

Bases de Datos



isa salcedo

Tema 1: Introducción a los SBD

¿Qué es una base de datos?

Una base de datos (BD) es una colección organizada de información (de datos estructurados), que normalmente se almacena de forma electrónica en un sistema informático.

Un dato es un hecho conocido con significado implícito que puede ser registrado.

PROPIEDADES

- Representa aspectos del mundo real.
- Dirigida a un grupo de usuarios.
- Conseguir objetivos y aplicar.

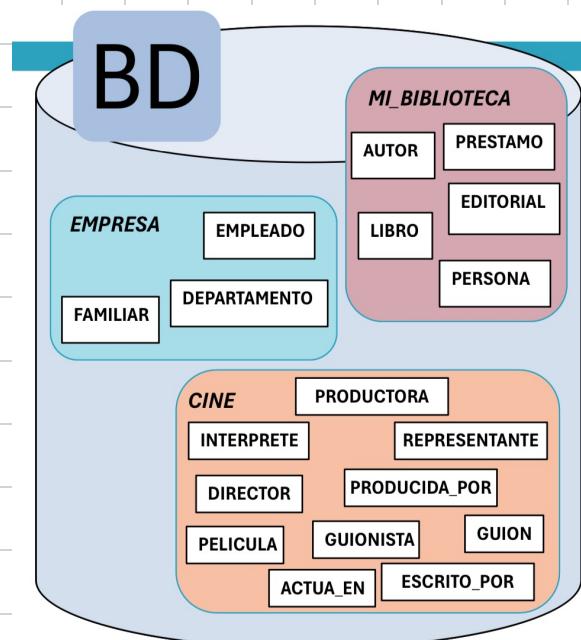
Primeras nociones

ESQUEMA

Un **esquema** es un conjunto de elementos de datos al que se le da un nombre.

Agrupa elementos de datos que pertenecen a un mismo contexto.

Una base de datos física puede contener varios esquemas.



SGBD

"Un SGBD es un conjunto de programas que permiten definir, crear, manipular y controlar el acceso a la base de datos."

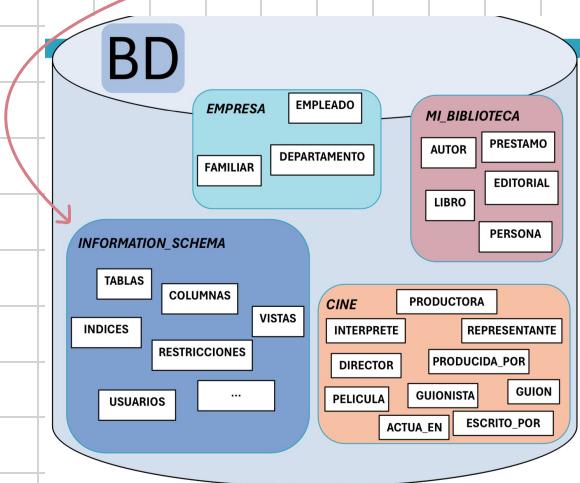
METADATOS

La BD almacena metadatos. Estos describen la estructura de la BD.

Incluyen descripciones de los datos y de las restricciones que los datos deben cumplir.

Proporciona a la base de datos una **naturaleza autodescriptiva**.

Están agrupados y almacenados en su propio esquema de la BD (Data Dictionary en Oracle). Es un esquema compuesto por tablas y vistas que contienen los metadatos en sí.



Tipos de SGBD

Según modelo de Datos:

- Relacional, Red, Jerárquico
- Orientado a Objetos
- Objeto / Relacional
- NoSQL: Documentos, Clave-Valor, Columnar, Grafos ...

Según el nº de usuario simultáneos

- Monousuario
- Multiusuario

Según su propósito

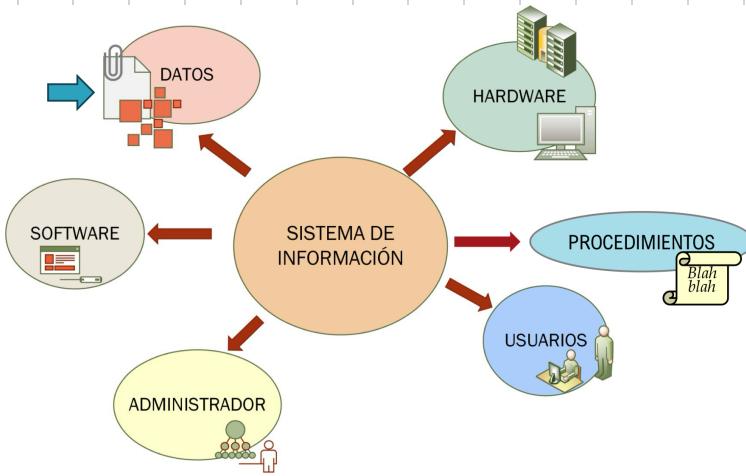
- propósito general
- propósito específico

Según el nº de lugares en que se almacenan datos

- Centralizado
- Distribuido

Tema 2: Modelos de datos

Introducción



Para construir un sistema de información es necesario realizar un análisis de la funcionalidad para hacer un diseño a través de **modelos** que facilitan su comprensión.

Se diseña el software (modelado de procesos) y la base de datos (modelado de datos).

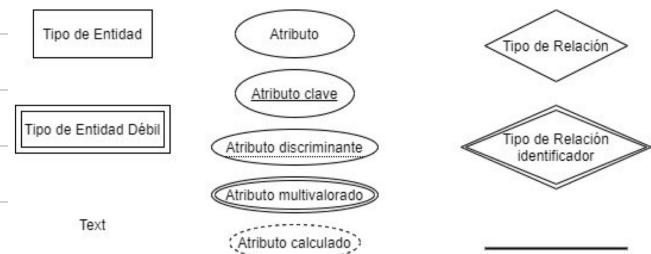
Los modelos de datos se usan para conseguir esa **visión abstracta**. Analogía: planos de casas o viviendas.

Son conjuntos de conceptos, reglas y convenciones que permiten describir y manipular datos.

Modelado de datos

SÍMBOLOGÍA

(Ejemplo: Modelo Entidad-Relación)



NIVELES DE ABSTRACCIÓN

- Nivel conceptual → cómo lo pensamos
- Nivel lógico → cómo lo vemos en el ordenador
- Nivel físico → de bajo nivel

Tema 3: El proceso de diseño de bases de datos

Etapas del diseño de bases de datos

1. Recopilación de requisitos de usuario

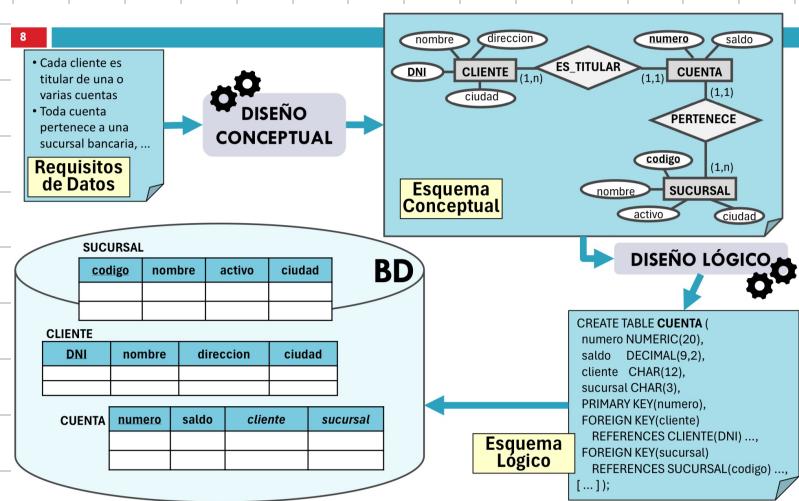
2. Diseño Conceptual

3. Elección del SGBD

4. Diseño lógico

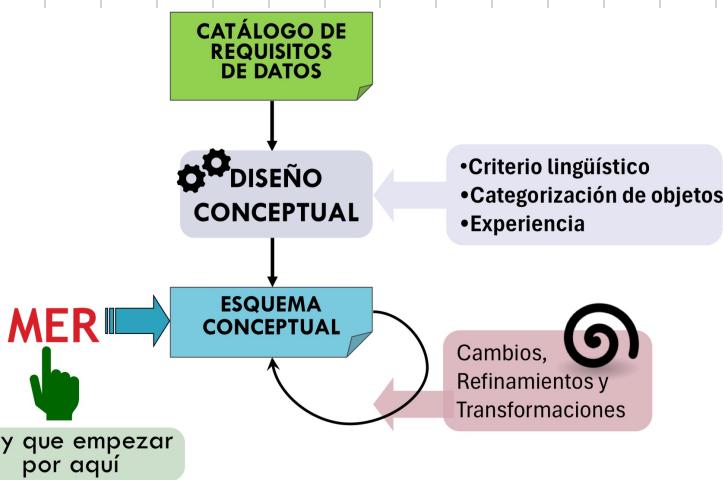
5. Diseño físico

6. Implementación



Tema 4: Diseño conceptual y MER

Modelo entidad-relación



Los tipos de entidad los podemos encontrar en forma de sujeto o complemento directo.

RELACIÓN

Asociación, vínculo o correspondencia entre instancias de tipos de entidad.

Notación

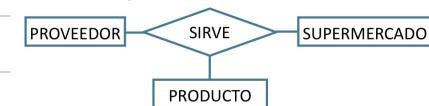


Hay tipos:

Binaria (la más frecuente)



Ternaria (grado 3)



Cuaternaria (grado 4)

no son nada frecuentes

Recursiva (grado 1)



Se encuentran en forma de verbos transitivos, frases verbales, preposiciones o frases preposicionales entre tipos de entidad.

Todo tipo de entidad que participa en un tipo de relación juega un papel específico: rol ↑

OBLIGATORIO INDICAR

ROLES EN RELACIONES

RECURSIVAS !!



Además, las relaciones tienen **cardinalidades**: describen las posibles combinaciones de entidades que pueden participar en las relaciones.

- "Una película está basada en un guion cinematográfico. Existen guiones que aún no hayan sido utilizados para realizar ninguna película. No hay dos películas con exactamente el mismo guion, ni siquiera los que son versiones o remakes de otras"



Si en los dos lados de la relación la cardinalidad máxima es n , en un lado cambiamos la letra ($a\ m$) !!

La cardinalidad mínima indica el tipo de participación:

- $0 \rightarrow$ opcional
- $1 \rightarrow$ obligatoria

Las cardinalidades máximas clasifican los tipos de relación:

1:1 (Uno a uno)



En el texto las encontramos en forma de singular y plural.

