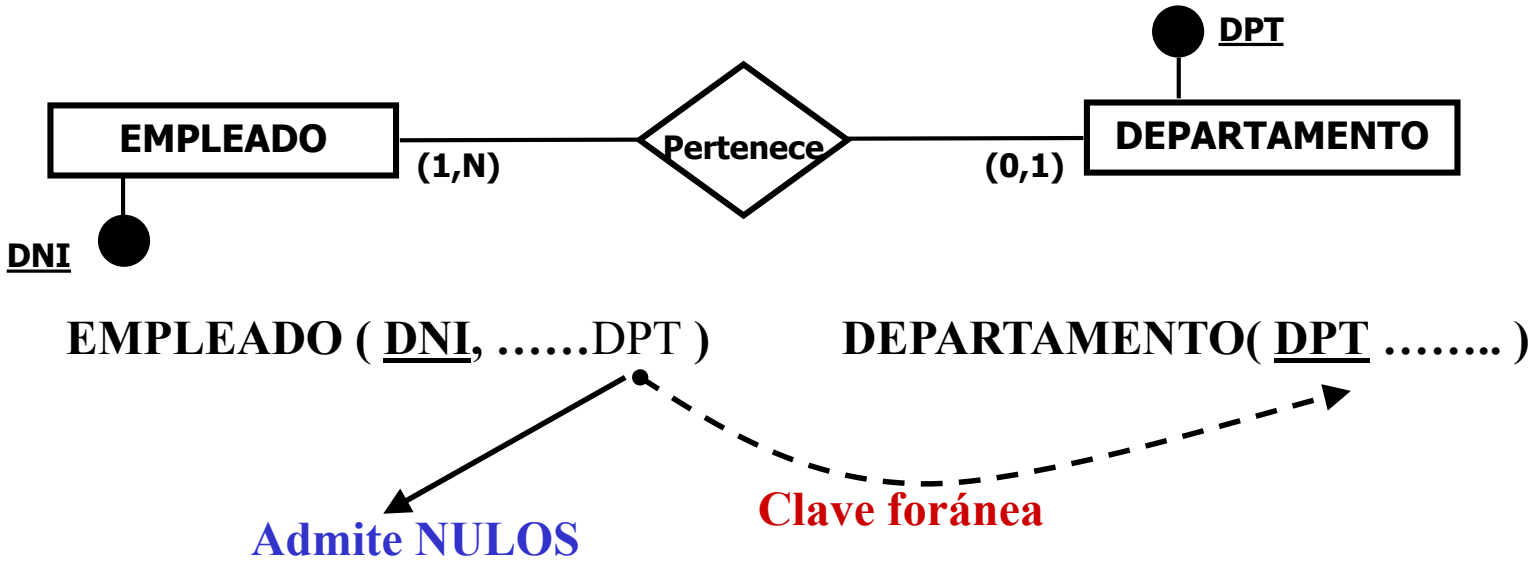
Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.2. Tipo de Relaciones Binarias 1:M**
  - La clave foránea estará en la relación correspondiente al tipo de entidad del lado de muchos. Si el otro tipo de entidad tiene clase de pertenencia obligatoria (cardinalidad mínima> 0) no se permiten valores nulos para la clave foránea.



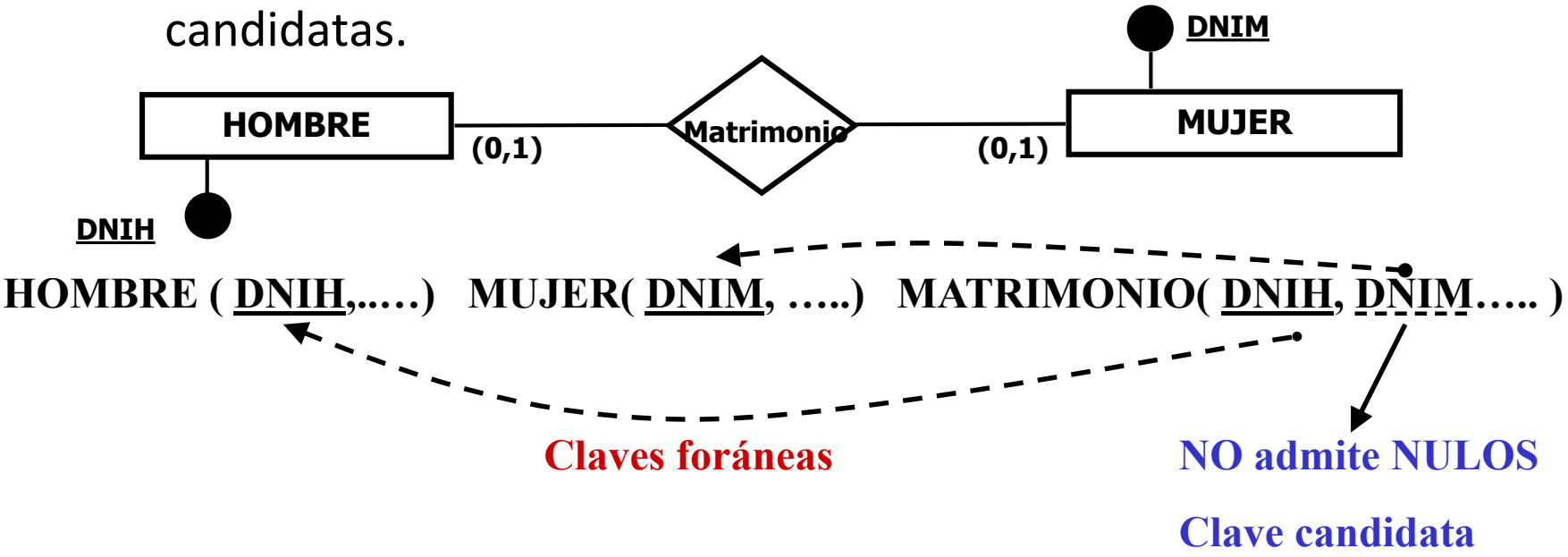
Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.2. tipo de Relaciones Binarias 1:M**
  - La clave foránea estará en la relación correspondiente a la entidad del lado de muchos. Si la otra entidad tiene clase de pertenencia opcional (cardinalidad = 0) se permiten valores nulos para la clave foránea.



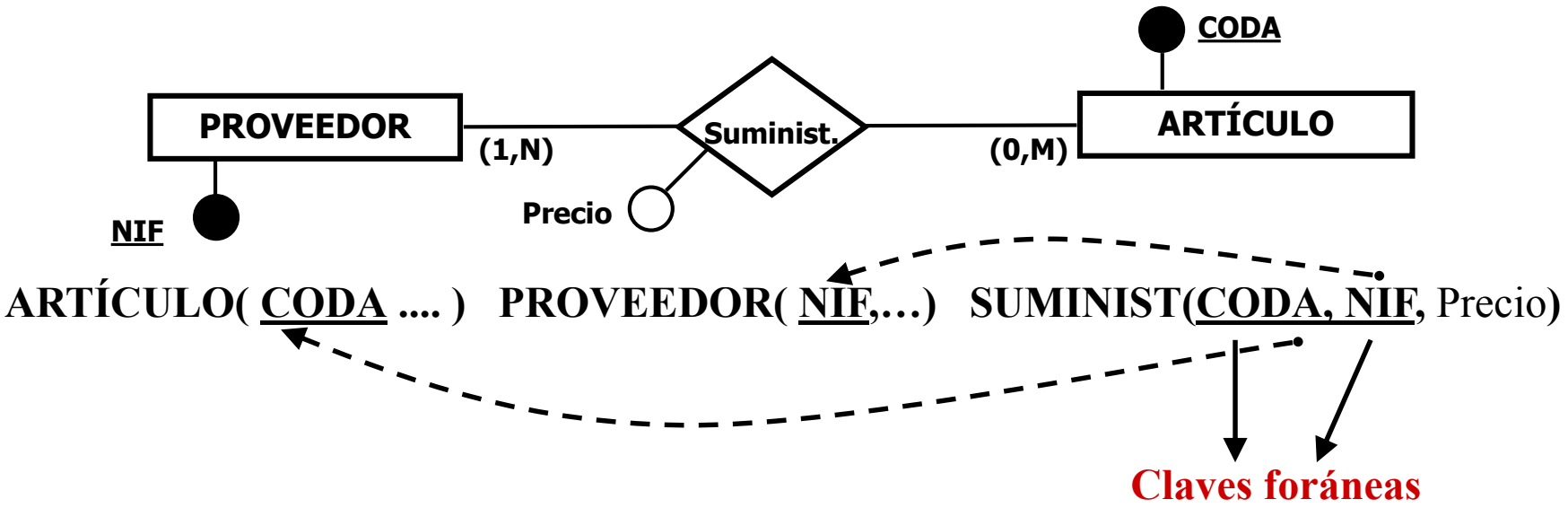
Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **Otra opción para tipo de Relaciones Binarias 1:1 y 1:M**
  - Se transforma el tipo de relación en una relación. En el caso de que la relación sea 1:M la clave primaria de la relación creada es el atributo correspondiente a la clave primaria de la relación con conectividad M. Si es 1:1 ambas claves foráneas son claves candidatas.



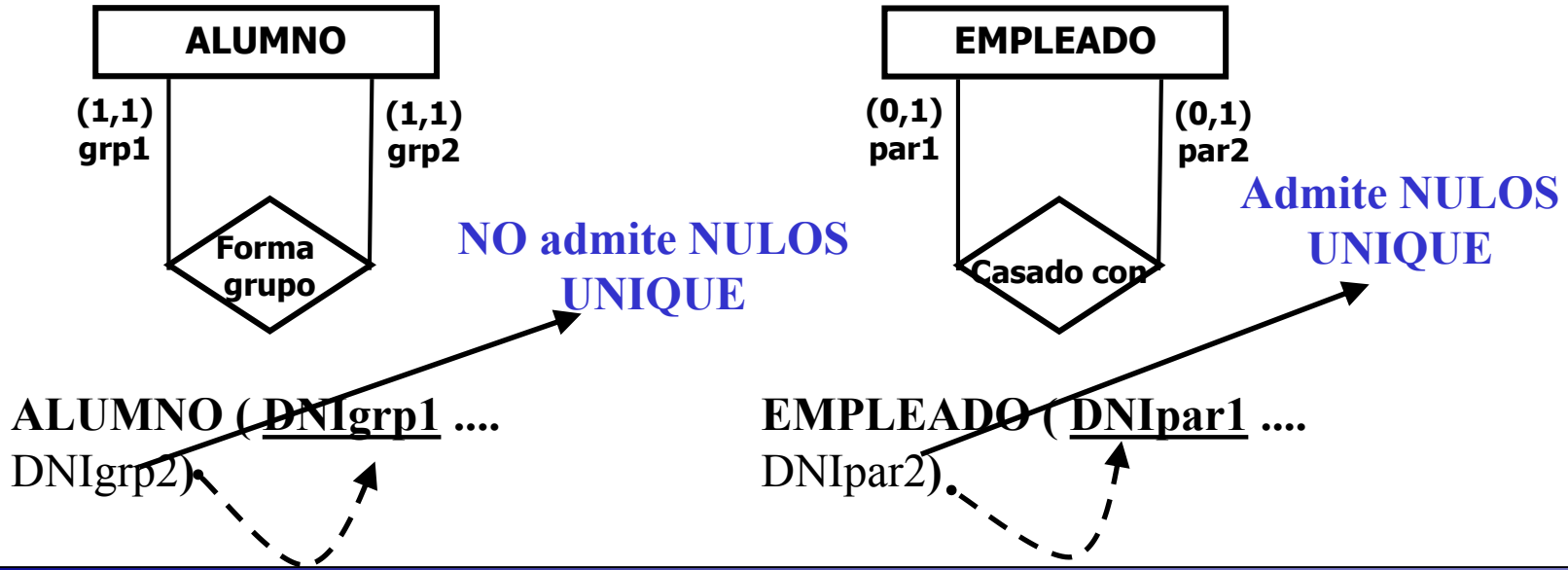
Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.3. tipo de Relaciones Binarias N:M**
  - Sin tomar en consideración la clase de pertenencia, se definen tres tablas, una para cada tipo de entidad y otra para el tipo de relación. Esta última contiene ambas claves foráneas (formando su clave principal) y los atributos que pudieran estar definidos en la relación.



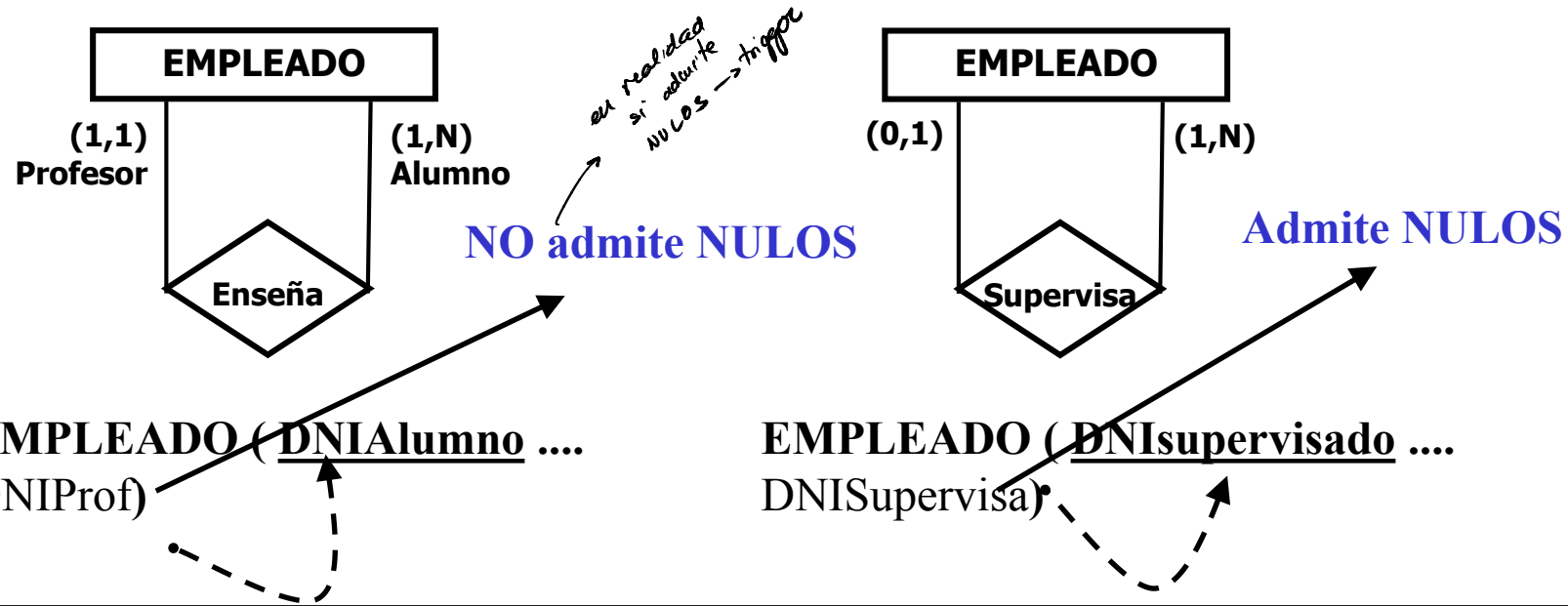
Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.4. tipo de Relaciones Unarias 1:1**
  - La clave del tipo de entidad aparece también como foránea en la relación procedente de la entidad. Si las clases de pertenencia son opcionales, la clave foránea admite nulos. Si son obligatorias, la clave foránea no admite nulos.