Motacón

(min, max)

min: 0 o 1

max: 1 o n (o m o p)

Una misma instancia del tipo de entidad DIRECTOR puede estar relacionada vía HA_RODADO con una o con muchas instancias del tipo de entidad PELICULA ($\text{mín}=1$ y $\text{máx}=n$); esto es lo mismo que preguntar ¿cuántas películas ha rodado un mismo director?

Una instancia concreta del tipo de entidad PELICULA está relacionada mediante HA_RODADO con sólo una instancia de DIRECTOR ($\text{mín}=1$ y $\text{máx}=1$); ¿Cuántos directores pueden rodar una misma película?



Una instancia del tipo de entidad DIRECTOR siempre ha rodado una 1ª película, y no es posible tener más de una película como 'opera prima'; ¿cuántas películas pueden ser la primera película rodada por un mismo director?

Una instancia determinada del tipo de entidad PELICULA puede no ser la OPERA_PRIMA de ningún director ($\text{mín}=0$), o sí ser la 1ª película que rodó un director ($\text{máx}=1$); ¿De cuántos directores puede ser 'opera prima' una misma película?

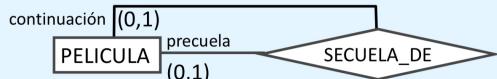
1:N (Uno a muchos)



M:N (Muchos a muchos)



Cardinalidades en tipos de relación recursivas:



Una peli puede ser secuela de 0 o 1 película, y puede ser precuela de 0 o 1 película.



Un empleado puede ser superior de 0 o n personas y puede tener 0 o 1 jefe.



ATRIBUTOS

Un atributo es una propiedad de un tipo de entidad o de relación. Se reconocen por ser un concepto al que se le asigna un valor y por no tener otras propiedades asociadas.

TIPOS

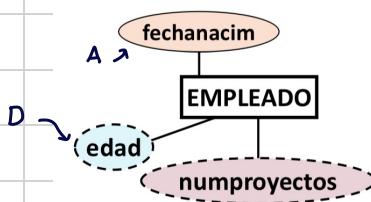
Simple / compuesto

Atómico / se divide en otros con significado propio

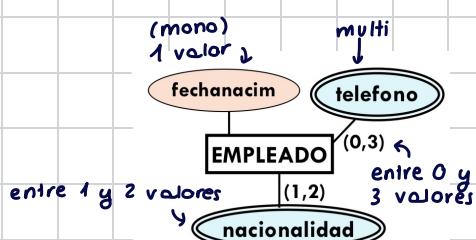


Almacenado / Derivado

Los derivados son valores calculados a partir de información ya existente.



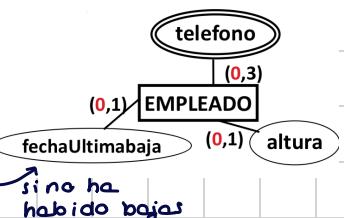
Monovalorado / multivalorado



Obligatorio / opcional

Los opcionales podrán contener **nulo**:

1. El valor existe pero no se conoce
2. No se sabe si el valor existe o no
3. No tiene ningún valor aplicable



El atributo **clave** es aquel que es distinto para cada instancia y que, por tanto, la identifica.

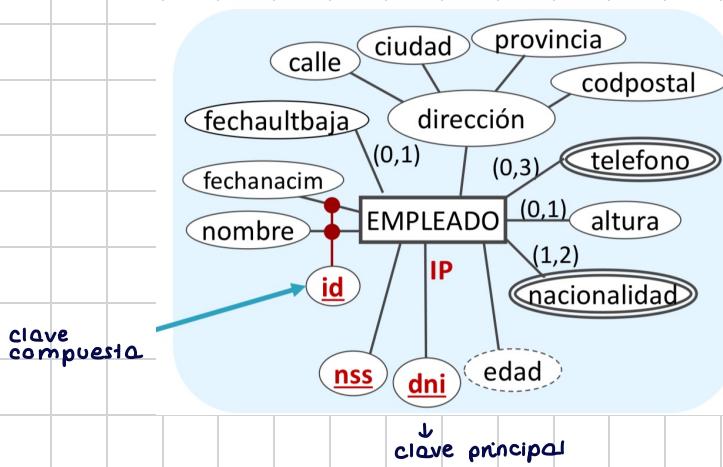
LIBRO

isbn

En realidad, una clave puede estar formada por varios atributos

(clave compuesta). Una clave compuesta debe ser mínima (el mínimo nº de atributos posible).

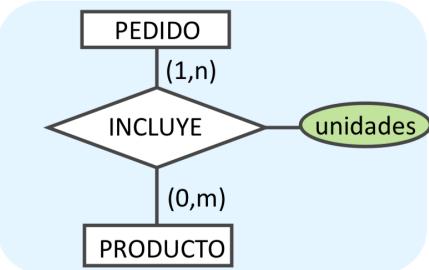
Un tipo de entidad puede tener más de una clave, pero hay que elegir una como **clave principal**, el resto serán las **claves alternativas**.



A veces, un tipo de relación puede contener atributos:

*“Un pedido puede incluir muchos productos, y cierto producto puede estar incluido en varios pedidos. Cada pedido indica cuántas **unidades** se desea comprar de cada producto.”*

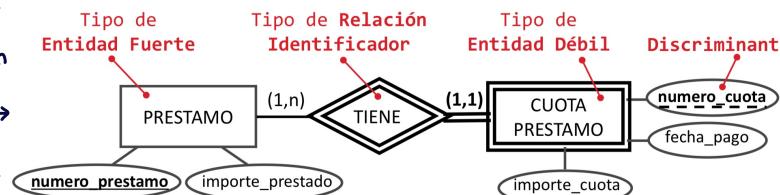
No se puede poner en pedido, porque en cada pedido a lo mejor queremos distintas unidades de cada producto y tampoco se puede poner en producto porque de un mismo producto distintas pedidas pedirán distintas unidades



TIPO DE ENTIDAD DÉBIL

Es débil si entre sus atributos no se puede encontrar una clave. Para distinguir entidades débiles entre sí necesitan de otro tipo de entidad (tipo de entidad fuerte).

Ambas entidades estarán relacionadas la notación es la siguiente



Lo que distingue a las instancias del tipo de entidad débil es la concatenación de la clave principal de la entidad fuerte y el **discriminante**.
↓
(numero-prestamo, numero-cuota)

La relación que los une es casi siempre de tipo 1:N (jamás M:N)!

JERARQUÍAS

La relación que se establece entre uno y otros corresponde a la noción de 'es-un' o de 'es-un-tipo-de'

Existe el **supertipo** (el que engloba a los demás) y el **subtipo** (el tipo específico).



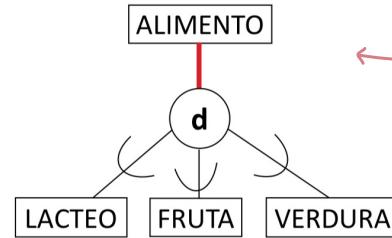
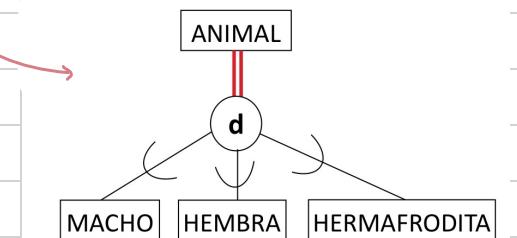
Disyunción / Solapamiento:

Los subtipos serán **disjuntos** si sólo se puede ser miembro de uno de ellos a la vez, o **solapados** si una instancia del supertipo puede pertenecer a más de un subtipo.

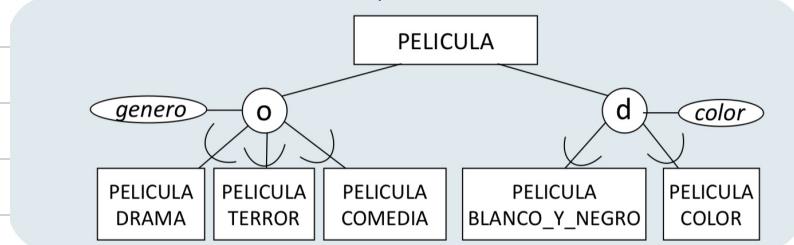
Si la jerarquía es disyunta se pone una "d" dentro del círculo y si es solapada se pone una "o".

Totalidad / Parcialidad:

Es **total** si toda instancia del supertipo pertenece a un subtipo obligatoriamente; y será **parcial** cuando puedan haber instancias del supertipo que no pertenezcan a ningún subtipo.



Es posible modelar varias especializaciones de un mismo tipo de entidad.



Un subtipo **hereda** todos los atributos del supertipo, y todo tipo de relación en la que **participa el supertipo**. !!

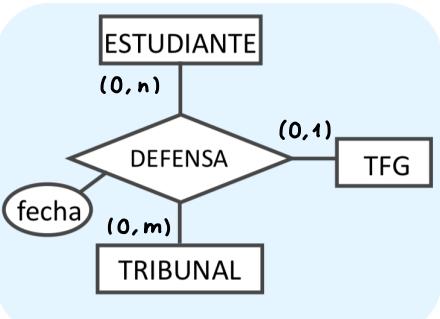
Los subtipos pueden tener sus propios atributos y relaciones.

¿Cómo sabemos que hay que modelar una jerarquía?

Si los subtipos tienen una estructura distinta entre sí: **atributos propios**, o **participan en distintas relaciones**.

RELACIONES N-ARIAS

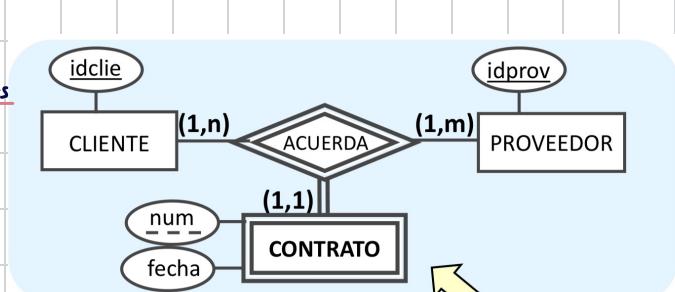
Vincula 3 o más tipos de entidad. La relación no existiría si faltara una de las instancias.



- Un estudiante en cierto momento o bien no ha defendido ningún TFG o ha defendido varios (ante el mismo o distinto tribunal). Cardinalidad **(0,n)**
- Un TFG o bien no ha sido defendido aún o sí lo ha sido. Un mismo TFG no puede ser defendido más de 1 vez. Cardinalidad **(0,1)**
- Un tribunal puede no haber presenciado aún la defensa de ningún TFG, o ha participado en muchas defensas de TFG (del mismo o distinto estudiante). Cardinalidad **(0,m)**

En ocasiones, nos encontraremos con relaciones ternarias 'falsa'.

Contrato es un tipo de entidad débil de varios tipos de entidad.



Pasos en el diseño conceptual

1. Determinar tipos de entidad.
2. Definir tipos de relación y cardinalidades.
3. Definir atributos y asociarlos a tipos de entidad.
4. Determinar los atributos clave (principales y alternativas) de los tipos de entidad.
5. Considerar incorporar restricciones de integridad.
6. Comprobar si existe redundancia.
7. Validar el esquema conceptual.
8. Revisar el esquema conceptual con el usuario.

Tema 5: Modelo relacional

Estructura de datos relacional

Ahora, las entidades se llaman **relaciones**. Se representan mediante una tabla. Cada fila (**tupla**) representa una instancia. Cada columna representa un **atributo**.

Este modelo está basado en la Teoría de Conjuntos y en la Lógica de predicados de primer orden.

DEFINICIONES FORMALES

Dominio: conjunto de valores atómicos del mismo tipo, donde toman su valor los atributos.

Relación: se compone de 2 partes:

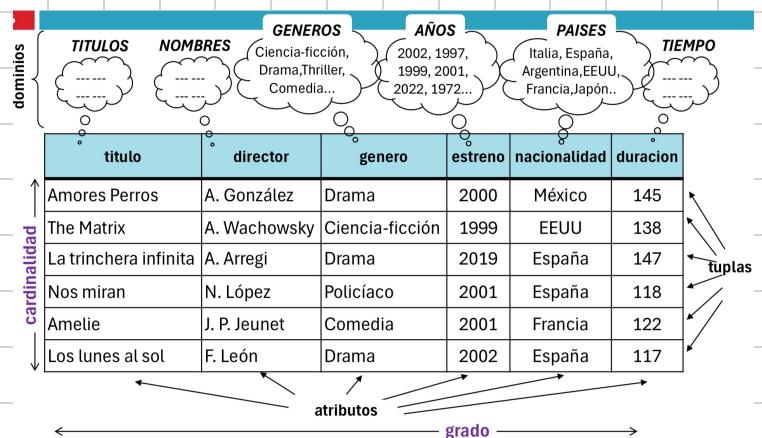
- **Esquema:** conjunto de pares (Atributo: Dominio) y restricciones de identidad.
- **Estado:** conjunto de pares (Atributo : Valor).

Propiedades:

- 1 No existen tuplas repetidas.
- 2 Las tuplas no están ordenadas.
- 3 Los atributos no están ordenados.
- 4 Los valores de atributos son atómicos.

Base de datos relacional: percibida por el usuario como una colección de relaciones.

Todo contenido de información está representado en la BD de una y sólo una forma: como un valor explícito dentro de una posición de columna dentro de una fila dentro de una tabla.



Características de integridad de datos

Todo estado de la BD refleja la realidad.

Las restricciones de integridad cumplen que:

- Forman parte de la BD.
- Se satisface para cualquier estado.
- No varía con el tiempo.

CLAVE DE UNA RELACIÓN

Una clave de R es un subconjunto de sus atributos tal que cumple la restricción de **Unicidad** (un valor por cada tupla), y la de **Irreductibilidad** (es mínima)

Recordemos que pueden ser simples o compuestas.

Además, una relación puede tener varias claves.

- Clave primaria (primary-key, PK): identifica las tuplas de R
- Claves alternativas (alternative-keys, AK):

NULL

Se usa para representar información perdida, ausencia de información y valor no aplicable a un atributo.

NULL no es un valor, y no es igual a otro NULL.

RESTRICCIÓN DE INTEGRIDAD DE ENTIDAD

Una clave primaria no puede contener NULL

Las claves alternativas si pueden contener NULL.

CLAVE AJENA

Es un atributo que se guarda en la tabla de una relación para vincularlo con otra.



CLIENTE

codigo	nombre	direccion	ciudad
1210	García, A.	Gran Vía, 6	Murcia
0300	López, B.	Ronda Norte, 3	Murcia
1003	Azorín, C.	Paseo Rosales, 9	Molina
2689	Pérez, D.	Plaza Mayor, 2	Patiño
...

CUENTA

numero	saldo	titular
200	85.005	2689
505	40.000	1003
821	50.000	1210
426	35.620	1003
005	29.872	2689
315	3.500	0300

titular solo toma valores

que estén en el atributo

código de CLIENTE

Una clave ajena SIEMPRE

hace referencia a una clave

PRIMARIA

Como una clave primaria puede ser compuesta, una clave ajena también puede ser compuesta.

RESERVA (cliente, hotel, habitacion, fecha_inicio, fecha_fin, ...)

Clave Primaria: (cliente, hotel, habitacion, fecha_inicio)

Clave Ajena: cliente referencia a CLIENTE (codigo)

Clave Ajena: (hotel, habitacion)

referencia a HABITACION_HOTEL (hotel, numero_habitacion)

ATENCIÓN !!

Como la clave primaria de HABITACIÓN-HOTEL es (hotel, numero_habitacion), la clave ajena en reserva que referencia a la habitación deberá contener ambos atributos.

Además, dicha clave ajena, forma parte de la clave primaria de RESERVA.

Nulos y claves ajenas

Una clave ajena puede contener NULL, significa que esa tupla no pertenece al vínculo.

EMPLEADO	NSS	nombre	dep	...
	123456789012	García, A.	D03	
	444444444444	Zapata, D.	D02	
	333333333333	Arjona, C.	NULL	
	222222222222	Sancho, B.	D03	
	556644332255	Gómez, H.	NULL	
	998877665544	Bolado, F.	D01	

dep hace referencia a DEPARTAMENTO (coddep)

Empleados sin departamento

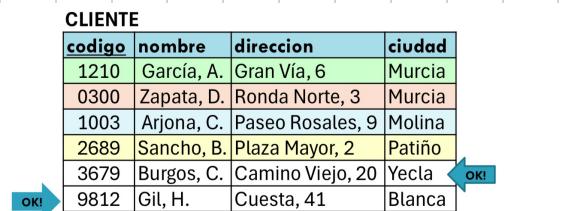
RESTRICCIÓN DE INTEGRIDAD REFERENCIAL

Las claves ajenas de una relación R1 hacen referencia a una relación R2:

- siendo su valor idéntico al de la PK de alguna tupla de R2.
- siendo su valor nulo

No habrá valores (no nulos) sin correspondencia.

Sí que puede existir un valor de clave primaria no referenciado.



Además, una clave ajena puede referenciar a su misma relación: autoreferencial.

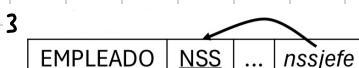
• Camino referencial (1)

• Ciclo referencial (2)

• Autoreferencia (3)



¿Cómo navegamos para encontrar cuál es la dirección de la editorial que ha publicado algún libro de la escritora cuyo nombre es 'Laura Gallego'?



Plantilla para describir una relación

TABLA (atributo1, atributo2, atributo3, atributo4, atributo5, atributo6, atributo7)

Admiten NULL: **atributo3, atributo6**

Clave Primaria: **atributo2**

Claves Alternativas (UNIQUE): 1. **atributo3** ; 2. **(atributo4, atributo5)**;

Claves Ajenas (FOREIGN KEY):

1. (**atributo7**) Referencia_a UNATABLA(clave)
 2. (**atributo4, atributo5**) Referencia_a OTRATABLA(clave1,clave2)

Der

Derivados:

- 1. atributo6 = atributo2*20/100**

•

- Comprobar:

1. **atributo1** IN ('SI', 'NO')
 2. **atributo6** > 0
 3. **atributo7** >= **atributo3**

• • •

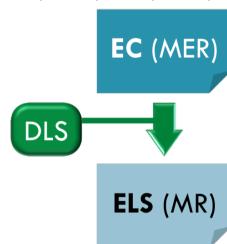
Tema 6: Diseño lógico

Fases del diseño lógico

DISEÑO LÓGICO ESTÁNDAR (DLS)

Se elige el modelo de datos lógico:

- Relacional ←
- Red
- jerárquico
- Orientado a objetos

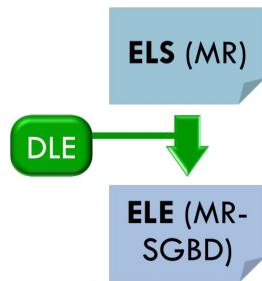


Se traduce el esquema conceptual y se obtiene el **Esquema Lógico estándar (ELS)**.

DISEÑO LÓGICO ESPECÍFICO (DLE)

Ya está elegido el SGBD. Se adapta el ELS al SGBD.

- Oracle ←
- MySQL
- MariaDB
- ...



Se obtiene el **Esquema lógico específico (ELE)**.

Dividir el diseño en DLS y DLE garantiza portabilidad, ya que podemos implementar el esquema lógico en distintos SGBD con el esquema estándar.

Diseño lógico estándar

PASOS

- 1 Obtener las relaciones (tablas).
- 2 Validarlas con las transacciones de usuario.
- 3 Revisar las restricciones de integridad.
- 4 Validar el esquema lógico con los usuarios.

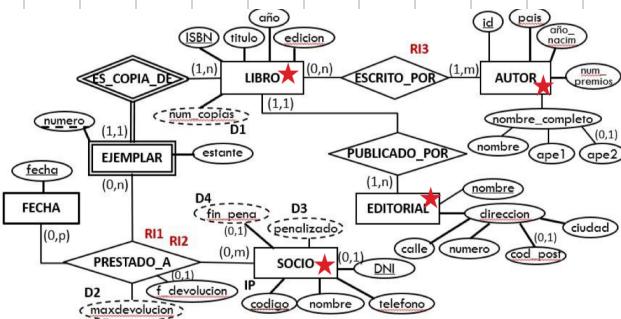
1 OBTENER RELACIONES (TABLAS)

1.1 TIPO DE ENTIDAD FUERTE

Se traduce a una relación (tabla). Se crea un atributo por cada atributo simple que tenga y uno por cada componente de atributo compuesto.

Se escribe la clave primaria y sus claves alternativas

Se indican los atributos que admiten nulo, restricciones, si son calculados...



EDITORIAL(nombre, calle, numero, cod_post, ciudad)
Admiten NULL: **cod_post**

Clave primaria: **nombre**

SOCIO(codigo, DNI, nombre, telefono, penalizado, fin_pena)

Admiten NULL: **DNI, fin_pena**

Clave primaria: **codigo**

Clave alternativa: **DNI**

Derivados: **penalizado, fin_pena**

(nota: faltan fórmulas de cálculo)

AUTOR(id, nombre, pais, año_nacim, num_premios, nombre, ape1, ape2)

Admiten NULL: **ape2**

Clave primaria: **id**

LIBRO(ISBN, título, año, edición, num_copias)

Admiten NULL: Ninguno

Clave primaria: **ISBN**

Derivado: **num_copias** (nota: falta fórmula de cálculo)

1.2 TIPO DE ENTIDAD DÉBIL

Se traduce a una relación (tabla). La clave primaria no se define hasta traducir el tipo de relación con su fuerte.