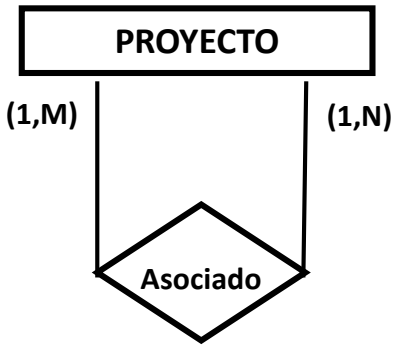
Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.5. tipo de Relaciones Unarias 1:N**
  - La clave de la entidad aparece también como foránea en la relación procedente del tipo de entidad. En el caso de clase de pertenencia opcional admite nulos y en caso contrario no.



Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.6. tipo de Relaciones Unarias N:M**
  - Además de la relación correspondiente al tipo de entidad se utiliza una relación para el tipo relación con dos claves foráneas referentes a la misma clave primaria de la entidad. La transformación es idéntica para los dos tipos de clase de pertenencia.



PROYECTO (NP .....)

ASOCIADO (NP1, NP2)

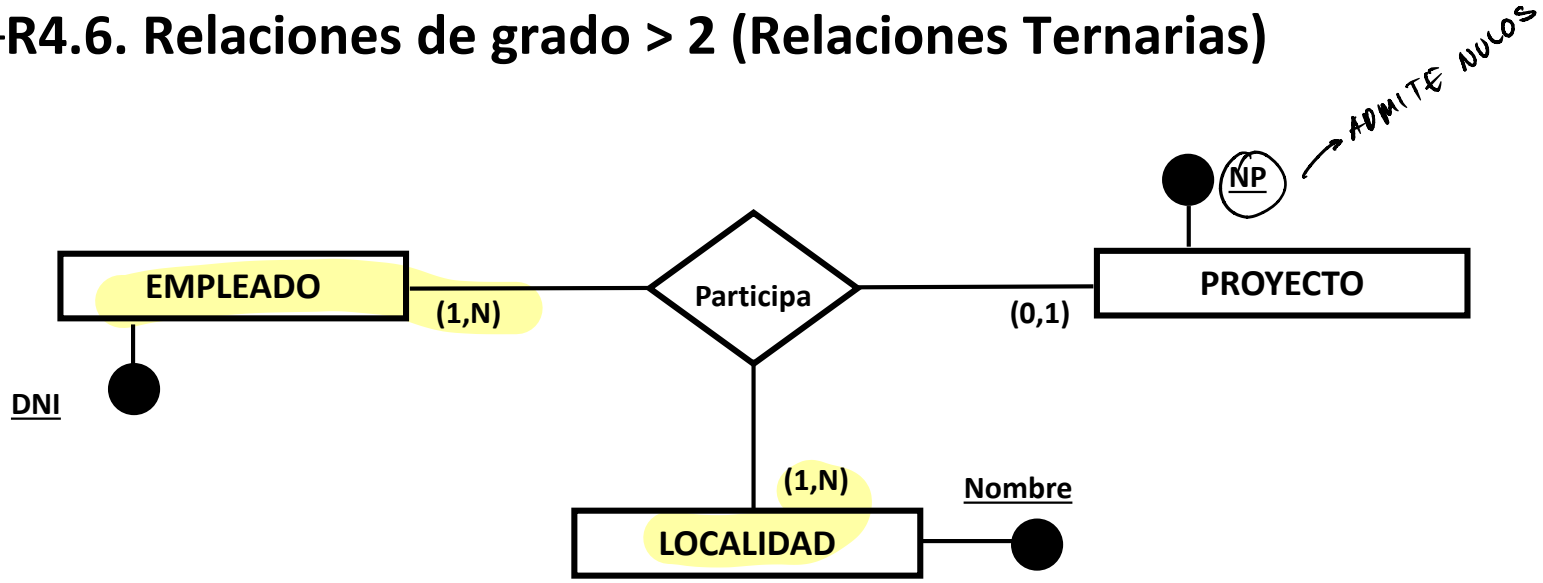
NO admite NULOS

Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

- **R4.6. Tipo de relaciones de grado > 2 (Relaciones Ternarias)**
  - Todas las variedades se transforman creando, además de las relaciones de los tipos de entidades, una relación del tipo de relación que contenga las claves primarias de todas las entidades relacionadas (formando una clave primaría) y los atributos de la relación.
  - En el caso de las ternarias, por cada lado de "uno", la pareja de claves foráneas de las entidades enfrentadas es una clave para la relación y por tanto existe una dependencia funcional entre esta clave y el resto de atributos (una DF para cada conectividad de "uno").
  - Es fácilmente generalizable para relaciones de grado mayor que 3.

Reglas para el diseño Lógico
R4. Transformación de tipos de Relaciones

–R4.6. Relaciones de grado > 2 (Relaciones Ternarias)



EMPLEADO(DNI ....) PROYECTO(NP,...) LOCALIDAD(Nombre, ...)

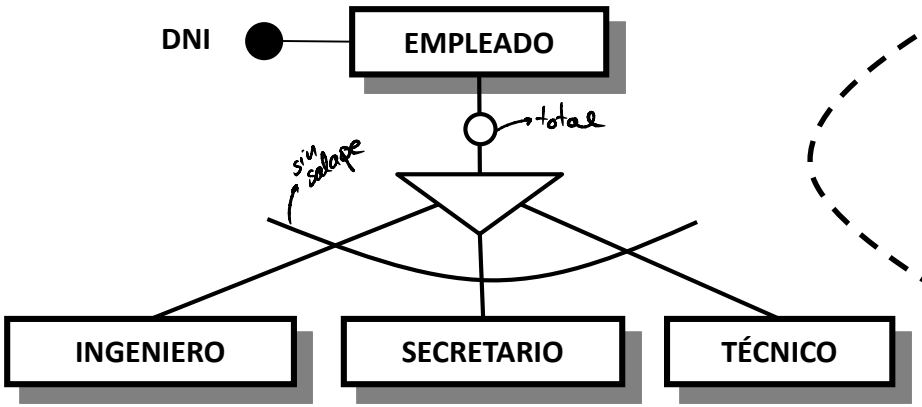
PARTICIPA (DNI, Nombre, NP) **F= { DNI, Nombre → NP }**

Reglas para el diseño Lógico
R5. Transformación de Jerarquías

- **Jerarquías Subconjunto y Generalización**
  - Tres posibles transformaciones de la jerarquía:
    1. Produce una relación para el tipo de entidad genérica, que contiene la clave de la entidad genérica y todos los atributos comunes, y una relación para cada una de las subentidades que contienen la clave de la entidad genérica como clave principal y clave foránea simultáneamente, y sólo los atributos específicos.
       - Esta es la solución adecuada cuando existen muchos atributos distintos entre los subtipos y se quieren mantener de todas maneras los atributos comunes a todos ellos en una relación.

Reglas para el diseño Lógico
R5. Transformación de Jerarquías

- **Jerarquías Subconjunto y Generalización**
  - Tres posibles transformaciones de la jerarquía:
    1. Produce una relación para el tipo de entidad genérica, …



EMPLEADO (DNI, At. comunes)

INGENIERO (DNI, At. específicos)

Modificación y borrado en cascada

SECRETARIO (DNI, At. específicos)

Mod. y borrado en cascada

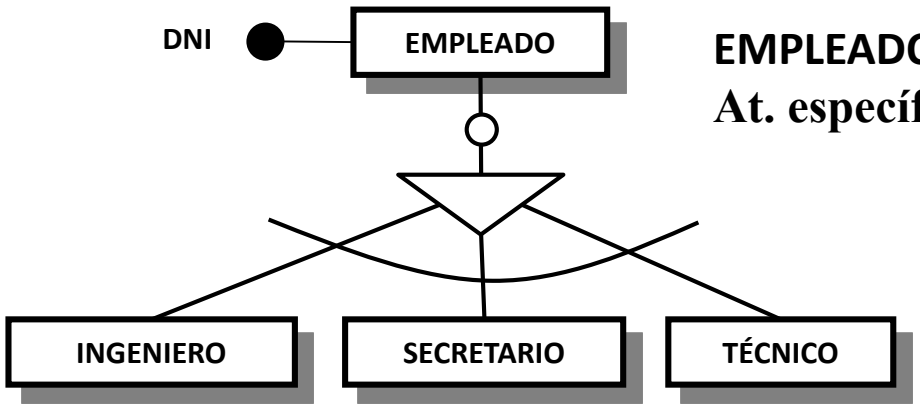
TÉCNICO (DNI, At. específicos)

Mod. y borrado en cascada

Ver enlace diapositivas MOODLE

Reglas para el diseño Lógico
R5. Transformación de Jerarquías

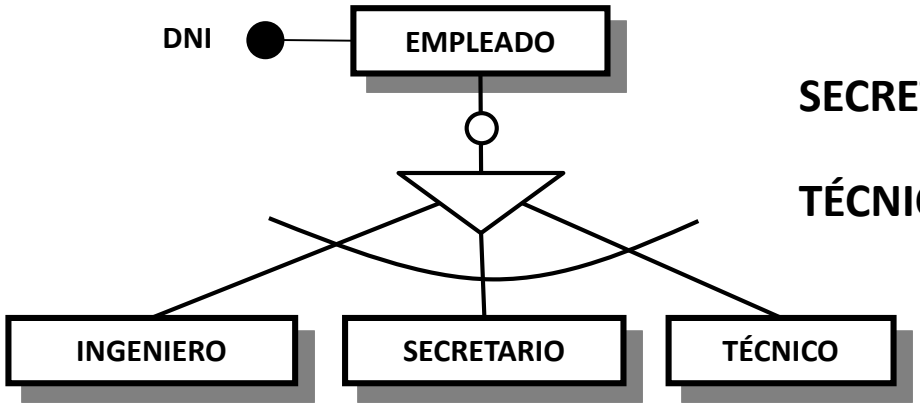
- **Jerarquías Subconjunto y Generalización**
  2. En una única relación con todos los atributos del tipo entidad genérica y de todos los subtipos de relaciones. Se puede añadir un atributo "tipo_de", para no perder información en la transformación
     - Adoptar esta solución cuando los subtipos se diferencian en muy pocos atributos y los tipos de relaciones que los asocian con el resto de los tipos de entidades del esquema sean las mismas para todos (o casi todos) los subtipos.



EMPLEADO (DNI, At. Comunes, tipo_empleado, At. específicos)

Reglas para el diseño Lógico
R5. Transformación de Jerarquías

- **Jerarquías Subconjunto y Generalización**
  3. Transformación en una relación para cada subtipo, con la clave principal, los atributos comunes y los específicos suyos.
     - Adoptar esta solución cuando se dan las mismas condiciones que en el primer caso–muchos atributos distintos- y los accesos realizados sobre los datos de los distintos subtipos siempre afectan a atributos comunes.



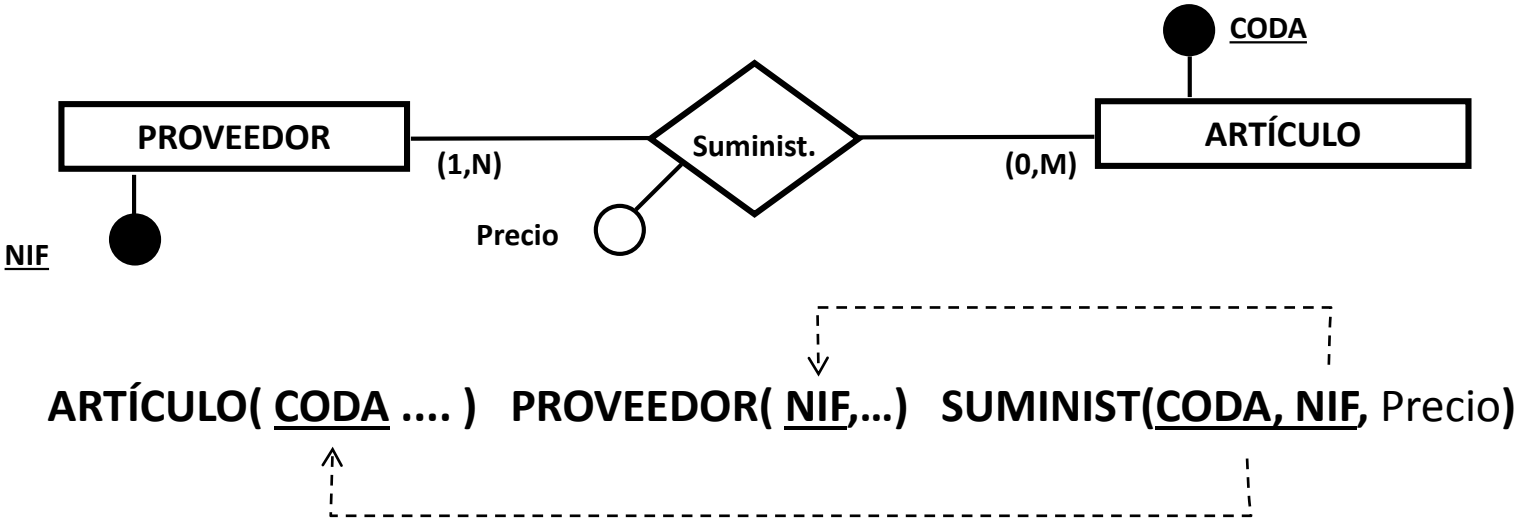
INGENIERO (<u>DNI</u> , At. comunes, At. específicos)

SECRETARIO (<u>DNI</u> , At. comunes, At. específicos)

TÉCNICO (<u>DNI</u> , At. comunes, At. específicos)

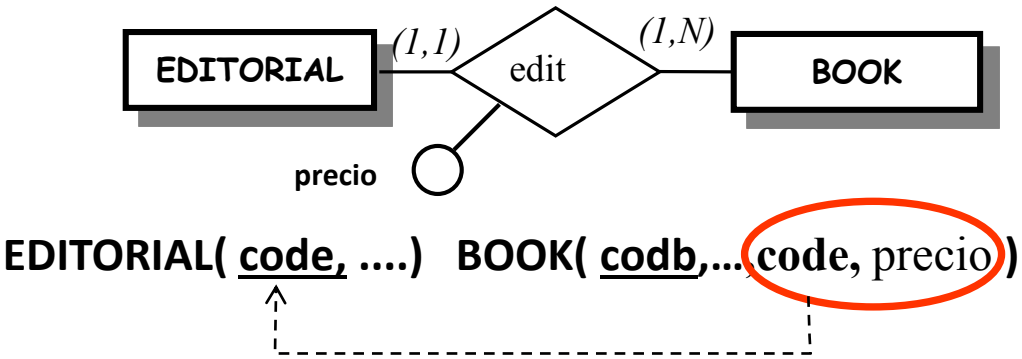
Reglas para el diseño Lógico
R6. Transformación de Atributos de Relaciones

- Si el tipo de relación se transforma en una relación, todos sus atributos pasan a ser atributos de la relación.



Reglas para el diseño Lógico
R6. Transformación de Atributos de Relaciones

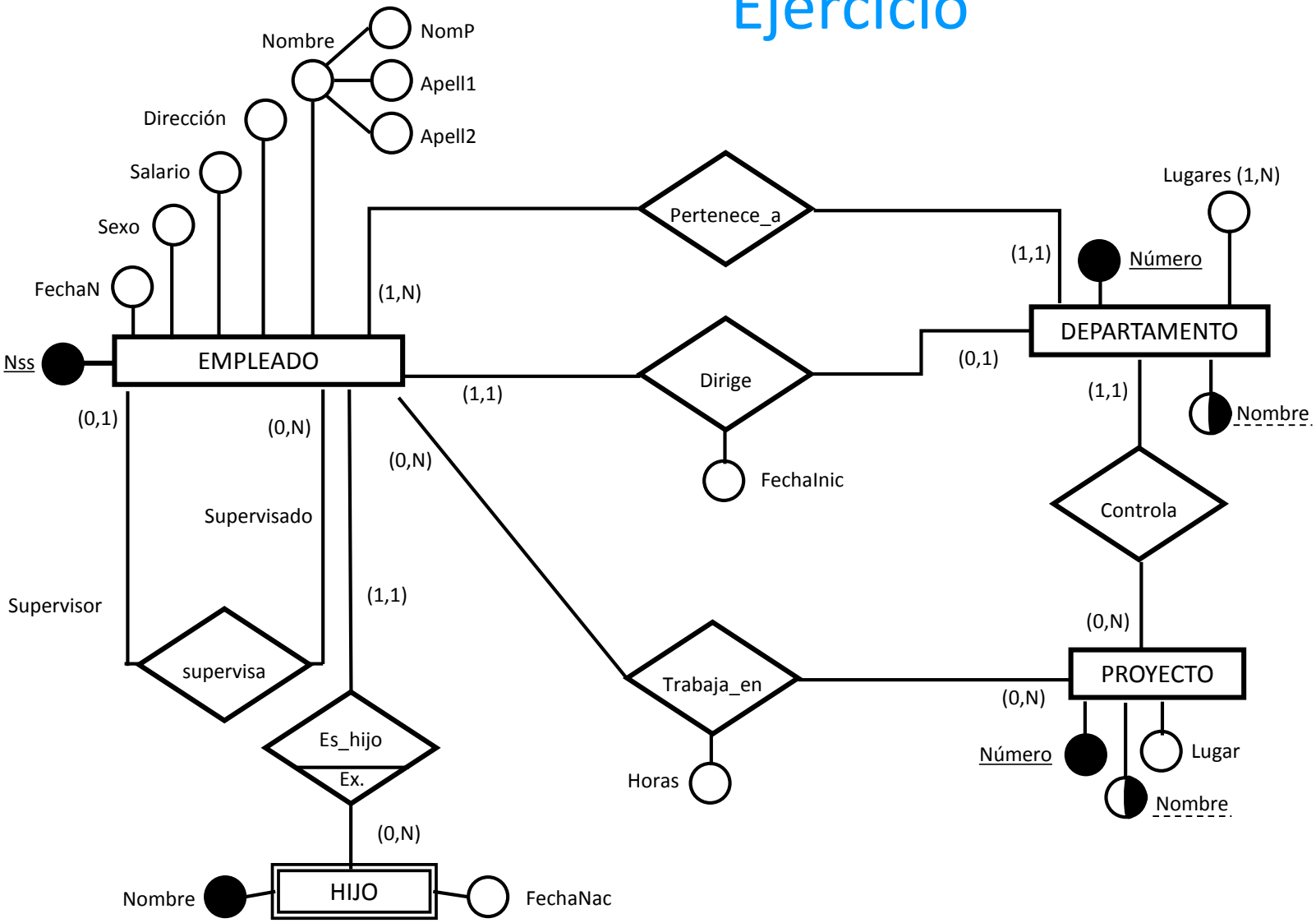
- Si el tipo de relación se transforma mediante propagación de claves, sus atributos migran junto a la clave a la relación del tipo entidad con cardinalidad N.
  - Es decir, en lugar de crear una nueva relación para el tipo de relación, se añaden los atributos de la clave principal del tipo de entidad con cardinalidad máxima a 1 a la relación transformada del tipo de entidad con cardinalidad N.



Reglas para el diseño Lógico
Relaciones múltiples y Entidades Débiles

- Si existe más de un tipo de relación entre n tipos de entidades, se consideran independientes unas de otras, mezclándose las relaciones de la misma entidad en una sola, salvo para tablas procedentes de relaciones N:M o n-arias.
- Las dependencias en **existencia e identificación de entidades débiles no son recogidas directamente en el modelo relacional**. Se realiza de la misma forma que las entidades fuertes, utilizando el mecanismo de la propagación de la clave, creando una clave foránea, con nulos permitidos o no, según el caso, en la tabla de la entidad dependiente, con la característica de obligar a una modificación y un borrado en cascada.

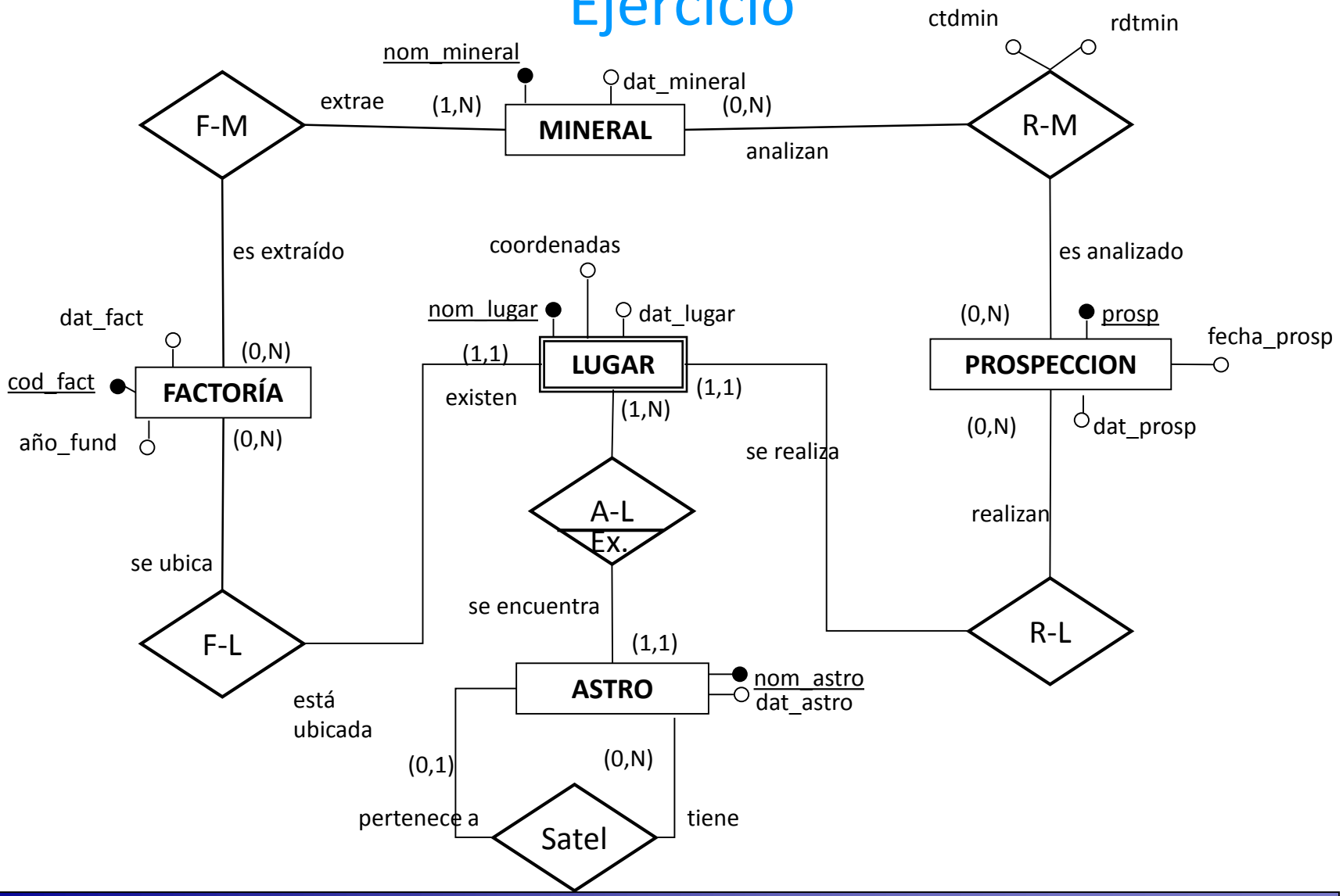
Reglas para el diseño Lógico
Ejercicio



Reglas para el diseño Lógico
Ejercicio (faltan las fechas de los fK)

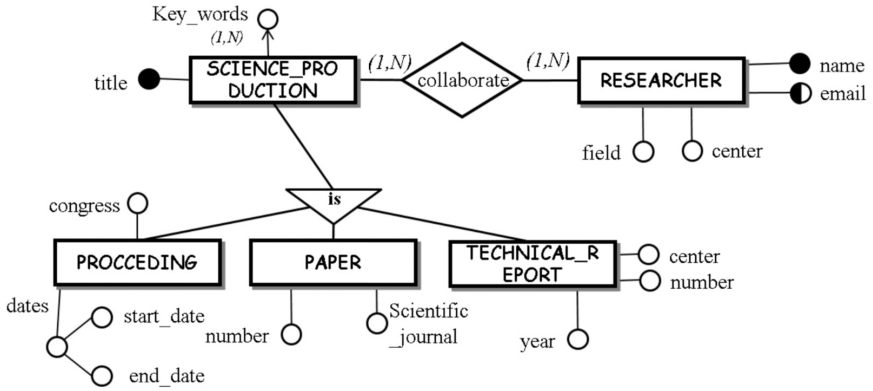
- EMPLEADO(**NSS**, fechan, sexo, salario, direccion, nomp,apell1, apell2,supervisor,n_dep)
  - -- donde supervisor admite nulos. (relación supervisa)
  - -- donde n_dep no admite nulos. (relación pertenece_a)
- HIJO(**nombre**,nss)
  - -- tipo de relación es_hijo => añadir nss a HIJO, pero no es clave.
- DEPARTAMENTO (**numero**, nombre,lugares,director,fechainic)
  - -- tipo de relación dirige => añadir director y fechainic
- PROYECTO (**numero**, nombre, lugar,n_dep)
  - -- por tipo de relación controla: añdimos n_dep NOT NULL
- TRABAJA_EN(**nss,n_proy**, horas)
  - -- Por ser N:M y con atributos

Reglas para el diseño Lógico
Ejercicio



2017 / 2018 (extraordinario)

[Problem 4]



SCIENCE_PRODUCTION: resultados científicos
PROCCEDING: actas de congreso
PAPER: Artículo
TECHNICAL_REPORT: informe técnico
RESEARCHER: investigador.
Scientific_journal: revista científica.
Congress: congreso.
Key_words: palabras clave.

	Is a	Admits	is	referential integrity constraints **(only for FK)**[2]		
Relation	**FK/K**[1]	**Null values**	**Unique**	**References**	**Delete**	**Update**
RESEARCHER(name,	**K**	**No**	**Yes**			
Email,		**No**	**Yes**			
Field,						
center)						
RESEARCHER_SCIENCE_PRODUCTION(name,	**K /FK**	**No**	**No**	RESEARCHER.name	**Yes**	**Yes**
title) collaborate	**K /FK**	**No**	**No**	SCIENCE_PRODUCTION.title	**Yes**	**Yes**
name, title	**K**	**No**	**yes**			
SCIENCE_PRODUCTION (title)	**K**	**No**	**Yes**			
PROCCEDING(title,	**K /FK**	**No**	**Yes**	SCIENCE_PRODUCTION.title	**Yes**	**Yes**
Congress,						
Start_date,						
End_date)						
PAPER(title,	**K /FK**	**No**	**Yes**	SCIENCE_PRODUCTION.title	**Yes**	**Yes**
number,						
Scientific_journal)						
TECHNICAL_REPORT(title,	**K /FK**	**No**	**Yes**	SCIENCE_PRODUCTION.title	**Yes**	**Yes**
number,						
center,						
year)						
KEY_WORD (word)	**K**	**No**	**Yes**			
KEY_WORD_SCIENCE_PRODUCTION (word)	**K**	**No**	**no**			
title)	**K**	**No**	**No**			
word, title	**K**	**No**	**yes**			

[1] FK-Foreign Key/K-key. When the primary key of a relation consists of two or more attributes, then all of them must be identified with a "K".
[2] Indicate if the foreign key has any cascading effect

Where the rows of PROCCEDING, PAPER and TECHNICAL_REPORT share the same code with SCIENCE_PRODUCTIONS. So, for each row of PROCCEDING, PAPER and TECHNICAL_REPORT there is a row in SCIENCE_PRODUCTION with the same title.

science_production (title) borrado/actualización en cascada

Key_word (title, keyword) multivaluado

researcher (name, email, field, center)

paper (title FK, number, scientific_journal) especificación de science_production

proceeding (title FK, congress, start_date, end_date) atributo compuesto

technical_report (title FK, year, center, number)

collaborate (title, name) no admiten nulos, no único, la combinación title name sí es única

---

Relación	FK / K	Null	Unique	References	Cascada

cliente (code, DNI)

reserva (codr, fecha_inicio, fecha_final, codc, coda) — no va a ser nulo, si tienes una reserva tienes una agencia

agencia (coda)

coche (matrícula, codg FK, codr, precio_alquiler) no admite nulos, único

garaje (codg)

guarda (matrícula FK, codg FK) no admiten nulos / cascada

↳ alternativa a poner codg en coche

hace (codr FK, codc FK) no admiten nulos y son únicos

↳ alternativa a poner codc en reserva

monitor (DNI, tlf, nombre, codc)

deporte (nombre, instalación, riesgo)

curso (codc, n-clausus, hora_inicio, fecha_inicio, nivel, nombre) — nombre: no es único

alumno (DNI, nombre, dirección, edad, codc, nombre)

preparado (DNI, nombre)

~~matriculo (DNI, nombre, codc)~~

---

trabajador (DNI, nombre, dirección, tlf, nombreDep, nombreSec) — nombreDep: not null, no único

departamento (nombre, tarea, DNI, codl)

localidad (codl, nombre)

sección (nombre, subtarea, DNI)

dirige (nombre, DNI)

dirige (nombre, DNI)

conyuge (DNI, DNI) !!!

3.1 Descomposición de Esquemas

3.1 Descomposición de Esquemas

- La descomposición de un esquema $\mathbf{R=\{ A_1, A_2, \ldots A_k \}}$ consiste en reemplazarlo por una colección (proyección)

$$\rho = (R_1, R_2, \ldots R_n)$$

de subconjuntos de R de forma que

$$R_1 \cup R_2 \cup \ldots \cup R_n = R$$

donde los R_i no son necesariamente disjuntos.

- Nos interesa que una descomposición de un esquema R sea equivalente a éste:
  - Conservación de la Información (propiedad LJ – Lossless Join)
  - Conservación de las dependencias

  y que los esquemas de la descomposición sean mejores:
  - Mínima redundancia de datos (normalización de las relaciones)

3.1 Descomposición de Esquemas

- Además de la normalización, existen otros criterios para calificar la bondad de un esquema relacional, como:
  - Eficiencia frente a las consultas más comunes.
  - Representar mejor la semántica del mundo real.