EJEMPLAR(numero, estante)

Admiten NULL: Ninguno

Clave primaria: <**Pendiente**>

(del esquema anterior)

1.3 TRADUCCIÓN DE JERARQUÍAS

Disjunta y total

Da lugar a una tabla por cada combinación supertipo/subtipo (tantas como subtipos).

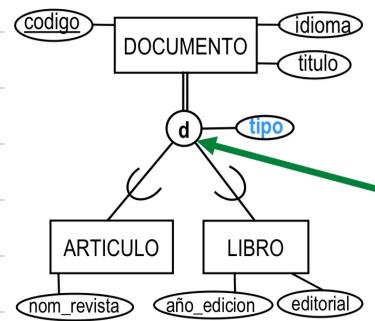
- Cada relación tendrá los atributos del supertipo y de su subtipo.
- Hay que definir un **aserto** para asegurar la disyunción.

ARTICULO (codigo, titulo, idioma, nom_revista)

Clave primaria: **codigo**

LIBRO (codigo, titulo, idioma, año_edicion, editorial)

Clave primaria: **codigo**



Se debe asegurar que ninguna instancia de tipo ARTICULO puede ser a la vez de LIBRO.

ASERTOS

Un aserto es una restricción de integridad que afecta a más de una relación y expresa una condición que sus tuplas deben cumplir.

Se redactan expresando lo que NO se debe cumplir.

- Comprobar que no existe una tupla de ARTICULO tal que el valor de "codigo" esté entre los valores de "codigo" de las tuplas de LIBRO

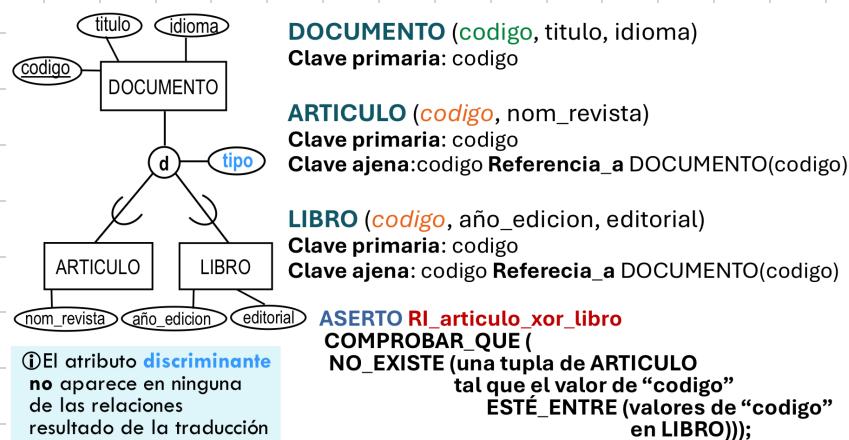
ASERTO RI_articulo_xor_libro

```
COMPROBAR_QUE(  
    NO_EXISTE (una tupla de ARTICULO  
        tal que el valor de "codigo"  
        ESTÉ_ENTRE (valores de "codigo"  
            de LIBRO)));
```

Disjunta y parcial

Una relación para el supertipo y una para cada subtipo.

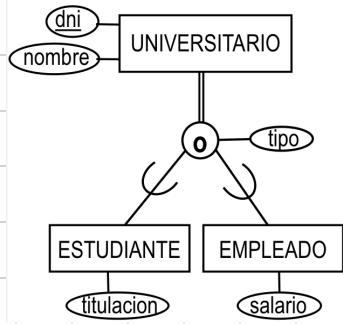
- Las relaciones de los subtipos contienen una clave ajena que referencia a la clave primaria del supertipo.
- La clave primaria de cada relación de subtipo es **dicha clave ajena**.
- Se incluye un aserto para la disyunción.



Solapada y total

Da lugar a una única relación, que corresponde al supertipo.

- Debe incluir uno o más atributos **discriminantes** para distinguir el tipo concreto de la tupla.
- Los atributos del subtipo deben admitir nulo.
- Hay que definir **restricciones de comprobación**.



UNIVERSITARIO(dni, nombre, **tipo**, titulacion, salario)

Admiten NULL: titulacion, salario

Clave primaria: dni

Comprobar:

- tipo IN ('Estudiante', 'Empleado', 'Est+Emp')
- ((tipo = 'Estudiante' AND titulacion IS NOT NULL)
OR (tipo = 'Empleado' AND salario IS NOT NULL)
OR (tipo = 'Est+Emp' AND titulacion IS NOT NULL
AND salario IS NOT NULL))

Solapada y parcial

Da lugar a 2 relaciones: una para el supertipo y una para todas los subtipos.

La relación que agrupa subtipos:

- Incluye una clave ajena que referencia al supertipo.
- La clave primaria es dicha clave ajena.
- Incluye discriminantes para distinguir el tipo.
- Los atributos de los subtipos deben admitir nulo.
- Restricciones de comprobación.

INDIVIDUO(dni, nombre, fechanac)

Clave primaria: dni

INDIVIDUO_ACTIVO(dni, **actividad**, titulacion, nss, sueldo)

Admiten NULL: titulacion, nss, sueldo

Clave primaria: **dni**

Clave alternativa: **nss**

Clave ajena: dni Referencia_a INDIVIDUO(dni)

Comprobar:

- actividad IN ('Estudia', 'Trabaja', 'Est+Trab')
- ((actividad='Estudia' AND titulacion IS NOT NULL)
OR (actividad='Trabaja' AND nss IS NOT NULL
AND sueldo IS NOT NULL)
OR (actividad='Est+Trab'
AND titulacion IS NOT NULL
AND nss IS NOT NULL AND sueldo IS NOT NULL))

① Uso de un **atributo discriminante**, con un valor para cada tipo y un valor extra para el solapamiento



1.4 TIPO DE RELACIÓN 1:N

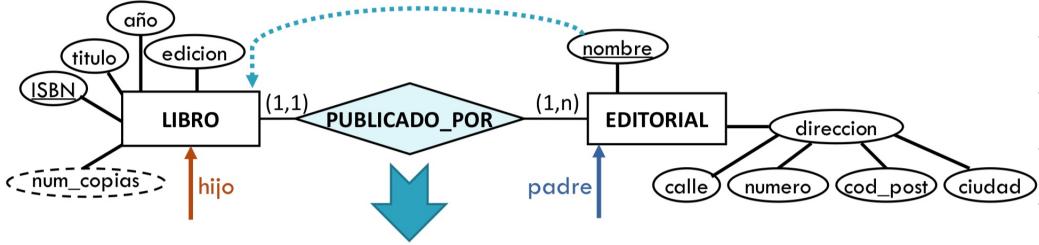
Se traduce a una clave ajena que referencia desde una relación a otra.

El tipo de entidad 'padre': participa muchos veces.

El tipo de entidad 'hijo': participa sólo una vez.



Hay que añadir al 'hijo' la clave ajena.



EDITORIAL(nombre, calle, numero, cod_post, ciudad)

Admiten NULL: Ninguno

Clave primaria: nombre

LIBRO(ISBN, título, año, edición, num_copias, **editorial)**

Admiten NULL: Ninguno

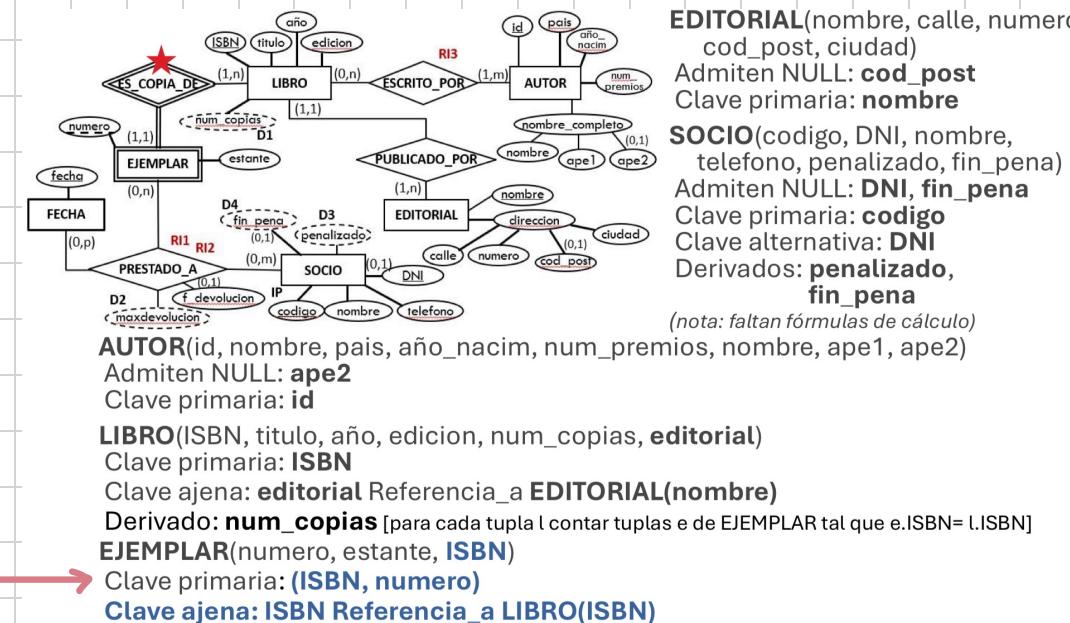
Clave primaria: ISBN

Clave ajena: editorial Referencia_a EDITORIAL(nombre)

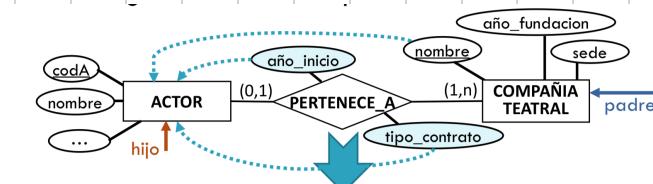
Derivado: num_copias (nota:falta fórmula de cálculo)

OJO!!

Si el tipo 'hijo' es débil, la clave ajena pasa a formar parte de la clave primaria.



Si el tipo de relación tiene atributos, se añaden en el tipo de relación 'hijo'. Persiguen a la clave ajena.