No siempre es posible cumplir a la vez todos los objetivos.

Ejemplo:

Proveedores (CodP, Nombre, Direccion, CodA, PVP)

F= { Codp → Nombre, Codp → Direccion, CodP CodA → PVP }

ρ = (R_1, R_2) donde R_1 = Codp, Nombre, Dirección

R_2 = Codp, Coda, PVP

3.1 Descomposición de Esquemas

– Ejemplo de Descomposición con pérdida de Información

LIBROS (Cod_Libro→Editorial; Editorial →País)

Cod_LIbro	Editorial	País
9030	RAMA	ESPAÑA
9110	PARANINFO	ESPAÑA
0167	A.WESLEY	EE.UU.

Descomposición

LIBROS1

Cod_LIbro	País
9030	ESPAÑA
9110	ESPAÑA
0167	EE.UU.

EDITORIALES

Editorial	País
RAMA	ESPAÑA
PARANINFO	ESPAÑA
A.WESLEY	EE.UU.

LIBROS1 * EDITORIALES

Cod_LIbro	Editorial	País
9030	RAMA	ESPAÑA
9030	**PARANINFO**	**ESPAÑA**
9110	**RAMA**	**ESPAÑA**
9110	PARANINFO	ESPAÑA
0167	A.WESLEY	EE.UU.

Tuplas espurias o falsas

Realmente obtenemos todas las tuplas de la relación original, junto con algunas tuplas "falsas" adicionales. Pero no podemos saber cuáles son falsas y cuáles no por lo que hay una "pérdida de información"

3.1 Descomposición de Esquemas
Descomposición preservando dependencias

ESTUDIA (Estu,Asig → Prof; Prof → Asig)

Estudiante	Asignatura	Profesor
López	BBDD	Prof. Blanco
López	Programación	Prof. de Lucas
Huete	BBDD	Prof. Tomás
Huete	Redes	Prof. Sánchez

- Se pierde la DF **Estu,Asig → Prof.**

Descomposición

PA Profesor -> Asignatura

Profesor	Asignatura
Prof. Blanco	BBDD
Prof. de Lucas	Programación
Prof. Tomás	BBDD
Prof. Sánchez	Redes

EP Profesor -> Estudiante

Estudiante	Profesor
López	Prof. Blanco
López	Prof. de Lucas
Huete	Prof. Blanco
Huete	Prof. Sánchez

3.1 Descomposición de Esquemas

Algoritmo Manila: calcular la proyección de un conjunto de dependencias sobre un conjunto de atributos

- ENTRADA:
  - Esquema de la relación (R, F)
  - X, un subconjunto de R, $X \subset R$
- SALIDA:
  - Proyección de F sobre X
- PROCESO:
  - 1) G = cobertura minimal de F
  - 2) W = R – X
  - 3) Mientras ($W \neq \emptyset$) Hacer: (BUCLE)
    - A = un atributo de W ⟶ » W=W – A
    - H ={$ZY \rightarrow V$ / $ZA \rightarrow V \in G \wedge Y \rightarrow A \in G$}, donde Z puede ser $\emptyset$
    - G = G –{DFs $\in$ G / A aparece en Dfs} $\cup$ H
  - $\Pi_X(F) = G$

3.1 Descomposición de Esquemas

Algoritmo: calcular la proyección de un conjunto de dependencias sobre un conjunto de atributos

- EJEMPLO:

  R=(C, D, E, I, J, K, L)

  F ={ $C \rightarrow D$, $E \rightarrow J$, $I \rightarrow J$, $J \rightarrow K$, $K \rightarrow J$, $DK \rightarrow L$} (es minimal)

  Proyección sobre X = {C, E, I, L}

  1. G ={ $C \rightarrow D$, $E \rightarrow J$, $I \rightarrow J$, $J \rightarrow K$, $K \rightarrow J$, $DK \rightarrow L$}; **2**. W = { D, J , K }

  3. Mientras W ≠ Ø

  1 Iteración: (dependencias funcionales implicante e implicado)

  A = D W = {J, K } H = {$CK \rightarrow L$}

  $DK \rightarrow L$, $C \rightarrow D$ ⟹ $CK \rightarrow L$

  H ={$ZY \rightarrow V$ / $ZA \rightarrow V \in G \wedge Y \rightarrow A \in G$};

  $KD \rightarrow L = ZA \rightarrow V$; Z=K, V=L ∧ $C \rightarrow D = Y \rightarrow A$ => $CK \rightarrow L$

  G = G –{DFs ∈ G / A aparece en Dfs} ∪ H

  G = G – { $C \rightarrow D$, $DK \rightarrow L$} ∪ {$CK \rightarrow L$}

  G = {$E \rightarrow J$, $I \rightarrow J$, $J \rightarrow K$, $K \rightarrow J$, $CK \rightarrow L$}

A = K

W = {J}

J -> K
K -> J

J -> J

J -> K
CK -> L

CJ -> L

→ implicado

→ implicante

G = { E -> J, I -> J , CJ -> L }

W = ∅

A = J

CJ -> L
E -> J

CE -> L

CJ -> L
I -> J

CI -> L

G = {CE -> L , CI -> L }

3.1 Descomposición de Esquemas
Algoritmo: calcular la proyección de un conjunto de dependencias sobre un conjunto de atributos

- EJEMPLO:

R=(C, D, E, I, J, K, L)

F ={ $C \rightarrow D$, $E \rightarrow J$, $I \rightarrow J$, $J \rightarrow K$, $K \rightarrow J$, $DK \rightarrow L$}

2 Iteración:

De la interación anterior: G = {$E \rightarrow J$, $I \rightarrow J$, $J \rightarrow K$, $K \rightarrow J$, $CK \rightarrow L$}

A = J W = {K} H = {$I \rightarrow K$, $E \rightarrow K$}

H ={$ZY \rightarrow V$ / $ZA \rightarrow V \in G \wedge Y \rightarrow A \in G$};

$J \rightarrow K = ZA \rightarrow V$; **Z= Ø**, $V=K \wedge E \rightarrow J = Y \rightarrow A \Rightarrow E \rightarrow K$

$\wedge\ I \rightarrow J = Y \rightarrow A \Rightarrow I \rightarrow K$

$\wedge\ k \rightarrow J = Y \rightarrow A \Rightarrow K \rightarrow K$ (reflexiva)

Al tener dos DF, que cumplen $Y \rightarrow A$, se añaden ambas a H.

G = G –{DFs $\in$ G / A aparece en Dfs} $\cup$ H

G = G – {$E \rightarrow J$, $I \rightarrow J$, $J \rightarrow K$, $K \rightarrow J$ } $\cup$ {$I \rightarrow K$, $E \rightarrow K$}

G = {$CK \rightarrow L$, $I \rightarrow K$, $E \rightarrow K$}

3.1 Descomposición de Esquemas

Algoritmo: calcular la proyección de un conjunto de dependencias sobre un conjunto de atributos

- EJEMPLO:

R=(C, D, E, I, J, K, L)

F ={ $C \rightarrow D$, $E \rightarrow J$, $I \rightarrow J$, $J \rightarrow K$, $K \rightarrow J$, $DK \rightarrow L$}

3 Iteración:

A = K W = {} H = {$CI \rightarrow L$, $CE \rightarrow L$}

De la iteración anterior: G = {$CK \rightarrow L$, $I \rightarrow K$, $E \rightarrow K$}

H ={$ZY \rightarrow V$ / $ZA \rightarrow V \in G \wedge Y \rightarrow A \in G$}

$CK \rightarrow L = ZA \rightarrow V$; **Z= C**, $V=L \wedge E \rightarrow K = Y \rightarrow A \Rightarrow CE \rightarrow L$

$\wedge I \rightarrow K = Y \rightarrow A \Rightarrow CI \rightarrow L$

$G = G - \{CK \rightarrow L, I \rightarrow K, E \rightarrow K\} \cup \{CI \rightarrow L, CE \rightarrow L\}$

G = {$CI \rightarrow L$, $CE \rightarrow L$}

4. $\Pi_{\{C,E,I,L\}}(F) = \{CI \rightarrow L, CE \rightarrow L\}$

3.1 Descomposición de Esquemas

- **Descomposición Sin Pérdida de Información (LJ)**
  - En algunos libros en español, se referencia como SPI
  - Dado (R,F) esquema de relación y $\rho = (R_1, R_2, \ldots R_n)$ descomposición de R, decimos que es sin pérdida de información (Lossless Join) con respecto a F, si para toda ocurrencia de relación r de R, satisfaciendo F, se cumple:
    - $R = \Pi_{R1}(r) \bowtie \ldots\ldots\ldots \bowtie \Pi_{Rn}(r)$
  - Es decir, al reconstruir la relación original obtenemos el mismo conjunto de tuplas

3.1 Descomposición de Esquemas
Algoritmo de Ullman Test de descomposición LJ

- ENTRADA:
  - Esquema de la relación R={A_1,A_2,....,A_n}, donde n es el nº de atributos.
  - F Conjunto de dependencias funcionales.
  - ρ ={R_1,R_2,....,R_k} descomposición de R, donde k es nº de esquemas R_i.
- SALIDA:
  - Decisión sobre si ρ es LJ o no.
- MÉTODO:
  - Construimos una tabla (k **x** n), donde la fila i corresponde al esquema R_i y la columna j al atributo A_j
  - En la posición (i,j) situamos:
    - a_j si $A_j \in R_i$, el atributo A_j está en R_i.
    - b_{ij} si $A_j \notin R_i$, el atributo A_j NO está en R_i.

3.1 Descomposición de Esquemas
Algoritmo Test de descomposición LJ

- MÉTODO
  - Transformar la tabla (k **x** n)
    - Para cada DF X→Y de F,
      - Seleccionar las filas de la matriz tales que sus valores coincidan para las columnas (atributos) del implicante de X→Y,
      - Para estas filas, igualar los valores de las columnas (atributos) del implicado Y de la siguiente forma:
        - ◊ Si un símbolo es a_j ponemos todas a a_j
        - ◊ Si todas son b, igualamos sus subíndices a cualquier subíndice de los b afectados.
  - El proceso se repite hasta que la tabla no varíe o nos aparezca una fila con todo aes (a_1,a_2,....a_n).
  - Al terminar: Es LJ si encontramos una fila con todo aes (a_1,a_2,....a_n)
- Teorema
  - El algoritmo determina correctamente si una descomposición es sin pérdida de información

3.1 Descomposición de Esquemas
Algoritmo Test de descomposición LJ

– **Ejemplo:**

R={ABCDE}

F ={A→C, B →C, C → D, DE → C, CE →A}

R1={AD} R2={AB} R3={BE} R4={CDE} R5={AE}

Tabla Inicial:		A	B	C	D	E
R1	AD	a_1	b_{12}	b_{13}		
R2	AB	a_1	a_2	b_{23}		
R3	BE	b_{31}	a_2	b_{33}		
R4	CDE	b_{41}	b_{42}	a_3		
R5	AE	a_1	b_{52}	b_{53}		

3.1 Descomposición de Esquemas
Algoritmo Test de descomposición LJ

- **Ejemplo:**

  R={ABCDE}

  F ={A→C, B →C, C → D, DE → C, CE →A}

  R1={AD} R2={AB} R3={BE} R4={CDE} R5={AE}

Tabla Inicial:		A	B	C	D	E
R1	AD	a_1	b_{12}	b_{13}	a_4	b_{15}
R2	AB	a_1	a_2	b_{23}	b_{24}	b_{25}
R3	BE	b_{31}	a_2	b_{33}	b_{34}	a_5
R4	CDE	b_{41}	b_{42}	a_3	a_4	a_5
R5	AE	a_1	b_{52}	b_{53}	b_{54}	a_5

(A→ C)	A	B	C	D	E
R1	a_1	b_{12}	b_{13}	a_4	b_{15}
R2	a_1	a_2	b_{13}	b_{24}	b_{25}
R3	b_{31}	a_2	b_{33}	b_{34}	a_5
R4	b_{41}	b_{42}	a_3	a_4	a_5
R5	a_1	b_{52}	b_{13}	b_{54}	a_5

3.1 Descomposición de Esquemas

(B→ C)	A	B	C	D	E
R1	a_1	b_{12}	b_{13}	a_4	b_{15}
R2	a_1	$\underline{a_2}$	b_{13}	b_{24}	b_{25}
R3	b_{31}	$\underline{a_2}$	b_{13}	b_{34}	a_5
R4	b_{41}	b_{42}	a_3	a_4	a_5
R5	a_1	b_{52}	b_{13}	b_{54}	a_5

(C→ D)	A	B	C	D	E
R1	a_1	b_{12}	$\underline{b_{13}}$	a_4	b_{15}
R2	a_1	a_2	$\underline{b_{13}}$	a_4	b_{25}
R3	b_{31}	a_2	$\underline{b_{13}}$	a_4	a_5
R4	b_{41}	b_{42}	a_3	a_4	a_5
R5	a_1	b_{52}	$\underline{b_{13}}$	a_4	a_5

(DE→ C)	A	B	C	D	E
R1	a_1	b_{12}	b_{13}	a_4	b_{15}
R2	a_1	a_2	b_{13}	a_4	b_{25}
R3	b_{31}	a_2	a_3	$\underline{a_4}$	$\underline{a_5}$
R4	b_{41}	b_{42}	a_3	$\underline{a_4}$	$\underline{a_5}$
R5	a_1	b_{52}	a_3	$\underline{a_4}$	$\underline{a_5}$

(CE→ A)	A	B	C	D	E
R1	a_1	b_{12}	b_{13}	a_4	b_{15}
R2	a_1	a_2	b_{13}	a_4	b_{25}
R3	a_1	a_2	$\underline{a_3}$	a_4	$\underline{a_5}$
R4	a_1	b_{42}	$\underline{a_3}$	a_4	$\underline{a_5}$
R5	a_1	b_{52}	$\underline{a_3}$	a_4	$\underline{a_5}$

3.1 Descomposición de Esquemas

– **Teorema:**

(lossless-join)

Si ρ =(R1, R2), en una descomposición de (R,F), entonces ρ es LJ respecto a *F* si y sólo si

$$(R1 \cap R2) \rightarrow (R1 - R2) \quad \text{ó} \quad (R1 \cap R2) \rightarrow (R2 - R1)$$

– **Lemas**

(proyección)

- Sea (R, F) un esquema relacional, y sea ρ ={R_1, R_2, ..., R_k} una descomposición de R LJ respecto a F. Si σ ={S_1, S_2, ..., S_m} es una descomposición LJ de R_i respecto a F_i (proyección de F sobre R_i), entonces la descomposición de R en {R_1, ..., R_{i-1}, S_1, ..., S_m, R_{i+1}, ..., R_k} es LJ respecto a F.
- Sea (R, F) y ρ ={R_1, R_2, ..., R_k} descomposición LJ.

  Si τ ={R_1, ..., R_k, R_{k+1}, ...,R_n} es una descomposición que incluye a ρ, entonces τ es LJ respecto a F.