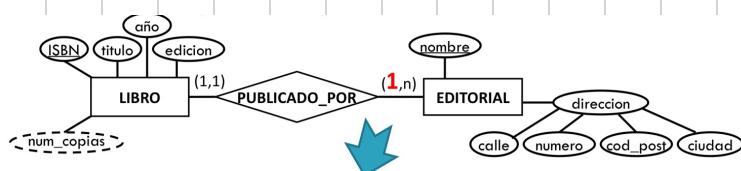


COMPAÑIA(nombre, año_fundacion, sede)
Clave primaria: nombre

ACTOR(codA, nombre, ..., **compañia, año_inicio, tipo_contrato)**
Clave primaria: codA
Clave ajena: compañía Referencia_a COMPAÑIA(nombre)

Cardinalidad

- Cardinalidad mínima 0 del 'hijo': la clave ajena admite nulo, además de los atributos de la relación (si hay).
- Cardinalidad mínima 1 del 'hijo': la clave ajena NO admite nulo.
- Cardinalidad mínima 0 del 'padre': indica que pueden haber tuplas del 'padre' que no están referenciadas en el hijo.
- Cardinalidad mínima 1 del 'padre': debe representarse mediante un **aserto**.



Esa cardinalidad mínima representa que

"Toda EDITORIAL ha publicado al menos un LIBRO".

► Hay que asegurar que **no existe una EDITORIAL que no esté vinculada con al menos un LIBRO vía PUBLICADO_POR**.



La redacción más adecuada es esta:

► **ASERTO RI_editorial_publica_libros**
COMPROBAR_QUE (NO_EXISTE (una tupla en EDITORIAL
donde el valor de "nombre" NO ESTÉ ENTRE
(valores de "editorial" en LIBRO));



Recursiva

Deducir los roles. La relación (ya definida) representa al 'hijo'. Se añade la clave ajena que toma el nombre del rol del 'padre'.



EMPLEADO(codE, nombre, dni, dirección,..., **jefe)**

Admiten NULL: Ninguno

Clave primaria: codE

Clave alternativa: dni

Clave ajena: jefe Referencia_a EMPLEADO(codE)

Comprobar: codE <-> jefe

1.5 TIPO DE RELACIÓN 1:1

Se traduce a una clave ajena.

a) Participación obligatoria en un lado

Padre → parte opcional, hijo → parte obligatoria

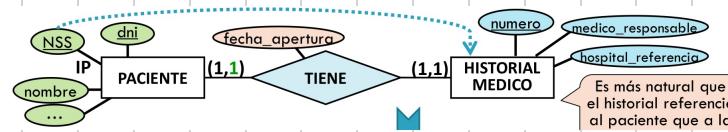
Se pone la clave ajena en el hijo.



La cardinalidad **máxima 1** del **padre** hace que la clave ajena sea también clave alternativa. !! (pasa en b) y c) también)

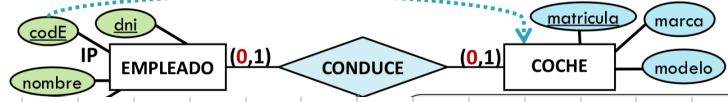
b) Participación obligatoria en ambos lados.

Se puede poner la clave ajena en cualquiera de los dos, pero normalmente hay una que tiene más sentido que la otra.



c) Participación opcional en ambos lados.

Elegir con sentido común a cuál ponerle la clave ajena (igual que en el anterior).



1.6 TIPO DE RELACIÓN 1:1 RECURSIVA

a) Obligatoria a ambos lados

Añadimos la clave ajena (copia de su clave primaria) a EMPLEADO y

para nombrarla usar los roles. La clave ajena será también clave alternativa. !!



b) Opcional en ambos lados

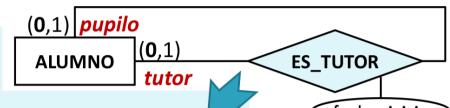
Se crea nueva relación con 2

copias de la clave primaria (una

será la nueva primaria y otra

alternativa).

ALUMNO(dni, num_expediente, nombre, ...)
Admiten NULL: Ninguno
Clave primaria: dni
Clave alternativa: num_expediente
ES_TUTOR(tutor, pupilo, fecha_inicio)
Admiten NULL: Ninguno
Clave primaria: **tutor**; Clave alternativa: **pupilo**
Clave ajena: tutor Referencia_a ALUMNO(dni)
Clave ajena: pupilo Referencia_a ALUMNO(dni)
Comprobar: **tutor <-> pupilo**



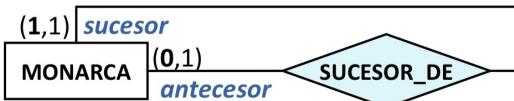
① En la nueva relación, una de las claves ajena será la **clave primaria**, y la otra será **clave alternativa**

Añadimos: un alumno no se puede tutorizar a sí mismo

c) Obligatoria en un lado.

Se traduce a una clave ajena. Padre \leftarrow opcional, hijo \leftarrow obligatoria

La relación existente corresponde al hijo.

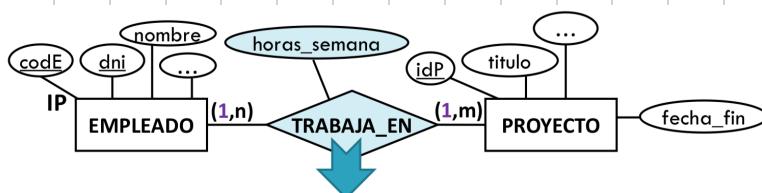


La clave ajena será 'antecesor'.

1.7 TIPO DE RELACIÓN M:N

Crear una nueva relación.

- Añadir una copia de las claves primarias de los tipos de entidad (claves ajenas).
- Incluir atributos para los atributos del tipo de relación (si los hay).
- La clave primaria será la concatenación de las claves ajenas.
- Añadir un **aserto** por cada cardinalidad mínima 1.



TRABAJA_EN(*empleado, proyecto, horas_semana*)

Admiten NULL: Ninguno

Clave primaria: (*empleado, proyecto*)

Clave ajena: empleado **Referencia_a** EMPLEADO(*codE*)

Clave ajena: proyecto **Referencia_a** PROYECTO(*idP*)

ASERTO RI_empleado_trabaja_en_proyectos COMPROBAR_QUE
(NO_EXISTE (una tupla en EMPLEADO

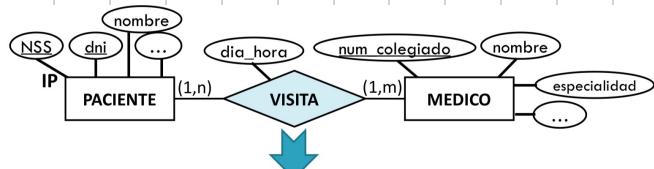
donde el valor de "codE" NO_ESTÉ_ENTRE
(valores de "empleado" en TRABAJA_EN));

ASERTO RI_proyecto_tiene_empleados COMPROBAR_QUE
(NO_EXISTE (una tupla en PROYECTO

donde el valor de "idP" NO_ESTÉ_ENTRE
(valores de "proyecto" en TRABAJA_EN));

Las cardinalidades mínimas 1 de los tipos de entidad conectados por la M:N hacen necesaria la definición de 2 asertos

A veces ocurre que la concatenación de las claves ajenas no es suficiente como clave primaria y se deben añadir más atributos:



VISITA(*paciente, doctor, dia_hora*)

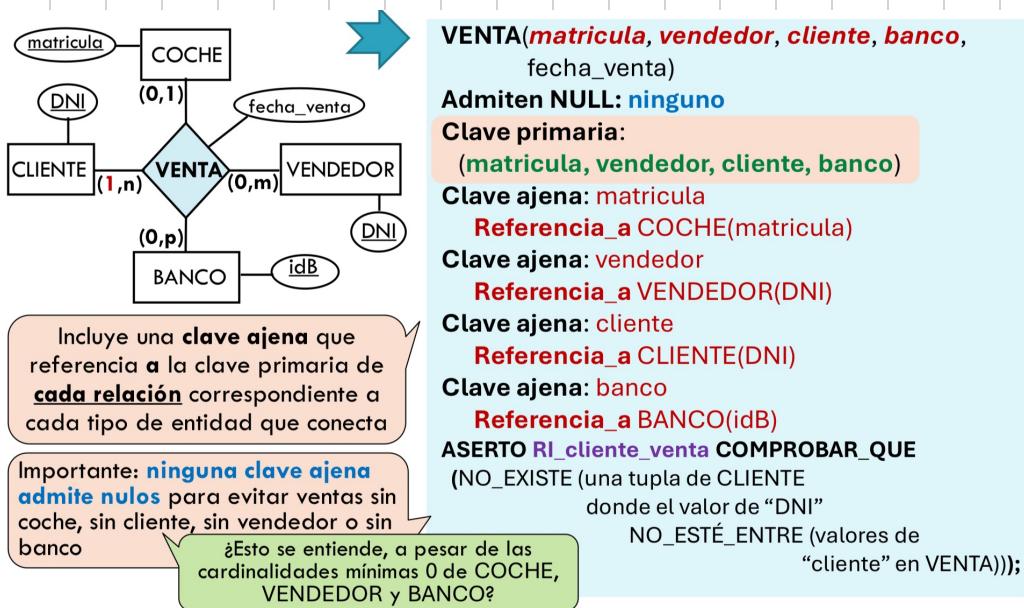
Clave primaria: (*paciente, doctor, dia_hora*) Ahora Sí

(paciente, doctor) no es suficiente para identificar una visita, le añadimos el atributo dia_hora para ser precisos.

1.8 TIPO DE RELACIÓN M-ARIA

Crear una nueva relación:

- Añadimos los atributos del tipo de relación.
- Añadimos una copia de las claves primarias de los tipos de entidad conectados (claves ajenas).
- Añadir un **aserto** por cada cardinalidad mínima 1.
- La clave **primaria** será la concatenación de algunas o todas las claves ajenas.



A veces la clave primaria puede **reducirse**, en este caso "matrícula" podría ser la clave primaria porque un coche participará sólo una vez en una venta (cardinalidad máxima 1).

1.9 ATRIBUTO MULTIVALORADO

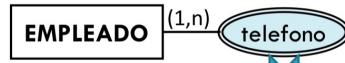
Crear una nueva relación. Incluir un atributo con el mismo nombre que el atributo multivalorado.

Añadimos una clave ajena, que será la clave primaria del tipo de entidad al que está conectado.

La clave primaria será:

- La concatenación de la clave ajena y el atributo.
- O solo el atributo.

Añadimos un **aserto** si la cardinalidad mín del atributo es 1, o si la cardinalidad máxima es conocida (no es n).



TELEFONO_EMPLEADO(empleado, telefono)

Admiten **NULL**: ninguno

Clave primaria: **(empleado, telefono)**

Clave ajena: empleado Referencia_a EMPLEADO(codE)

ASERTO RI_telefono_empleado COMPROBAR_QUE

(NO_EXISTE (una tupla de EMPLEADO
donde el valor de "codE"
NO ESTÉ ENTRE (valores de
"empleado" en TELEFONO_EMPLEADO)))

Reajuste de traducción (93-96)

VER !!

Ver reajuste
de
traducción

Yo

REVISAR RESTRICCIONES

1. Datos requeridos
2. Restricciones de dominio de datos
3. Integridad de entidad
4. Integridad referencial (claves ajena)
5. Restricciones generales (asertos)

Diseño lógico específico

Elegimos ahora el SGBD comercial y lo implementamos. En el tema siguiente se estudia el estándar SQL. Utilizaremos sentencias CREATE TABLE para definir/crear las tablas del esquema.

Tema 7: Definición de datos

Un lenguaje de BD completo se compone de comandos organizados en:

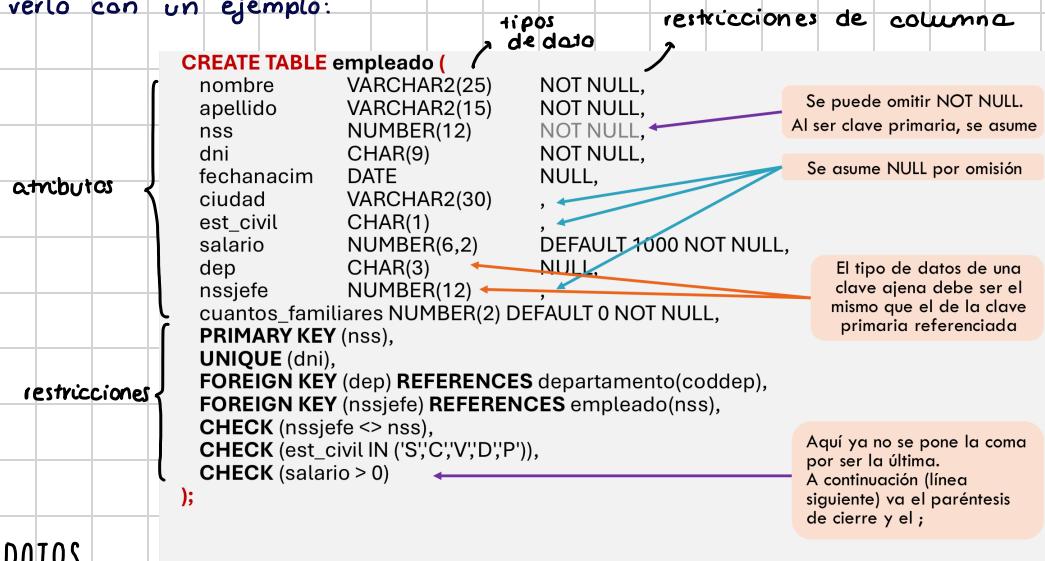
- **LDD**: Lenguaje de definición de datos: ordenar para crear, modificar o eliminar tablas, vistas...
CREATE, ALTER, DROP,..
 - **LMD**: Lenguaje de manipulación de datos: introducir datos, actualizarlos, eliminarlos...
INSERT, UPDATE, DELETE y SELECT

Sentencia CREATE TABLE

Sirve para crear tablas (relación en el Modelo-Relacional). El nombre de la tabla será único y dentro habrá que indicar:

- **Columnas (atributos)**: con su nombre, tipo de datos y restricciones
 - **Restricciones de tabla**: clave primaria, ajena, restricciones...

Vamos a verlo con un ejemplo:



TIPOS DE DATOS

- ### • Cadenas de caracteres:

`CHAR[(size)]` -- por defecto size = 1

VARCHAR2(size)

- ## • Números:

NUMBER[(p[,s])] -- por defecto p:=38 u s:=C

- Fecha •

DATE

RESTRICCIONES DE COLUMNAS

- Cláusula **MNULL o NOT NULL**: por omisión se asume **NULL**, excepto para las columnas componentes de una clave primaria, que nunca podrán aceptar **NULL**.
- Cláusula **DEFAULT valor**: debe ir dentro del tipo de datos y delante de cualquier otra restricción.

RESTRICCIONES DE TABLA

- Cláusula **PRIMARY KEY (lista柱as)**: indica que columnas forman parte de la clave primaria, asegura que no puedan tener el valor **NULL**.
- Cláusula **UNIQUE KEY (lista柱as)**: indica que columnas forman parte de la clave alternativa. Puede haber muchas y puede ser compuesta.
- Cláusula **CHECK(expresión)**: para especificar restricciones adicionales, que deben cumplir las columnas indicadas. Debe ser una expresión booleana. sólo se pueden comparar columnas de la tabla o columnas y constantes.
- Cláusula **FOREIGN KEY (lista柱as)**: define que columnas forman una clave ajena. Se indica la tabla y la clave a la que referencia. Debe coincidir el tipo de datos y ser compuesta en caso de que la clave a la que refiera lo sea. !!

Ejemplo de clave ajena compuesta:

```
CREATE TABLE hotel(  
codigo CHAR(4) NOT NULL,  
nombre VARCHAR2(30) NOT NULL,  
... -- otras columnas  
CONSTRAINT hotel_pk PRIMARY KEY(codigo)  
);  
  
CREATE TABLE salon_hotel(  
id_salon CHAR(2) NOT NULL,  
hotel CHAR(4) NOT NULL,  
capacidad NUMBER(3) NOT NULL,  
... -- otras columnas  
CONSTRAINT salon_pk PRIMARY KEY (hotel, id_salon),  
CONSTRAINT salon_fk_hotel FOREIGN KEY(hotel) REFERENCES hotel(codigo)  
...  
);
```

Cada salón referencia al hotel en el cual está ubicado (clave ajena)

```
CREATE TABLE reserva_salon(  
numero NUMBER(3) NOT NULL,  
hotel CHAR(4) NOT NULL,  
salon CHAR(2) NOT NULL,  
fecha DATE NOT NULL,  
... -- otras columnas  
CONSTRAINT reserva_pk PRIMARY KEY(numero),  
CONSTRAINT res_fk_salon FOREIGN KEY(hotel,salon)  
REFERENCES salon_hotel(hotel,id_salon)
```

Una clave ajena debe tener tantas columnas como tiene la clave primaria a la que referencia

La clave primaria está compuesta por dos columnas

...

Hay operaciones que pueden romper la integridad Referencial, por ejemplo, cambiar una clave primaria que está siendo referenciada en otra tabla, o borrar una fila que está siendo referenciada.

CLIENTE			
codigo	nombre	direccion	ciudad
1210	Garcia, A.	Gran Vía, 6	Murcia
0300	Zapata, D.	Ronda Norte, 3	Murcia
1003	Ajona, C.	Paseo Rosales, 9	Molina
2689	Sancho, B.	Plaza Mayor, 2	Patiño
3679	Burgos, C.	Camino Viejo, 20	Yecla

CUENTA		
numero	saldo	titular
200	85.005	2689
505	40.000	1003
821	50.000	1210
426	35.620	1003
005	29.872	2689
315	3.500	0300



* Si se borra la fila que contenga la clave primaria a la que referencia esa clave ajena

¿Cómo evitamos que rompa la Integridad Referencial?

Debemos definir las acciones de mantenimiento, es decir, como actuará si se borra la FOREIGN KEY (ON DELETE) o si se actualiza (ON UPDATE).

Acciones que se pueden realizar:

ON DELETE

- NO ACTION: rechaza la acción
- CASCADE: propaga la eliminación
- SET NULL: establecer a nulo (si es posible)
- DEFAULT: valor por defecto (si existe)

ON UPDATE

- NO ACTION: rechaza la acción
- CASCADE: propaga la modificación
- SET NULL: establecer a nulo (si es posible)
- DEFAULT: valor por defecto (si existe)

Según el tipo de situación y usando el sentido común, elegimos la acción que se ajuste mejor (y sea posible).

CREATE TABLE empleado (

```
    ...
    FOREIGN KEY (dep) REFERENCES departamento(coddep)
        ON DELETE NO ACTION —> no se permite borrar un departamento
        ON UPDATE CASCADE, —> se actualiza en cascada
    FOREIGN KEY (nssjefe) REFERENCES empleado(nss)
        ON DELETE SET NULL —> si se borra un empleado jefe a NULL
        ON UPDATE CASCADE, —> si se cambia se actualiza
    ...);
```

ANSI SQL

!!

En ORACLE sólo se permiten 3 acciones de ON DELETE (NO ACTION, CASCADE y SET NULL) y NO EXISTE la cláusula ON UPDATE (se asume NO ACTION).

En las prácticas las escribiremos en forma de comentarios.

NOMBRES DE RESTRICCIÓN

Conviene dar nombre a las restricciones como las PRIMARY KEY, UNIQUE, CHECK...

Para ello se antepone a dichas cláusulas:

CONSTRAINT nombre_R1

El nombre debe ser único y podemos seguir un patrón:

abreviaturaTabla_TipoRestricción

```

CREATE TABLE empleado (
    nombre          VARCHAR2(25)  NOT NULL,
    apellido        VARCHAR2(15)  NOT NULL,
    nss            NUMBER(12)   NOT NULL,
    dni             CHAR(9)      NOT NULL,
    fechanacim     DATE         NULL,
    ciudad          VARCHAR2(30)  ,
    est_civil       CHAR(1)      ,
    salario         NUMBER(6,2)  DEFAULT 1000 NOT NULL,
    dep             CHAR(3)      NULL,
    nssjefe         NUMBER(12)  ,
    cuantos_familiares NUMBER(2) DEFAULT 0 NOT NULL,
    CONSTRAINT emp_pk PRIMARY KEY (nss),
    CONSTRAINT emp_ak UNIQUE (dni),
    CONSTRAINT emp_fk_dep FOREIGN KEY (dep)
        REFERENCES departamento(coddep),
        -- ON DELETE NO ACTION ON UPDATE CASCADE
    CONSTRAINT emp_fk_emp FOREIGN KEY (nssjefe) REFERENCES empleado(nss),
        -- ON DELETE NO ACTION ON UPDATE CASCADE
    CONSTRAINT emp_jefe_ok CHECK (nssjefe <> nss),
    CONSTRAINT emp_ec_ok CHECK (est_civil IN ('S','C','V','D','P')),
    CONSTRAINT emp_sal_ok CHECK (salario > 0)
);

```



Sentencia ALTER TABLE

Permite modificar la estructura de una tabla: añadir columnas, eliminarlas, modificarlas, añadir restricciones, eliminarlas, modificarlas.