3.1 Descomposición de Esquemas

- **Descomposición preservando las dependencias**
  - La transformación del esquema origen R(A,F) en un conjunto de esquemas Ri (Ai,Fi) debe llevarse a cabo sin pérdida de estas dependencias.
  - El conjunto de dependencias funcionales de partida F debe ser equivalente al conjunto de dependencias funcionales de los esquemas resultantes.
  - Para ello, recordando el concepto de equivalencia entre conjuntos de dependencias funcionales del tema anterior:
    - $(U_{i=1}^{n} F_i)^{+} = F^{+}$

cierre

3.1 Descomposición de Esquemas: Algoritmo Test de Ullman de descomposición preservando dependencias

- ENTRADA:
  - Descomposición de R. ρ ={R_1,R_2,....,R_k}
  - *F* Conjunto de dependencias funcionales.
- SALIDA:
  - Decisión sobre si ρ preserva dependencias o no.
- MÉTODO:
  - Sea G=$(U_{i=1}^{n} F_i)$
  - Queremos ver que G cubre a *F* sin calcular F^+ ni G^+ (son equivalentes)

3.1 Descomposición de Esquemas: Algoritmo Test de Ullman de descomposición preservando dependencias

- MÉTODO:
  - Averiguar si G es un recubrimiento de F.
  - Para cada $X \rightarrow Y$ de *F* verificar si $Y \subset X^+_G$. Para ello, calculamos X^+_G como:
    - Definimos la R_i-Operación sobre Z cjto. de atributos respecto a *F*, como:

      inicial = X

      $\mathbf{Z=Z \cup ((Z \cap R_i)^+_F \cap R_i)}$, donde el cierre se calcula respecto a *F.*
    - Esta R_i-Operación añade a Z los atributos A, para los que se cumple que $\mathbf{(Z \cap R_i)} \rightarrow A \subseteq$ en el conjunto de dependencias F en R_i.
    - Para calcular X^+_G, se comienza con **Z=X**, y para cada esquema R_i se aplica repetidamente las R_i-operación. El cálculo acaba cuando para ninguno de los esquemas R_i, la R_i-Operación produzca un cambio de X^+_G. **Es decir, X^+_G=Z resultante de iterar las R_i-Operaciones**.

3.1 Descomposición de Esquemas: Algoritmo Test de Ullman de descomposición preservando dependencias

- Algoritmo:

  G= $(U^{k}_{i=1}F_i)$, k = nº de particiones.

  Z=X

  Mientras Z cambie

  Desde i=1 hasta k hacer

  $Z=Z\cup((Z \cap R_i)^{+}_{F} \cap R_i)$

  Fin Desde

  FinMientras

  Si $Y \subseteq Z$ Entonces $(X\rightarrow Y) \in (U^{k}_{i=1}F_i)$

$((Z \cap R_i)^{+}_{F} \cap R_i) \Rightarrow$

1. $(Z \cap R_i)$ determinar los atributos de R_i que están en Z.
2. $(Z \cap R_i)^{+}_{F}$ Calcular su cierre en F
3. $((Z \cap R_i)^{+}_{F} \cap R_i)$ obtener los atributos de ese cierre que están en R_i.

  - Si para toda dependencia $\in$ F, se encuentra que $(X\rightarrow Y) \in (U^{k}_{i=1}F_i)$ hay preservación de dependencias.
  - Y, en caso contrario, no la habrá preservación de dependencias.

3.1 Descomposición de Esquemas

- **Ejemplo:**

  R={ABCD}

  F ={A→B, B →C, C → D, D →A}

  R1={AB} R2={BC} R3={CD}

  Donde:

  F1 = $\Pi_{R1}(F)$ = { A → B, B → A} F2 = { B→C, C →B} F3 = { C→D, D →C}

  Siendo $\Pi_{R1}(F)$ la proyección de F sobre R_i. Lo que realmente se proyecta es F^+ y no F. De ahí que, en la $\Pi_{R1}(F)$, aparece (que originalmente estaba en F), y B → A.

  G = { A → B, B → C, C → D, B → A, C → B, D → C }

  El algortimo para el cálculo de $\Pi_{Ri}(F)$ se ve en 3.2 Formas Normales de Codd Algoritmo: calcular la proyec...

3.1 Descomposición de Esquemas

– **Ejemplo:**

R={ABCD}

F ={A→B, B →C, C → D, D →A}

$G = \{ A \to B,\ B \to C,\ C \to D,\ B \to A,\ C \to B,\ D \to C \}$

R1={AB} R2={BC} R3={CD}

$F1 = \Pi_{R1}(F) = \{ A \to B, B \to A\}$ F2 = { B→C, C →B} F3 = { C→D, D →C}

Como {A→B, B →C, C → D} ya están en G, sólo falta comprobar que la dependencia D →A se puede obtener de G

Calculamos las R_i-operaciones sobre Z=X={D}, e iteramos:

Iteración 1:

R1-operación Z={D} ; R2-operación Z={D}; R3-operación Z={C,D}

Iteración 2:

R1-operación Z={C,D} ; R2-operación Z={B,C,D}; R3-operación Z={B,C,D}

Iteración 3:

R1-operación Z={A, B,C,D} está A, SIN PÉRDIDAS

Luego:

$\{D\} \cup ((\{D\} \cap R_i)^+_F \cap R_i) = \{D\} \cup ((D^+_F \cap R_i) = R = D^+_G$

$A \in D^+_G \Rightarrow D \to A \in G^+$ luego preserva dependencias

3.1 Descomposición de Esquemas

- Ejemplo:

R={Ciudad, Dirección, CódigoPostal}

F ={ Ciudad Dirección $\rightarrow$ CódigoPostal, CódigoPostal $\rightarrow$ Ciudad }

Claves: Ciudad Dirección y CódigoPostal Dirección

ρ ={R1,R2}

R1={ Dirección, CódigoPostal } R2={ Ciudad, CódigoPostal }

Esta descomposición es **SIN PÉRDIDA y NO PRESERVA DEPENDENCIAS**

3.1 Descomposición de Esquemas

- **Ejemplo:**

  R={Ciudad, Dirección, CódigoPostal}

  F ={ Ciudad Dirección $\rightarrow$ CódigoPostal, CódigoPostal $\rightarrow$ Ciudad }

  Claves: Ciudad Dirección y CódigoPostal Dirección

  $\rho = \{R_1, R_2\}$

  R_1={ Dirección, CódigoPostal } R_2={ Ciudad, CódigoPostal }

- **Teorema:**

Si ρ =(R1, R2), en una descomposición de (R,F), entonces ρ es LJ respecto a *F* si y sólo si

$$(R1 \cap R2) \rightarrow (R1 - R2) \quad \text{ó} \quad (R1 \cap R2) \rightarrow (R2 - R1)$$

$$\text{CódigoPostal} \rightarrow \text{Ciudad} \Rightarrow \text{es LJ}$$

EJERCICIO

$\rho = (R1, R2, R3)$

$R_1 = \{A, B, C\}$ $R2 = \{A, D, E\}$ $R_3 = \{C, E\}$

$F = \{C \to E, C \to D, AB \to E\}$

$(R1 \cap R2) = \{A\}$

$(R1 - R2) = \{BC\}$

$A \to BC$

$(R2 - R1) = \{DE\}$

$A \to DE$

(con pérdidas)

$(R2 \cap R3) = \{E\}$

$(R2 - R3) = \{AD\}$

$E \to AD$

$(R3 - R2) = \{C\}$

$E \to C$

(sin pérdidas)

3. Normalización. Formas normales de Codd

3.2 Formas Normales de Codd

- La teoría de la normalización se centra en lo que se conoce como "Formas Normales". Se dice que un esquema relacional está en una determinada forma normal si satisface un conjunto específico de restricciones.
- Codd propuso en 1970 tres formas normales basadas en las Dependencias Funcionales. (1FN, 2FN y 3FN)
- Debido a que en 3FN aún persisten ciertos problemas en determinadas relaciones en 1974 Boyce y Codd introdujeron una definición más restrictiva de la 3FN que se denominó Forma Normal de Boyce-Codd (FNBC).
- Cada forma normal conserva las propiedades que verifica la forma normal anterior y exige condiciones más estrictas.

- Minimizar # dependencias o # atributos implicados
- Minimizar # esquemas resultantes (relaciones) y # atributos en las proyecciones

-

3.2 Formas Normales de Codd

- Recordando lo visto en 3.1 Descomposición de Esquemas.
  1. Conservación de la Información (propiedad LJ – Lossless Join)
  2. Conservación de las dependencias
  3. Mínima redundancia de datos (normalización de las relaciones)
- La 3ª propiedad que debe cumplir el conjunto R_i de esquemas redundantes en un proceso de descomposición es que estas relaciones alcancen un nivel de normalización superior al del esquema origen, a fin de eliminar en lo posible las redundancias y, por tanto, las anomalías de actualización.
- Los esquemas R_i además de ser equivalentes a R, deben ser mejores en el sentido de haber alcanzado una forma normal superior.

3.2 Formas Normales de Codd
Definiciones

- Del tema anterior, se dice que:
  - Una dependencia funcional X → Y es trivial si $Y \subseteq X$
  - dependencia funcional completa: X → Y si
    - $X \rightarrow Y \wedge \nexists X' \subset X / X' \rightarrow Y$
  - Y depende funcionalmente en forma elemental de X si
    - X→ Y es una DF completa e
    - Y es simple (un solo atributo).
- Dado un esquema relacional (R,F), un atributo A de R se dice principal si pertenece a una clave de R. Los atributos no pertenecientes a ninguna clave se denominan no principales.
- Clave simple => está formada por un único atributo

3.2 Formas Normales de Codd
FNBC (Forma Normal de Boyce-Codd)

- **Ejemplo**: ESTUDIA (Estu, Asig, Prof)

Estu	Asig	Prof
López	BBDD	Prof. Blanco
López	Programación	Prof. de Lucas
Huete	BBDD	Prof. Inés S.
Huete	Redes	Prof. Sánchez

- Requisitos:
  - Cada profesor imparte sólo una asignatura. (pero una asignatura puede ser impartida por varios profesores)
  - A un estudiante en una asignatura sólo le da clase un profesor
  - Determinar
    - las Restricciones y sus Dependencias Funcionales
    - Claves candidatas? Estudiante (no está implicado)
      Asignatura o (Estudiante, Profesor)

Profesores -> Asignatura (no al revés)

Estudiante -> Asignatura

Profesores -> Estudiante

Si el implicante es único, no tiene un subconjunto de atributos que implique al implicado.

Atributo no principal no aparece en ninguna clave candidata.

3FN: R está en 2FN, ningún campo no principal de R depende transitivamente de alguna clave candidata de R.

↳ no está cuando algún atributo no principal depende funcionalmente de atributos que no son clave

- claves candidatas: en su cierre obtenemos todos los atributos. (únicos que son implicantes)
- 2FN: dependencias funcionales completas (si el implicante es un subconjunto de la clave principal NO está en 2FN)

↳ claves candidatas unarias => directamente en 2FN

FNBC: todo implicante debe ser una clave candidata de la relación, está en 3FN

Si la relación no tiene DF estará en FNBC

todo determinante es clave candidata

proyectar en otras dos relaciones (R1, R2)

- R1: formada por atributos de la DF cuyo determinante no es clave candidata.
- R2: atributos de R menos los que forman el implicado de la DF que no cumple la FNBC.

no trivial: el implicado no forma parte del implicante

3.2 Formas Normales de Codd
FNBC (Forma Normal de Boyce-Codd)

- **Ejemplo**: ESTUDIA (Estu, Asig, Prof)

Estu	Asig	Prof
López	BBDD	Prof. Blanco
López	Programación	Prof. de Lucas
Huete	BBDD	Prof. Inés S.
Huete	Redes	Prof. Sánchez

  - Determinar las Restricciones y sus Dependencias Funcionales:
    - A un estudiante en una asignatura sólo le da clase un profesor.
      - { Estu,Asig $\rightarrow$ Prof}
    - Cada profesor imparte sólo una asignatura. (pero una asignatura puede ser impartida por varios profesores)
      - {Prof $\rightarrow$ Asig}
  - Tendríamos dos claves candidatas:
    - {Estu,Asig} y {Estu,Prof}

3.2 Formas Normales de Codd
FNBC (Forma Normal de Boyce-Codd)

- Las DFs derivadas de las restricciones son las siguientes:

  { Estu,Asig → Prof; Prof → Asig}

  y las claves son: {Estu,Asig} y {Estu,Prof}

  De la 1FN → FNBC

  De la FNBC → 1FN

- Una relación se encuentra en FNBC si, y sólo si para toda dependencia no trivial X → Y, X es una clave candidata o superclave de la relación.
- No está en FNBC porque en Prof → Asig el determinante no es clave
- Al no estar en FNBC la relación presenta ciertas anomalías de actualización;
  - por ejemplo, si queremos eliminar la información de que Huete estudia Redes no podemos hacerlo sin perder la información de que el Prof. Sánchez enseña Redes, al ser el único alumno en la asignatura.

3.2 Formas Normales de Codd
3FN (Tercera Forma Normal)

- En algunas circunstancias, la FNBC es una condición demasiado fuerte, en el sentido de que, en ocasiones, no es posible descomponer el esquema en subesquemas en FNBC **preservando dependencias**.
- Un esquema relacional (R,F) está en Tercera Forma Normal si siempre que $X \rightarrow A$ no trivial se verifica en (R,F) entonces X es clave o superclave o **A es principal**.

- **SI (R,F) ESTÁ EN FNBC $\Rightarrow$ (R,F) ESTÁ EN 3FN**

3.2 Formas Normales de Codd
3FN (Tercera Forma Normal)

- **Ejemplo**

  R={Ciudad, Dirección, CódigoPostal}

  F ={ Ciudad Dirección → CódigoPostal, CódigoPostal → Ciudad }

  Claves: {Ciudad Dirección } y {CódigoPostal, Dirección}

  ρ ={R1,R2}

  R1={ Dirección, CódigoPostal } R2={ Ciudad, CódigoPostal }

  - (R,F) no está en FNBC ya que en CódigoPostal → Ciudad, Código no es clave, y sí en 3FN puesto que Ciudad es principal.
  - Sin embargo en la descomposición ρ los esquemas están en FNBC pero NO PRESERVA DEPENDENCIAS

  - Sin pérdida de info / Pérdida de DF

3.2 Formas Normales de Codd nomenclatura.

- Si tenemos una Dependencia Parcial $Y \rightarrow A$ ($Y \subset X$, siendo X clave), entonces toda tupla usada para asociar un X-valor con valores de otros atributos además de A y X, debe aparecer la misma asociación entre Y y A.

  R=(Nombre, Dirección, Artículo, Precio) **Clave**: Nombre,Artículo

  F ={ Nombre $\rightarrow$ Dirección, Nombre,Artículo $\rightarrow$ Precio }

  Y={nombre}; X= {nombre, artículo}; A={dirección}

- Si tenemos una Dependencia Transitiva $X \rightarrow Y \rightarrow A$, entonces no podemos asociar un Y-valor con un X-valor a menos que haya un A-valor asociado con un Y-valor.

  R=(Tienda, Artículo, NºDpto, Encargado) **Clave**: {Tienda,Artículo}

  F ={ Tienda Artículo $\rightarrow$ NºDpto, Tienda NºDpto $\rightarrow$ Encargado }

  Dependencia Transitiva: Tienda Artículo $\rightarrow$ Tienda NºDpto $\rightarrow$ Encargado

  Y={Tienda, NºDpto}; X= {Tienda Artículo}; A={Encargado}

3.2 Formas Normales de Codd
2FN (Segunda Forma Normal)

- Si X $\rightarrow$ A viola la 3FN (**ni** X es clave o superclave **ni** A es principal), entonces siendo A no principal pueden darse uno de los dos siguientes casos:
  - X es un subconjunto propio de una clave. En este caso decimos que X $\rightarrow$ A es una **Dependencia Parcial**. **NO está en 2FN**.
  - X **no** es un subconjunto de una clave. En este caso decimos que X $\rightarrow$ A es una **Dependencia Transitiva**. **Está en 2FN pero no en 3FN**
    - También se dice que existe **dependencia es total:** A depende de alguna clave candidata o principal de forma directa o tiene una dependencia transitiva.