- Añadir columna:

```

ALTER TABLE empleado
ADD (fecha_contrato DATE);

```

se puede añadir la cláusula NULL (o NOT) y
DEFAULT

- Renombrar columnas.

```

ALTER TABLE empleado
RENAME COLUMN fecha_contrato TO inicio_contrato;

```

- Renombrar tabla:

```

ALTER TABLE empleado RENAME TO emp;

```

para renombrar un elemento:

RENAME _ TO _

- Modificar columna:

```

ALTER TABLE empleado
MODIFY apellido VARCHAR2(30);
--antes tamaño 15

```

se puede modificar el tipo,
el DEFAULT, el NULL, el UNIQUE...

- Eliminar columna / columnas:

```

ALTER TABLE empleado
DROP COLUMN fechanacim;

```

/ ALTER TABLE empleado
 DROP (fechanacim, est_civil);

Si una columna está referenciada no se permitirá su eliminación. Se podrá forzar su eliminación con **CASCADE CONSTRAINTS**.

```
ALTER TABLE departamento
DROP COLUMN coddep CASCADE CONSTRAINTS;
```

La columna "dep" que hace referencia a "coddep", ya no sería clave ajena, si no un atributo normal.

- Añadir una restricción:

```
ALTER TABLE tabla ADD
CONSTRAINT nombre_RI definición_RI;
```

ya sea CHECK, clave primaria, ajena...

CICLO REFERENCIAL

EMPLEADO.dep → DEPARTAMENTO(coddep)
DEPARTAMENTO.nssdire → EMPLEADO(nss)

EMPLEADO					DEPARTAMENTO		
nombre	apellido	nss	...	dep	coddep	nombre	nssdire
JONÁS	SOLANO	123		D1	D2	INVESTIGACION	222
RIGOBERTA	CALAVERA	321		D3	D1	ADMINISTRACION	111
EUSEBIO	MULETAS	222		D2	D3	PERSONAL	333
MACARENO	SOSO	111		D1	D4	TRAINING	NULL
CASIANA	FABERGÉ	333		D3			
FILOMENA	RASCAS	234	34E	D1			
GUMERSINDA	MIMOS	543	45F	NULL			

Si dos tablas se refieren entre sí y queremos crearlas, primero creamos una sin definir la restricción clave ajena, luego la otra, y luego añadimos la restricción con **ALTER TABLE**

```
ALTER TABLE empleado ADD CONSTRAINT emp_fk_dep
FOREIGN KEY dep REFERENCES departamento(coddep);
-- ON DELETE NO ACTION ON UPDATE CASCADE
```

- Desactivar una restricción:

```
ALTER TABLE tabla
DISABLE CONSTRAINT nombre_RI;
```

- Activar una restricción

```
ALTER TABLE tabla
ENABLE CONSTRAINT nombre_RI;
```

ELIMINAR UNA RESTRICCIÓN

Para eliminar cualquier restricción usamos:

```
ALTER TABLE tabla  
DROP CONSTRAINT nombre_RI [CASCADE];
```

Si la restricción es clave primaria podemos usar **DROP PRIMARY KEY [CASCADE]** en vez de la segunda línea. (o **DROP UNIQUE()** para claves alternativas)

Eliminar una clave falla si existe alguna clave ajena que la refiere, ahí entra en juego la opción **CASCADE**.

IMPORTANTE!! Eliminar la restricción clave primaria no es eliminar su columna, si no que ahora permite valores repetidos.

Sentencia **DROP TABLE**

Destrucción de una tabla (estructura y contenido). Si se omite **CASCADE CONSTRAINTS** solo destruye la tabla si no se le hace referencia desde otra.

```
DROP TABLE table  
[CASCADE CONSTRAINTS] [PURGE];
```

Con **PURGE**, se elimina la tabla y se libera su espacio en un solo paso.

En el powerPoint del tema hay ejemplos de esquema lógico estándar para practicar, además de su implementación en SQL

Tema 8: manipulación de datos

Consulta

Permite extraer información sobre las tablas almacenadas.

Esto lo hacemos en Oracle mediante la sentencia SELECT:

```
SELECT lista_columnas  
FROM lista_tablas  
WHERE condición ; ← siempre termina con un punto y coma
```

Las condiciones pueden ser varias, unidas con AND, OR o incluso pueden ser negativas con NOT.

Para eliminar filas duplicadas se usa la cláusula DISTINCT, que va justo después de SELECT.

Además, para seleccionar todos los columnas de la tabla o tablas del FROM, podemos, o bien escribir el nombre de todas ellas, o usar SELECT *.

CADERNAS DE CARACTERES

- Operador LIKE: sirve para comparar cadenas. Sigue usar comodines:

- % cantidad cualquiera de caracteres
- _ un solo carácter

* Nombres y apellidos de empleados de Las Torres de Cotillas o Cabezo de Torres
SELECT nombre, apellido
FROM empleado
WHERE ciudad LIKE '%TORRES%';

* Nombres completos, en una sola columna, de los empleados de las ciudades de Aledo, Alguazas, Alhama y Alcantarilla
SELECT nombre || ' ' || apellido
FROM empleado
WHERE ciudad LIKE 'AL%';

- Operador || concatenación de cadenas

ARITMÉTICA

Se pueden aplicar operaciones (+, *, -, /) sobre las columnas para mostrarlo en el resultado del SELECT.

* ¿Cómo quedarían los salarios de los empleados del departamento D3 tras un aumento del 10%?
SELECT apellido, nombre, 1.1*salario
FROM empleado
WHERE dep = 'D3';

apellido	nombre	1.1*salario
CALAVERA	RIGOBERTA	990
FABERGÉ	CASIANA	1012

El valor de "salario" en empleado no cambiara!!

FECHAS

- TO_CHAR(fecha, formato): convierte una fecha en una cadena.
- TO_DATE(cadena, formato): convierte una cadena en una fecha.

Formatos (ejemplo):

- 'dd/mm/yy'
- 'dd-Mon-yyyy'
- 'yyyy'
- 'dd/month hh24:mi:ss'

* Año de nacimiento y ciudad del empleado con apellido 'RASCAS'

```
SELECT TO_CHAR(fechanacim, 'yyyy'), ciudad  
FROM empleado  
WHERE apellido = 'RASCAS';
```

TO_CHAR(fechnacim, 'yyyy')	ciudad
1970	MURCIA

* Nombres y apellidos de los empleados nacidos antes de 1972

```
SELECT nombre, apellido  
FROM empleado  
WHERE fechanacim < TO_DATE('01/01/1972', 'dd/mm/yyyy');
```

- EXTRACT(campo FROM fecha): campo puede ser YEAR, MONTH, DAY, HOUR, MINUTE, SECOND

CONJUNTOS DE VALORES

Se puede usar IN o

Nss de lxs empleadxs que tienen padres/madres o abuelxs a su cargo

NOT IN.

```
SELECT nssemp  
FROM familiar  
WHERE parentesco IN ('MADRE', 'PADRE', 'ABUELO', 'ABUELA');
```

NULL

Es una marca que indica desconocimiento y ausencia de información. No se puede comparar.
null es distinto de cualquier otra cosa, incluso de otro null.

Para comprobar si una columna tiene el valor null, se necesita un comparador específico:

- IS NULL, IS NOT NULL

```
SELECT nombre, apellido  
FROM empleado  
WHERE nssjefe IS NULL;
```

ORDEN DE PRESENTACIÓN

Permite ordenar las filas con la cláusula ORDER BY. de manera ascendente ASC (por defecto) o descendente DESC.

```
SELECT apellido, nombre, fechanacim  
FROM empleado  
WHERE ciudad = 'MURCIA'  
ORDER BY apellido ASC, nombre DESC;
```

REUNIÓN

¿Qué ocurre si queremos mostrar datos almacenados en varias tablas?

SELECT *
FROM empleado JOIN departamento
ON dep = coddep;

Unimos con JOIN y escribimos el vínculo entre las 2 tablas con ON.
↓
clave ajena y clave primaria

EMPLEADO				
nombre	apellido	nss	...	dep
JONÁS	SOLANO	123	...	D1
RIGOBERTA	CALAVERA	321	...	D3
EUSEBIO	MULETAS	222	...	D2
MACARENO	SOSO	111	...	D1
CASIANA	FABERGÉ	333	...	D3
FILOMENA	RASCAS	234	...	D1
GUMERSINDA	MIMOS	543	...	NULL

DEPARTAMENTO			
nombre	coddep	nssdire	
INVESTIGACION	D2	222	
ADMINISTRACION	D1	111	
PERSONAL	D3	333	
TRAINING	D4	NULL	

SELECT *
FROM empleado JOIN departamento
ON dep=coddep;

nombre	apellido	nss	...	dep	nombre	coddep	nssdire
JONÁS	SOLANO	123	...	D1	ADMINISTRACION	D1	111
RIGOBERTA	CALAVERA	321	...	D3	PERSONAL	D3	333
EUSEBIO	MULETAS	222	...	D2	INVESTIGACION	D2	222
MACARENO	SOSO	111	...	D1	ADMINISTRACION	D1	111
CASIANA	FABERGÉ	333	...	D3	PERSONAL	D3	333
FILOMENA	RASCAS	234	...	D1	ADMINISTRACION	D1	111

Resultado del JOIN: una tabla con una fila por cada dos filas vinculadas de las tablas origen

ni el departamento D4 ni Gumersinda aparecen, porque Gumersinda tiene dep a NULL, y el D4 no está referenciado.

La condición de reunión puede incluir más de 1 comparación por igualdad, ligadas con un AND. Esto ocurre cuando la clave ajena y la clave primaria son compuestas.

SELECT ISBN, numero, estante, fecha
FROM prestamo JOIN ejemplar → reunión de dos tablas
ON libro = ISBN → condición de reunión
AND ejemplar = numero con 2 comparaciones
WHERE socio = 'S03'; → condición de selección de filas

Clave primaria EJEMPLAR: (ISBN, numero)

↓ ↓
Clave ajena PRÉSTAMO: (libro, ejemplar)

CALIFICACIÓN

Distintas tablas pueden tener columnas que se llamen igual. Esto puede causar ambigüedad en un SELECT si las seleccionamos.

La mejor opción es usar **pseudónimos** para diferenciar las tablas:

SELECT e.nombre, apellido, d.nombre
FROM empleado e JOIN departamento d
ON dep = coddep;

Solo hace falta calificar las columnas ambiguas, pero se puede hacer con todas.