3.2 Formas Normales de Codd
2FN (Segunda Forma Normal)

- Un esquema (R,F) está en 2FN si **NO** tiene dependencias parciales. (subconjunto del implicante que también implicara)
- Podríamos definir en orden inverso
  - la 2FN como que:
    - cada atributo no principal depende funcionalmente de manera total (completa) de cada una de las claves de la relación,
  - y la 3FN
    - si está en 2FN y **no** existe ningún atributo no principal que dependa transitivamente de alguna clave de la relación.

3.2 Formas Normales de Codd
2FN (Segunda Forma Normal)

- Se puede afirmar que se encuentran en 2FN:
  - Cualquier relación de dos atributos.
    - Dos opciones:
      - Depende un atributo del otro ⇒ no es dependencia parcial
      - No dependen de ningún atributo ⇒ no hay dependencias (FNBC)
  - Cualquier relación en las que todas las claves son simples (tienen un solo atributo).
    - No existen dependencias parciales.
    - Al ser las claves simples ⇒ cada atributo no principal NO tienen dependencias parciales: o su implicante es una clave o no forma parte de una clave.
  - Cualquier relación en la que todos sus atributos son principales (forman parte de alguna clave). => 2FN

3.2 Formas Normales de Codd
2FN (Segunda Forma Normal)

- Ejemplo

  R=(Nombre, Dirección, Artículo, Precio)

  F ={ Nombre $\rightarrow$ Dirección, Nombre,Artículo $\rightarrow$ Precio }

  Clave: ¿?

  – ¿está en 3FN? ¿en 2FN?

3.2 Formas Normales de Codd
2FN (Segunda Forma Normal)

- Ejemplo

  R=(Nombre, Dirección, Artículo, Precio)

  F ={ Nombre $\rightarrow$ Dirección, Nombre,Artículo $\rightarrow$ Precio }

  Clave: Nombre,Artículo

  – *Dirección* no principal en *Nombre* $\rightarrow$ *Dirección* con el determinante *Nombre* subconjunto propio de una clave, luego la dependencia *Nombre* $\rightarrow$ *Dirección* es parcial y por tanto:

    (R,F) **no** está en 3FN ni en 2FN

3.2 Formas Normales de Codd
2FN (Segunda Forma Normal)

- Ejemplo

  R=(Tienda, Artículo, NºDpto, Encargado)

  F ={ Tienda Artículo $\rightarrow$ NºDpto, Tienda NºDpto $\rightarrow$ Encargado }

  Clave: Tienda Artículo

  – Tienda NºDpto no superclave y Encargado no principal $\Rightarrow$ no 3FN.

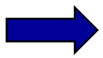
  – No tiene Dependencias Parciales (Tienda NºDpto $\not\subset$ Tienda Artículo) luego si está en 2FN.

3.2 Formas Normales de Codd
1FN (Primera Forma Normal)

- Una relación está en 1FN cuando:
  - los dominios de cada atributo están formados por valores atómicos (no compuestos ni multivaluados).
  - Es una restricción inherente al modelo relacional, por lo que su cumplimiento es obligatorio.
  - En una relación **no** se admiten grupos o valores multivaluados.

COD	CLUB	COMPETICIÓN
FCB	FC Barcelona	Liga Champions
VCF	Valencia C.F	Liga UEFA

NO EN 1FN



COD	CLUB	COMPETICIÓN
FCB	FC Barcelona	Liga
FCB	FC Barcelona	Champions
VCF	Valencia C.F	Liga
VCF	Valencia C.F	UEFA

SI EN 1FN

(3/4/19)

$R = \{A, B, C, D, E\}$

$F = \{AB \to, C \to E, E \to C, C \to D, AB \to E\}$

- Atributos simples (solo hay uno en el implicado)
- No hay atributos externos o extraños (cierres)
- DF redundantes $AB \to C$, $C \to E$ $\Rightarrow AB \to E$

  $C \to E$, $E \to C$ es equivalente no redundante

- **FNBC**: todo implicante debe ser clave

  L> **CLAVE** $= \{A, B\}$

  - AB sólo implicantes => su cierre da R
  - D sólo implicado
  - CE implicante e implicado

  - No está en FNBC (todos los implicantes no son claves)

- **3FN**: todos los implicantes son clave o son principales

  C, E y D no son clave

- **2FN**: no hay dependencias completas (el implicante es un subconjunto de la clave no está en 2FN).

3.2 Formas Normales de Codd
Algoritmo de Descomposición LJ en FNBC

- ENTRADA:
  - Esquema de la relación $R=\{A_1,A_2,....,A_n\}$
  - F Conjunto de dependencias funcionales.
- SALIDA:
  - Una descomposición de R sin pérdida de información en la que cada esquema está en FNBC con respecto a la proyección de F sobre tal esquema.
- MÉTODO:
  - Construimos iterativamente la descomposición ρ
  - Inicialmente ρ = {R}

3.2 Formas Normales de Codd
Algoritmo de Descomposición LJ en FNBC

- MÉTODO:
  - Inicialmente ρ = {R}
  - Si S es un esquema en ρ y no está en FNBC, consideramos $X \rightarrow A$ de la proyección de F en S que viole la FNBC
    - X no es clave
  - Reemplazamos (S,F) por (S_1,F_1)y (S_2,F_2)donde
    - $S_1 = AX\ (X \xrightarrow{DF\ (FNBC)} A)(S1 \subset S)$ F1 proyección de F en S_1
    - $S_2 = S - \{A\}$ $(S2 \subset S)$ F2 proyección de F en S_2
  - Esta descomposición es LJ pues $(S_1 \cap S_2) = X \rightarrow (S_1 - S_2) = A$ y por el apartado a) del lema de descomposición esta propiedad se conserva al reemplazar S por S_1 y S_2
  - Al ir reduciendo atributos llegará un momento en que todos los esquemas están en FNBC

3.2 Formas Normales de Codd
Algoritmo de Descomposición LJ en FNBC

- EJEMPLO:

  R=(**C**urso, **P**rofesor, **H**ora, **A**ula, **E**studiante, **N**ota)

  R=(C,P,H,A,E,N)

  F ={ C → P, HA → C, HP → A, CE → N, HE → A }

  Clave: HE

  ρ1 = {CEN, CP, CHA, HE}= $\{R_1,R_2,R_3,R_4\}$

  ρ2 = {CEN, CP, CHA, HAE}

  $H^+ = H$

  $A^+ = A$

  - todos los atributos son simples

  $P^+ = P$

  $C^+ = C,P$

  - **NO PRESERVAN DEPENDENCIAS FUNCIONALES**.

- no todas las implicantes son clave
- no hay equivalentes
- implicadas únicamente en 1 DF : N

S_1 : CE → N

$\{C, P, H, A\}$ (menos 1)

$\{C \to P,\ HA \to C,\ HP \to A,\ HE \to A\}$

- no está en FNBC

S_{21} : $C \to P$

S_{22} : $\{CHAE\}$

- Normalizar sin pérdidas de datos

$R = \{A, B, C, D, E\}$

$F = \{AB \to C,\ C \to E,\ E \to C,\ C \to D\}$

- Comprobar si está en FNBC

clave: $\{A\ B\}$

- $C \to D$
- $C \to E$

$p_1 = \{CD, CE\}$

$p_2 = \{ABCE, ABC\}$ — dejar C para recomponer

- $E \to C$

$p_1 = EC$

$p_2 =$ ninguna DF $\emptyset$

3.2 Formas Normales de Codd
Algoritmo de Descomposición LJ en FNBC

- Consideraciones al aplicar el algoritmo :
  - Es relevante el orden en que se van tomando los grupos de dependencias:
    - Siempre que sea posible, las primeras dependencias que se elegirán en el proceso de descomposición serán aquellas cuyo implicado no sea implicante en otra dependencia.
  - Atributos equivalentes:
    - Al seleccionar una dependencia que defina atributos equivalentes (atributos que dependen mutuamente uno de otro A ↔ B) incluiremos en el nuevo esquema que se crea no sólo los atributos de la dependencia por la que se crea, sino todos aquellos que sean equivalentes.

3.2 Formas Normales de Codd: Algoritmo de Descomposición Preservando Dependencias y LJ en 3FN

- ENTRADA:
  - Esquema de la relación R ={A_1,A_2,....,A_n}
  - F Conjunto de dependencias funcionales cobertura minimal
- SALIDA:
  - Una descomposición de R que preserva dependencias en la que cada esquema está en 3FN con respecto a la proyección de F sobre tal esquema.
- MÉTODO:
  - Si hay algún atributo de R que no está en ninguna dependencia de F, este forma un esquema de relación por sí mismo y lo eliminamos de R.

3.2 Formas Normales de Codd: Algoritmo de Descomposición Preservando Dependencias y LJ en 3FN

- MÉTODO:
  - Se agrupan las DFs de F en particiones que tengan el mismo implicante. (consideramos a los descriptores equivalentes como el mismo implicante)
  - En otro caso, la descomposición consiste en un esquema {X,Y para cada dependencia $X \rightarrow Y$ de F.
  - Si ningún esquema contiene la clave de la relación inicial añadimos la clave como otro esquema de relación.
- Teorema:
  - El algoritmo da una descomposición en 3FN que preserva dependencias

3.2 Formas Normales de Codd: Algoritmo de Descomposición Preservando Dependencias y LJ en 3FN

- Algoritmo:

i:=0

Para cada Df X→Y de F hacer

Si ningún esquema R_j, j = 1,2,... i contiene XY entonces

i:= i+1;

R_i := XY;

FinSi

FinPara

Si ninguno de los esquemas R_j, j=1,..., i contiene una clave candidata de R Entonces

i := i+1;

R_i := cualquier clave candidata de R

FinSi

Devuelve (R_1, R_2, ...,R_i)

3.2 Formas Normales de Codd: Algoritmo de Descomposición Preservando Dependencias y LJ en 3FN

- TEOREMA:
  - Sea ρ una descomposición de (R,F) en 3FN construida por el algoritmo anterior, y sea X una clave de (R,F). Entonces la descomposición $\tau = \rho \cup \{X\}$ es una descomposición de (R,F) con todos sus esquemas en 3FN que preserva dependencias y es sin pérdida de información.
- EJEMPLO:

  R=(Curso, Profesor, Hora, Aula, Estudiante, Nota)=(C,P,H,A,E,N)

  F ={ $C \rightarrow P$, $HA \rightarrow C$, $HP \rightarrow A$, $CE \rightarrow N$, $HE \rightarrow A$ }

  Clave: ¿?

3.2 Formas Normales de Codd: Algoritmo de Descomposición Preservando Dependencias y LJ en 3FN

- EJEMPLO:

  R=(Curso, Profesor, Hora, Aula, Estudiante, Nota)=(C,P,H,A,E,N)

  F ={ C $\rightarrow$ P, HA $\rightarrow$ C, HP $\rightarrow$ A, CE $\rightarrow$ N, HE $\rightarrow$ A }

  Clave: HE

  ρ = {CP, HAC, HPA, CEN, HEA} $\cup$ {HE}

  como HE $\subset$ HEA puede ser eliminado de ρ

  - atributo de R que no está en ninguna dependencia: ninguno.
  - particiones que tengan el mismo implicante.: ninguno
  - un esquema {X,Y para cada dependencia X$\rightarrow$Y de F.
    - {CP, HAC, HPA, CEN, HEA}
  - Si ningún esquema contiene la clave de la relación inicial añadimos la clave como otro esquema de relación.: ya está.

3.2 Formas Normales de Codd: Algoritmo de Descomposición Preservando Dependencias y LJ en 3FN

- EJEMPLO:

  R=(Curso, Profesor, Hora, Aula, Estudiante, Nota, Teléfono, Despacho) = (C,P,H,A,E,N,T,D)

  F ={ HA $\rightarrow$ C, HP $\rightarrow$ A, CE $\rightarrow$ N, HE $\rightarrow$ A, HE $\rightarrow$ C, D $\rightarrow$ T
  P $\rightarrow$ T, P $\rightarrow$ D, T $\rightarrow$ P, D $\rightarrow$ P }

  Redundantes

  – Cobertura Minimal:

    F1 ={ HA $\rightarrow$ C, HP $\rightarrow$ A, CE $\rightarrow$ N, HE $\rightarrow$ A, P $\rightarrow$ T, P $\rightarrow$ D,
    T $\rightarrow$ P, D $\rightarrow$ P }

  – Las dependencias no redundantes entre profesor, despacho, y teléfono pueden ser otras

3.2 Formas Normales de Codd: Algoritmo de Descomposición Preservando Dependencias y LJ en 3FN

- **Determinación de la Forma Normal**:
  - atributos equivalentes:
  - Atributos IMPLICANTES:
  - Atributos IMPLICADOS:
  - CLAVES:
  - ATRIBUTOS NO PRINCIPALES:
  - A no depende de la totalidad de la clave, luego la relación R NO ESTÁ EN 2FN

$R = (C,P,H,A,E,N,T,D)$

$F1 = \{ HA \rightarrow C,\ HP \rightarrow A,\ CE \rightarrow N,$
$HE \rightarrow A,\ P \rightarrow T,\ P \rightarrow D,$
$T \rightarrow P,\ D \rightarrow P \}$

- **DESCOMPOSICIÓN**:

i:=0
Para cada Df X→Y de F hacer
  Si ningún esquema R_j, j = 1,2,... i contiene XY entonces
    i:= i+1;
    R_i := XY;
  FinSi
FinPara

Si ninguno de los esquemas R_j, j=1,..., i contiene una clave candidata de R
Entonces
  i := i+1;
  R_i := cualquier clave candidata de R
FinSi

3.2 Formas Normales de Codd: Algoritmo de Descomposición Preservando Dependencias y LJ en 3FN

Determinación de la Forma Normal:

- P, D y T son atributos equivalentes, nos quedamos sólo con P
- IMPLICANTES: H, E, P, A, C
- IMPLICADOS: A, C, N
- CLAVES: HEP, HED, HET (una por cada atributo equivalente)
- ATRIBUTOS NO PRINCIPALES: N, C, A

A no depende de la totalidad de la clave, luego la relación ***NO ESTÁ EN 2FN***

DESCOMPOSICIÓN:

R1 (H, A, C; HA $\rightarrow$ C) Curso por aula y hora

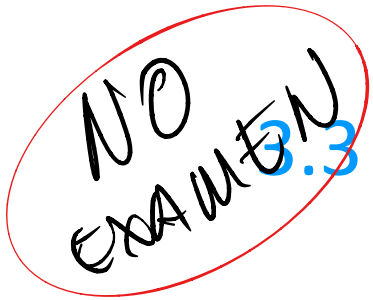
R2 (H, P, A; HP $\rightarrow$ A) Aula por profesor y hora

R3 (C, E, N; CE $\rightarrow$ N) Nota por asignatura y estudiante

R4 (H, E, A; HE $\rightarrow$ A) Aula por estudiante y nota

R5 (P, D, T; P $\leftrightarrow$ D, P $\leftrightarrow$ T) Profesor con despacho y teléfono

R6 (H, E, P) Esquema procedente de la clave



3.3 Formas Normales Avanzadas

– Consideramos la siguiente relación no normalizada

Asignatura	Profesor	Texto
Física	Sr. Sánchez Sra. Hidalgo	Mecánica Clásica Física General
Matemáticas	Sra. Hidalgo Sr. Martín	Álgebra Lineal Calculus

✓ Donde Profesor, Asignatura y Texto son independientes entre sí.

✓ La "Sra. Hidalgo" no me da siempre el mismo valor de Texto ni de Asignatura.

3.3 Formas Normales Avanzadas

- Como hay atributos multivaluados, si lo transformamos a 1FN, se obtiene:

Asignatura	Profesor	Texto
Física	Sr. Sánchez	Mecánica Clásica
Física	Sr. Sánchez	Física General
Física	Sra. Hidalgo	Mecánica Clásica
Física	Sra. Hidalgo	Física General
Matemáticas	Sra. Hidalgo	Álgebra Lineal
Matemáticas	Sra. Hidalgo	Calculus
Matemáticas	Sr. Martín	Álgebra Lineal
Matemáticas	Sr. Martín	Calculus

- En este caso, NO existen DF no triviales. Por tanto:
  - una vez normalizada, al no tener DF esta tabla estará en FNBC.
  - **¡¡¡SIN EMBARGO EXISTE REDUNDANCIA DE DATOS!!!**

3.3 Formas Normales Avanzadas

- En relaciones en FNBC, a veces siguen presentándose redundancias que provocan anomalías de actualización.
- Hasta ahora se han considerado las dependencias funcionales, por las que un valor de un cjto. de atributos X determina el valor de otro cjto. de atributos Y, $X \rightarrow Y$.
- Pero pueden considerarse otros tipos de dependencias en las que el valor de un cjto. de atributos X determina un conjunto bien definido de **posibles** valores de un atributo Y.
- Esto conduce a una generalización de las dependencias funcionales, apareciendo nuevos tipos de dependencias como las multivaluadas y las de combinación.
- Podemos considerar las dependencias funcionales como un caso especial de multivaluadas en el que el número de repeticiones es 1.

3.3 Formas Normales Avanzadas

- Las dependencias multivaluadas (DM) se originan cuando en una relación aparecen atributos multivaluados independientes entre sí, y se normaliza para que esté en 1FN.
- Dada la relación R, se cumple la DM $X \rightarrow\rightarrow Y$ (X multidetermina a Y) si, para cada valor de X, hay un conjunto de cero o más valores de Y, independientemente de los valores del resto de atributos {R – X – Y}.
- Esta independencia del resto de atributos origina la siguiente regla:
  - Las DM siempre se producen por parejas:
    - Si en el esquema R existe la dependencia $X \rightarrow\rightarrow Y$, al mismo tiempo deberá cumplirse $X \rightarrow\rightarrow R - \{X\ Y\}$.
    - Se representa como $X \rightarrow\rightarrow Y \mid R - \{X\ Y\}$.

3.3 Formas Normales Avanzadas

μ[X] es la proyección de la tupla μ sobre esos atributos.

- Definición formal de dependencia multivaluada:
  - Dado el esquema R, la dependencia multivaluada $X \rightarrow\rightarrow Y$ se verifica en R si, y sólo si, para toda tupla μ y υ de cualquier estado (extensión) r de R tales que

    $\mu[X] = \upsilon[X]; \mu[Y] \neq \upsilon[Y]$ – condición de multivaluado

    $\mu[R–X–Y] \neq \upsilon[R–X–Y]$ – casi condición de independiente
  - Existen, necesariamente, dos tuplas τ y ω de r tales que:

    $\mu[X] = \upsilon[X] = \tau[X] = \omega[X]$

    $\tau[Y] = \mathbf{\upsilon}[Y]$ y $\tau[R–X–Y] = \mathbf{\mu}[R–X–Y]$ - MD

    $\omega[Y] = \mathbf{\mu}[Y]$ y $\omega[R–X–Y] = \mathbf{\upsilon}[R–X–Y]$
  - No necesariamente X e Y disjuntos

Se han intercambiado los valores de Y

3.3 Formas Normales Avanzadas
DM - Dependencia multivaluada

- Asig→ → Prof (X→→ Y) y Asig→ → Texto (X →→ R – {X Y}.) son DM.
  - Ya que el conjunto de valores de *profesor* es el mismo en función del atributo *texto*.
  - Ya que el conjunto de valores de *texto* es el mismo en función del *Profesor*.

	Asignatura	Profesor	Texto
ω	Física	Sr. Sánchez	Mecánica Clásica
μ	Física	Sr. Sánchez	Física General
υ	Física	Sra. Hidalgo	Mecánica Clásica
τ	Física	Sra. Hidalgo	Física General
	Matemáticas	Sra. Hidalgo	Álgebra Lineal
	Matemáticas	Sra. Hidalgo	Calculus
	Matemáticas	Sr. Martín	Álgebra Lineal
	Matemáticas	Sr. Martín	Calculus

3.3 Formas Normales Avanzadas
DM

- Ejemplo sin DM
  - Curso → → Texto no es DM.
    - Ya que el conjunto de valores de *texto* es distinto en función de *Idioma*.
    - ω?

	Curso	Texto	Idioma
μ	A1	Introducción a BBDD	Español
υ	A1	Introducción a BBDD	Inglés
τ	A1	Modelo relacional	Español
	B1	Concepción y diseño de BBDD	Francés
	B1	Concepción y diseño de BBDD	Español

3.3 Formas Normales Avanzadas

– Ejemplo:

Curso	Profesor	Hora	Aula	Estudiante	Nota
C101	Sr. Sánchez	L9	8	Antonio	7
C101	Sr. Sánchez	L9	8	Luís	5
C101	Sr. Sánchez	V9	7	Antonio	7
C101	Sr. Sánchez	V9	7	Luís	5
C101	Sr. Sánchez	X9	6	Antonio	7
C101	Sr. Sánchez	X9	6	Luís	5

DF: $C \leftrightarrow P$

Se dan las dependencias multivaluadas:

$$C \rightarrow\!\!\!\rightarrow HA,\ C \rightarrow\!\!\!\rightarrow EN,\ HA \rightarrow\!\!\!\rightarrow EN$$

3.3 Formas Normales Avanzadas

– **Axiomas para Dependencias Multivaluadas:**

- **A1. Reflexividad para DF**
- **A2. Aumento para DF**
- **A3. Transitividad para DF**

} **Axiomas de Armstrong DF**

- **A4. Complementación para DM**

$$\mathbf{X} \rightarrow\!\!\!\!\rightarrow Y \Rightarrow \mathbf{X} \rightarrow\!\!\!\!\rightarrow U - X - Y$$

- **A5. Aumento para DM**

$$\mathbf{X} \rightarrow\!\!\!\!\rightarrow Y, V \subseteq W \Rightarrow W\mathbf{X} \rightarrow\!\!\!\!\rightarrow VY$$

- **A6. Transitividad para DM**

$$\mathbf{X} \rightarrow\!\!\!\!\rightarrow Y, \mathbf{Y} \rightarrow\!\!\!\!\rightarrow Z \Rightarrow \mathbf{X} \rightarrow\!\!\!\!\rightarrow Z - Y$$

- **A7. Replicación**

$$\mathbf{X} \rightarrow Y \Rightarrow \mathbf{X} \rightarrow\!\!\!\!\rightarrow Y$$

- **A8. Fusión:**

$$\{\mathbf{X} \rightarrow\!\!\!\!\rightarrow Y, Z \subseteq Y \text{ y } W \rightarrow Z, W \cap Y = \emptyset\} \Rightarrow \mathbf{X} \rightarrow Z$$

3.3 Formas Normales Avanzadas

– **No se cumple la transitividad al estilo DF:**

Curso	Profesor	Hora	Aula	Estudiante	Nota
C101	Sr. Sánchez	L9	8	Luís	5
C101	Sr. Sánchez	X9	6	Luís	5
C101	Sr. Sánchez	V9	7	Luís	5
C101	Sr. Sánchez	L9	8	Antonio	7
C101	Sr. Sánchez	X9	6	Antonio	7
C101	Sr. Sánchez	V9	7	Antonio	7

Se dan las dependencias multivaluadas $C \rightarrow\!\!\!\rightarrow HA$ y $HA \rightarrow\!\!\!\rightarrow H$

pero no se cumple $C \rightarrow\!\!\!\rightarrow H$

5. INTRODUCCIÓN A LAS BASES DE DATOS ACTIVAS

Contenidos

- Conceptos Básicos
- Reglas de Comportamiento
- Disparadores. Aplicaciones

Referencias

- **Principales:**
  - Elmarsi, R.; Navathe,S.B.; ***Sistemas de Bases de Datos: Conceptos fundamentales*** (3ª edición). Addison-Wesley, 2002.
  - De Miguel, A.; Piattini, M;

    Tecnología y Diseño de Bases de Datos

    Ra-Ma, 2006. Capítulos. 4, 5, 6, 7, 10 y **18**
  - Silberschatz, A.; Korth, H.F.; Sudarshan, S.; ***Fundamentos de Bases de Datos*** (4ª edición) McGraW-Hill, 2002.

5.1. Conceptos básicos

5.1. Conceptos Básicos

- Posibilidad de crear y ejecutar reglas de ejecución:

  Evento - Condición - Acción

- Reacciona de forma autónoma a sucesos realizados sobre la base de datos
- Parte del comportamiento del sistema se delega en la propia BD. Esto aumenta la eficiencia al evitar que los resultados vuelvan a los programas para que estos reaccionen contra la BD.

5.1. Conceptos Básicos

- Los SGBD activos se descomponen en:
  - El modelo de conocimiento: describe la situación y la reacción correspondiente (reglas ECA).
  - Modelo de ejecución: realiza un seguimiento de la situación y gestiona el comportamiento activo. Se encarga de indicar cómo se comportan las reglas en tiempo de ejecución.
- Más independencia de las aplicaciones al expresar parte de su semántica con reglas almacenadas (independencia de conocimiento)
- El conocimiento se codifica una sola vez y es compartido por todos los usuarios

5.2. Reglas de Comportamiento

Reglas activas

- Las reglas activas están basadas en el paradigma ECA

  EVENTO - CONDICIÓN - ACCIÓN

- Los eventos son primitivas de manipulación de datos en SQL

  INSERT, DELETE, UPDATE

- La condición es un predicado booleano sobre el estado de la base de datos, expresado en SQL
- Las acciones pueden ser ordenes arbitrarios de SQL

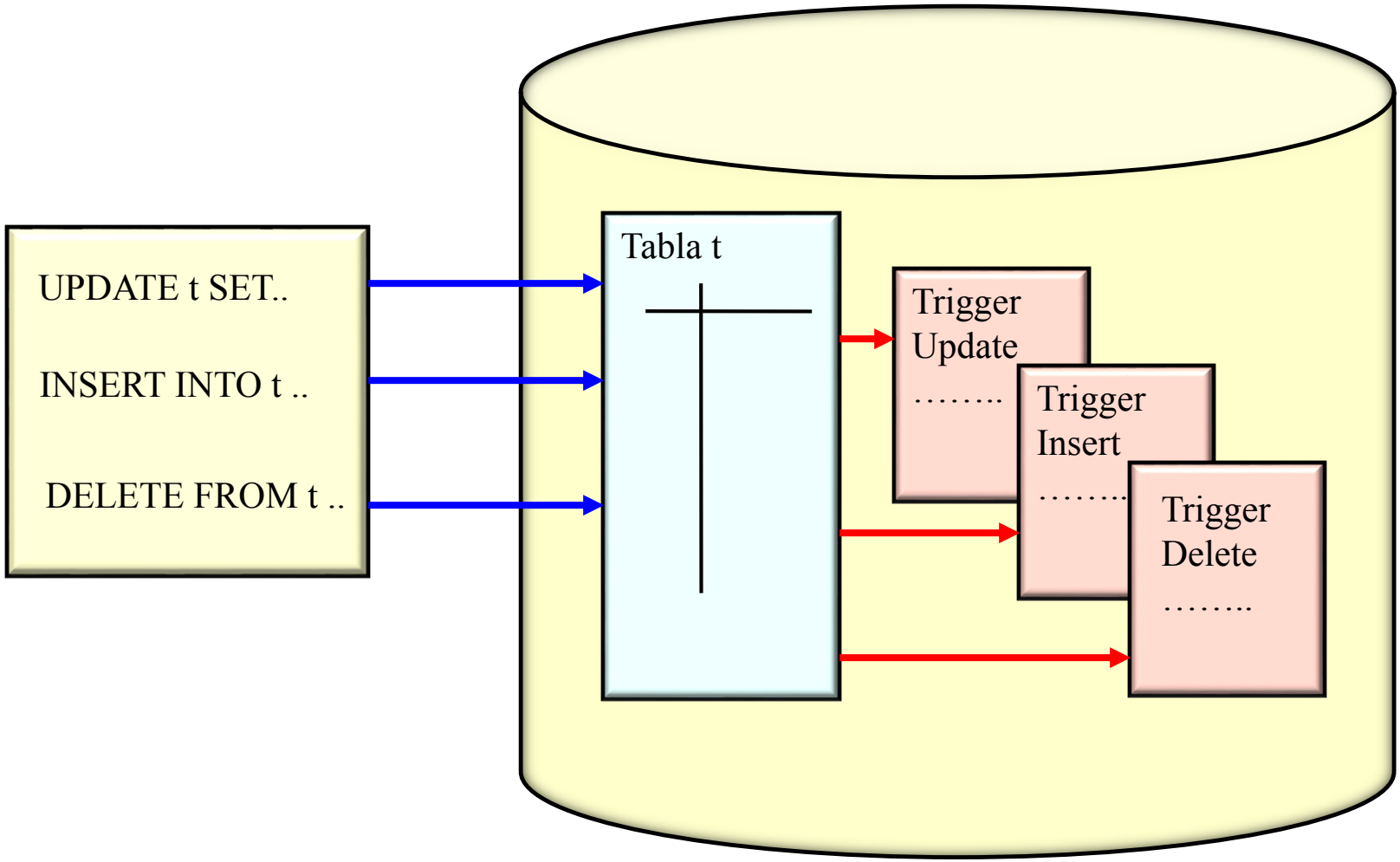
  SELECT, INSERT, DELETE, UPDATE, ROLLBACK…

Paradigma ECA

- Paradigma ECA
  - Cuando sucede el evento, si se cumple la condición, entonces se realiza la acción.
- Una regla es:
  - disparada cuando sucede el evento.
  - considerada cuando se evalúa la condición.
  - ejecutada cuando se realiza la acción.

- Las reglas activas se almacenan en el esquema y son compartidas por todas las aplicaciones

Esquema de Comportamiento



Conceptos

- **Evento/Suceso**:
  - Primitiva que modifica el estado de la base de datos
- **Condición**:
  - Predicado o consulta. Una consulta es interpretada TRUE si contiene tuplas, FALSE en caso contrario
- **Acción**:
  - Programa arbitrario de manipulación de datos (incluye órdenes transaccionales -ROLLBACK-)

Niveles de granularidad en detección de eventos

- **Nivel de Instancia**:
  - Los cambios afectan a filas individuales en tablas u objetos individuales en clases.

- **Nivel de orden**:
  - Los cambios de estado se detectan considerando las órdenes de manipulación de datos como eventos.
  - Es decir, la acción se ejecuta una vez terminada la orden SQL, con independencia del número de filas afectado.

Valores de transición

- Los valores de transición son datos temporales que describen los cambios de estado realizados por una transacción, y cuyo contenido se ve afectado por el nivel de granularidad de la regla:
  - Nivel de instancia: Los valores de transición son cambios que afectan a una tupla u objeto (variables correlación NEW y OLD)
  - Nivel de orden: Los valores de transición son valores colectivos almacenados en estructuras temporales.

Usos

- **Gestión clásica de BBDD**: (trigger para accesos no autorizados)
  - Frecuentemente generadas por el sistema y ocultas al Usuario.
  - Mantenimiento de la integridad.
  - Gestión de replicaciones.
  - Mantenimiento de datos derivados.
- **Reglas externas**:
  - Realiza parte de la computación normalmente incluida en las aplicaciones.
  - Ejemplos:
    - Reglas de gestión de inventario según las variaciones del stock.
    - Advertencias …..

5.3. Disparadores. Usos

Tipos de Disparadores

(gestión/modificación de datos)

- Trigger DML sobre **tablas**: Actúan cuando se efectúan operaciones del lenguaje de manipulación de datos (DELETE, INSERT, UPDATE) sobre una tabla.
- Trigger de Sistema sobre el **ESQUEMA**: Se disparan por cada evento específico del usuario (ALTER; CREATE, …)
- Trigger INSTEAD OF sobre **vistas**: Actúan en lugar de una operación realizada sobre las vistas. Permiten actualizar vistas complejas.
- Trigger de Sistema sobre la **BASE DE DATOS**: Se disparan a partir de eventos de la base de datos que afectan a todos los usuarios (SERVERERROR, LOGON, LOGOFF, STARTUP, SHUTDOWN, ...)

Tipos de Disparadores

Trigger Fila: Se dispara cada vez que la tabla se ve afectada por la orden que provoca el disparo. Se dispara por cada tupla actualizada (FOR EACH ROW).

Trigger Orden: Se dispara una única vez cuando se ejecuta la orden que provoca el disparo

Trigger BEFORE: Ejecuta el trigger inmediatamente antes que la orden que provoca el disparo.

Trigger AFTER: Ejecuta el trigger inmediatamente después que la orden que provoca el disparo

No se pueden utilizar sobre vistas o un objeto vistas no actualizables.

Combinaciones:

Trigger BEFORE a nivel de orden/Trigger AFTER a nivel de orden

Trigger BEFORE a nivel de fila / Trigger AFTER a nivel de fila

Eventos

Eventos DML:

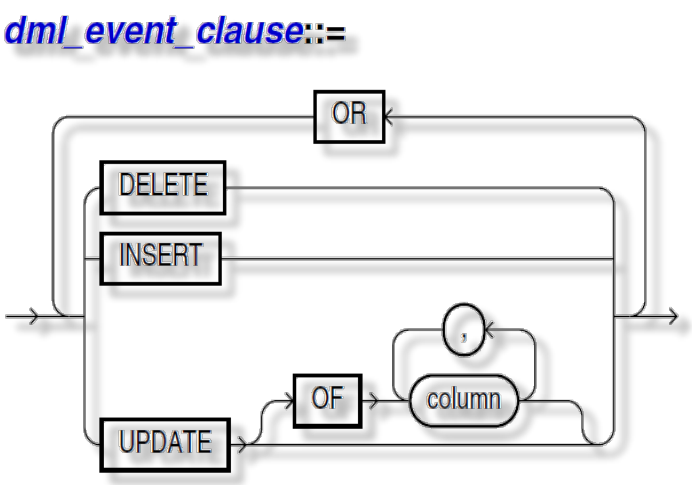
DELETE, INSERT, UPDATE [OF columna]:

El trigger se dispara cuando se produce un borrado, inserción o modificación de cualquier columna (si se omite OF) o las columnas especificadas en la cláusula OF.

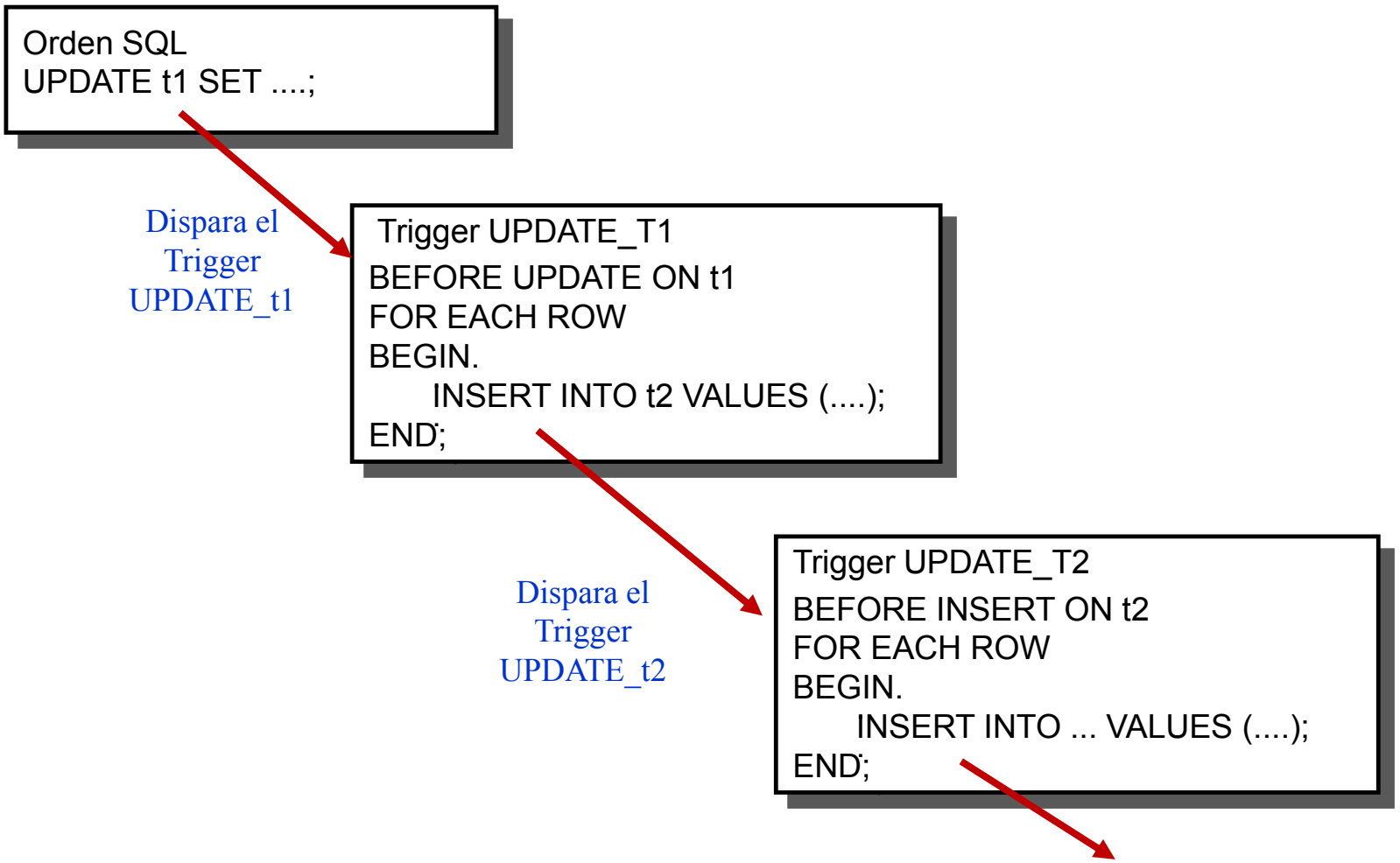
Eventos DDL:

CREATE, ALTER, DROP:

El trigger se dispara cuando se crean, modifican o borran objetos de la base de datos.



Ejecución en Cascada



5.3. Disparadores en Postgres

Postgres: CREATE TRIGGER -- define a new trigger

CREATE [CONSTRAINT] **TRIGGER** name { BEFORE | AFTER | INSTEAD OF }
{ **event** [OR ...] } ← **Event**
ON table_name
[**FROM** referenced_table_name]
[NOT DEFERRABLE | [DEFERRABLE] [INITIALLY IMMEDIATE | INITIALLY DEFERRED]]
[**FOR** [EACH] { ROW | STATEMENT }]
[**WHEN** (condition)] ← **Condition**
EXECUTE PROCEDURE function_name (arguments); ← **Action**

where event can be one of: INSERT, UPDATE [OF column_name [, ...]], DELETE or TRUNCATE

Postgres: CREATE TRIGGER -- Parameters

- name
  - This must be distinct from the name of any other trigger for the same table.
- BEFORE|AFTER|INSTEAD OF
  - Determines whether the function is called before, after, or instead of the event.
- table_name
  - The name (optionally schema-qualified) of the table, view, or foreign table the trigger is for.
  - like a normal table, but its data are not stored in the Postgres server.
- referenced_table_name (not recommended for general use).

Postgres: CREATE TRIGGER -- Parameters

- event
  - One of INSERT, UPDATE, DELETE, or TRUNCATE; this specifies the event that will fire the trigger. Multiple events can be specified using OR.
  - For UPDATE events, it is possible to specify a list of columns using this syntax:
    - UPDATE OF column_name1 [, column_name2 ...]
    - The trigger will only fire if at least one of the listed columns is mentioned as a target of the UPDATE command.
    - INSTEAD OF UPDATE events do not support lists of columns.

Postgres: CREATE TRIGGER -- Parameters

- FOR EACH ROW | FOR EACH STATEMENT (default)
  - This specifies whether the trigger procedure should be fired once for every row affected by the trigger event, or just once per SQL statement. (sin parámetros)
- function_name
  - A user-supplied function that is declared as taking no arguments and returning type trigger, which is executed when the trigger fires.

Postgres: CREATE TRIGGER -- Parameters

- condition
  - A Boolean expression that determines whether the trigger function will actually be executed.
  - If **WHEN** is specified, the function will only be called if the condition returns true.
  - In FOR EACH ROW triggers, the WHEN condition can refer to columns of the old and/or new row values by writing OLD.column_name or NEW.column_name respectively. INSERT triggers cannot refer to OLD and DELETE triggers cannot refer to NEW.
  - INSTEAD OF triggers do not support WHEN conditions.
  - Currently, WHEN expressions cannot contain subqueries.

Postgres: CREATE TRIGGER -- Parameters

- arguments
  - An optional comma-separated list of arguments to be provided to the function when the trigger is executed.
  - The arguments are literal string constants. Simple names and numeric constants can be written here, too, but they will all be converted to strings. **Please check the description of the implementation language of the trigger function to find out how these arguments can be accessed within the function; it might be different from normal function arguments** (**TG_ARGV**).

Todos los valores que pasamos se convierten a STRING

Postgres: CREATE TRIGGER -- Notes

- To create a trigger on a table, the user must have the TRIGGER and EXECUTE privilege on that table.
- Use DROP TRIGGER to remove a trigger.
- A column-specific trigger (one defined using the UPDATE OF column_name syntax)
  - will fire when any of its columns are listed as targets in the UPDATE command's SET list. nombre de columna o lista de nombres de columnas
  - It is possible for a column's value to change even when the trigger is not fired, because changes made to the row's contents by BEFORE UPDATE triggers are not considered.
  - Conversely, a command such as UPDATE ... SET x = x ... will fire a trigger on column x, even though the column's value did not change.

no comprueba la modificación, notifica el UPDATE: comprobar OLD ≠ NEW

Postgres: CREATE TRIGGER -- Notes

- In a BEFORE trigger,
  - the WHEN condition is evaluated just before the function is or would be executed, so using WHEN is not materially different from testing the same condition at the beginning of the trigger function. Note in particular that the NEW row seen by the condition is the current value, as possibly modified by earlier triggers.

Postgres: CREATE TRIGGER -- Notes

- In an AFTER trigger,
  - the WHEN condition is evaluated just after the row update occurs, and it determines whether an event is queued to fire the trigger at the end of statement. So when an AFTER trigger's WHEN condition does not return true, it is not necessary to queue an event nor to re-fetch the row at end of statement. This can result in significant speedups in statements that modify many rows, if the trigger only needs to be fired for a few of the rows.

Postgres: CREATE TRIGGER -- Example

- Execute the function check_account_update whenever a row of the table accounts is about to be updated:

```
CREATE TRIGGER check_update
   BEFORE UPDATE ON accounts
   FOR EACH ROW
   EXECUTE PROCEDURE check_account_update();
```

- The same, but only execute the function if **column** balance is specified as a target in the UPDATE command:

```
CREATE TRIGGER check_update
   BEFORE UPDATE OF balance ON accounts
   FOR EACH ROW
   EXECUTE PROCEDURE check_account_update();
```

Postgres: CREATE TRIGGER -- Example

- This form only executes the function if column balance has in fact changed value:

```
CREATE TRIGGER check_update
   BEFORE UPDATE ON accounts
   FOR EACH ROW
   WHEN (OLD.balance IS DISTINCT FROM NEW.balance)
   EXECUTE PROCEDURE check_account_update();
```

- Call a function to log updates of accounts, but only if something changed:

```
CREATE TRIGGER log_update
   AFTER UPDATE ON accounts
   FOR EACH ROW
   WHEN (OLD.* IS DISTINCT FROM NEW.*)
   EXECUTE PROCEDURE log_account_update();
```

Postgres: CREATE TRIGGER -- Example

- Execute the function view_insert_row for each row to insert rows into the tables underlying a view:

```
CREATE TRIGGER view_insert
   INSTEAD OF INSERT ON my_view
   FOR EACH ROW
   EXECUTE PROCEDURE view_insert_row();
```

Postgres: Trigger Procedures

- PL/pgSQL can be used to define trigger procedures on data changes (Triggers on Data Changes) or database events (Triggers on Events).
- A trigger procedure is created with:
  - the CREATE FUNCTION command,
  - declaring it as a function with **no arguments** even if it expects to receive some arguments specified in CREATE TRIGGER — such arguments are passed via **TG_ARGV[]**.
  - a return type of trigger (for data change triggers) or event_trigger (for database event triggers). Le devuelve el control al disparador que hizo la modif.
  - Special local variables named PG_something are automatically defined to describe the condition that triggered the call.

Postgres: CREATE TRIGGER -- Example

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp, -- who changed the payroll and when
    last_user text
);

CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$ ...

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```

eventos de manipulación de datos

Postgres: CREATE TRIGGER -- Example

```
CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
BEGIN
  -- Check that empname and salary are given
  IF NEW.empname IS NULL THEN
      RAISE EXCEPTION 'empname cannot be null'; END IF;
  IF NEW.salary IS NULL THEN
      RAISE EXCEPTION '% cannot have null salary', NEW.empname; END IF;
  -- Who works for us when they must pay for it?
  IF NEW.salary < 0 THEN
      RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
  END IF;
  -- Remember who changed the payroll and when
  NEW.last_date := current_timestamp;
  NEW.last_user := current_user;
  RETURN NEW;
END; $emp_stamp$ LANGUAGE plpgsql;
```