RENOMBRAR COLUMNAS

SELECT e.nombre **empleado**, j.nombre **jefe**
FROM empleado e JOIN empleado j
ON e.nssjefe = j.nss;

Resultado SIN los nuevos nombres

e.nombre	j.nombre
JONÁS	MACARENO
RIGOBERTA	CASIANA
EUSEBIO	JONÁS
CASIANA	JONÁS
FILOMENA	MACARENO

Resultado CON los nuevos nombres

empleado	jefe
JONÁS	MACARENO
RIGOBERTA	CASIANA
EUSEBIO	JONÁS
CASIANA	JONÁS
FILOMENA	MACARENO

Simplemente añadimos el nuevo nombre separado por un espacio de la columna.

REUNIÓN PARTE II:

Se pueden usar varios JOIN ON en una SELECT para extraer información de varias tablas

* Para cada familiar 'ABUELA' o 'MADRE', mostrar tal parentesco, el apellido del empleado del que depende, y el nss del director del departamento al que pertenece el/la empleado.

SELECT parentesco, apellido, nssdire
FROM familiar JOIN empleado
ON nssempp = nss
JOIN departamento
ON dep = coddep
WHERE parentesco IN ('ABUELA', 'MADRE');

El 1º JOIN obtiene una fila con los datos de cada familiar y los de su empleado

El 2º JOIN obtiene una fila para cada familiar con todos sus datos, los de su empleado y los del departamento al que pertenece el empleado

Antes, las tablas del JOIN se escribían en el FROM separadas por comas, y la condición de reunión iba en el WHERE.

notación clásica ↴

```
SELECT e.nombre, apellido, ciudad
FROM empleado e, departamento d
WHERE dep = coddep
AND d.nombre = 'INVESTIGACION';
```

← reunión de tablas
← condición de reunión
← condición de selección

Si se hace un JOIN de varias tablas y se omite el WHERE, se combina cada fila de una tabla con las filas de la otra: **producto cartesiano**

REUNIÓN NATURAL

El SGBD asume que la condición de reunión es las columnas que se llamen igual.

```
SELECT ...
FROM R1 NATURAL JOIN R2
WHERE ...
```

REUNIÓN EXTERNA

Necesaria si se necesita obtener filas con NULL en la condición de reunión o no referenciadas.

LEFT [OUTER] JOIN	Reunión externa izquierda
RIGHT [OUTER] JOIN	Reunión externa derecha
FULL [OUTER] JOIN	Reunión externa total o completa

Ejemplo de reunión externa con la tabla departamento y empleado.

```
SELECT *
FROM (empleado e FULL JOIN departamento d
      ON dep = coddep);
```

e.nombre	apellido	nss	...	dep	d.nombre	coddep	nssdire
JONÁS	SOLANO	123	...	D1	ADMINISTRACION	D1	111
RIGOBERTA	CALAVERA	321	...	D3	PERSONAL	D3	333
EUSEBIO	MULETAS	222	...	D2	INVESTIGACION	D2	222
MACARENO	SOSO	111	...	D1	ADMINISTRACION	D1	111
CASIANA	FABERGÉ	333	...	D3	PERSONAL	D3	333
FILOMENA	RASCAS	234	...	D1	ADMINISTRACION	D1	111
GUMERSINDA	MIMOS	543	...	NULL	NULL	NULL	NULL
NULL	NULL	NULL	...	NULL	TRAINING	D4	NULL

A veces, queremos que se muestre algo distinto de NULL en las columnas con dicho valor.

Para ello, usamos la función COALESCE(columna, valor).

```
SELECT nombre, apellido, COALESCE(dep, 'SIN DEPARTAMENTO')
FROM empleado;
```

La columna y el valor deben de ser del mismo tipo. Si no, podemos llevar a cabo transformaciones.

```
SELECT nss, nombre, COALESCE(TO_CHAR(nssjefe), 'SIN JEFE')
FROM empleado;
```

↳ nssjefe es de tipo NUMBER

TABLAS COMO CONJUNTOS

Se les pueden aplicar operaciones de conjuntos a las tablas: UNION, INTERSECTION y MINUS.

- UNION: filas de ambas tablas (sin duplicados).
- INTERSECT: filas que estén en ambas tablas.

- MINUS: filas en la primera tabla que no estén en la segunda.

CONSULTAS ANIDADAS

Es una consulta `SELECT` completa, dentro de la cláusula `WHERE` de otra consulta.

Los valores que selecciona se usan en la condición de la consulta externa.

- * Familiares de empleadxs que son directores de departamento

```
SELECT nombre FROM familiar
WHERE nssemp IN (SELECT nssdire
                  FROM departamento);
```

Primeramente se evalúa la subconsulta, luego se resuelve el `WHERE`, y por último se resuelve el `SELECT`.

Es posible tener varios niveles de consultas anidadas.

```
SELECT dni, nombre
      FROM empleado
     WHERE dep IN (SELECT coddep
                     FROM departamento
                    WHERE nssdire IN (SELECT nss
                                      FROM empleado
                                     WHERE nombre = 'MACARENO'
                                         AND apellido = 'SOSO'));
```

Si la consulta anidada devuelve una sola columna y fila y es un valor escalar, se pueden usar los operadores (`=, <, >, <=, >=, <>`) en vez de `IN`.

Estos operadores se pueden concatenar con `ANY`, `SOME` o `ALL`.

* Nombres y apellidos de lxs empleadxs cuyo salario es menor que el de todxs los empleadxs del departamento D3

```
SELECT nombre, apellido
      FROM empleado
     WHERE salario < ALL (SELECT salario
                           FROM empleado
                          WHERE dep = 'D3');
```

CORRELACIÓN

A veces una consulta anidada necesita un valor que está en un `FROM` exterior.

- * Código de los departamentos cuyo director pertenece a otro departamento distinto (detección de errores en los datos introducidos)

```
SELECT coddep
      FROM departamento
     WHERE nssdire IN ( SELECT nss
                           FROM empleado
                          WHERE dep <> coddep);
```

Se dice que las consultas están correlacionadas.

OJO! Es una operación muy costosa.

Operador `EXISTS`: sirve para comprobar si una tabla es vacía (devuelve `TRUE` o `FALSE`).

Se usa sobre todo en sentido negativo (con `NOT`).

* Nombres y apellidos de los empleados sin familiares

```
SELECT nombre, apellido
      FROM empleado
     WHERE NOT EXISTS (SELECT *
                           FROM familiar
                          WHERE nssemp=nss);
```

FUNCIONES DE AGREGADOS

Es muy habitual necesitar hacer cálculos con los datos de una tabla.

Para ello podemos usar funciones como: `SUM()`, `MIN()`, `MAX()`, `AVG()`. Dentro se escribe el nombre de la columna con la que queremos operar.

Función `COUNT(*)` cuenta el nº de filas o, si en vez de * escribimos una columna, cuenta valores no nulos de ella.

Si queremos que cuente valores distintos: `COUNT(DISTINCT columna)`.

¿Cuántos empleados son jefes de algún otro?

```
SELECT COUNT(DISTINCT nssjefe)
FROM empleado;
```

* Apellidos, nombres y salarios de empleados cuyo sueldo coincide con el sueldo medio
SELECT nombre, apellido, salario
FROM empleado
WHERE salario = (SELECT AVG(salario)
FROM empleado);

apellido	nombre
vacío	

Las funciones de agregados...

- Se pueden anidar: `MAX(COUNT(*))`
- Admiten el DISTINCT (no sólo el COUNT)
- `SUM`, `MAX`, `MIN`, `AVG` ignoran el NULL, pero lo pueden devolver si no hay filas.
- `COUNT` no ignora NULL, pero siempre devolverá un nº o un 0.

AGRUPACIÓN

A veces necesitamos agrupar valores para realizar algunos cálculos.

Esto se hace con la cláusula `GROUP BY`, que selecciona una columna por la que se agruparán los valores en la tabla.

↓
columna de agrupación

```
SELECT dep, COUNT(*), AVG(salario)
FROM empleado
GROUP BY dep;
```

↓
en el SELECT

dep	COUNT(*)	AVG(*)
D1	3	1100
D2	1	2100
D3	2	910
NULL	1	850

solo podrá aparecer la columna de agrupación

y funciones de agregados

nombre	apellido	salario	...	dep
JONÁS	SOLANO	1100	...	D1
MACARENO	SOSO	1100	...	D1
FILOMENA	RASCAS	1100	...	D1
EUSEBIO	MULETAS	2100	...	D2
RIGOBERTA	CALAVERA	900	...	D3
CASIANA	FABERGÉ	920	...	D3
GUMERSINDA	MIMOS	850	...	NULL

Grupo 1: D1

Grupo 2: D2

Grupo 3: D3

Grupo 4: NULL

Usamos la cláusula `HAVING` para poner una condición a los grupos de filas. Es como un `WHERE` para grupos.

```
SELECT nombre, nssdire
FROM departamento
WHERE coddep IN (SELECT dep
                  FROM empleado
                  GROUP BY dep
                  HAVING AVG(salario) < 1200);
```

ONLINE VIEW

Es una consulta dentro de la cláusula FROM de otra SELECT. No es una anidada.

* Para cada empleado, mostrar su nombre, salario y cuántos familiares tiene.
Columnas: (nombre, salario, cuantos_familiares)

```
SELECT nombre, salario, n_familiares  
FROM empleado e  
JOIN (SELECT nssemp, COUNT(*) n_familiares  
      FROM familiar  
      GROUP BY nssemp) f  
ON e.nss = f.nssemp;
```

2. El JOIN obtiene una fila para cada empleado, junto con el nssemp y su número de familiares

1. La online view obtiene el número de familiares que tiene cada empleado

NOMBRE	SALARIO	N_FAMILIARES
MACARENO	1100	2
JONAS	1100	1
EUSEBIO	2100	1
RIGOBERTA	900	3

Se pueden hacer tantas Online Views como se quieran, siempre añadiendo luego la condición de reunión en el ON.

(en el power point hay más ejemplos)

Sirve para tener disponible otra tabla creada por nosotros para sacar columnas e información de ella

DIVISION EN SQL

Podemos encontrar un tipo de consultas que requieren la obtención de filas de una tabla que estén relacionadas con todas las de otra tabla.

Ejemplos:

"Clientes que tienen cuenta en todas las sucursales".

"Clientes que han hecho pedidos de todos los productos"

Para resolverlo, se pone el enunciado de la consulta en negativo y luego expresarlo en SQL.

"Clientes para los cuales no hay un producto que no hayan pedido"

Consulta: Códigos de los clientes con cuenta en todas las sucursales que están en 'MURCIA'

- Código de los clientes tales que NO EXISTE una sucursal de Murcia en la que NO tengan cuenta

-- Dos correlaciones

```
SELECT codigo  
FROM cliente C  
WHERE NOT EXISTS  
(SELECT * ← 1. NO EXISTE una sucursal de Murcia...  
   FROM sucursal S  
   WHERE ciudad = 'MURCIA'  
   AND NOT EXISTS  
     (SELECT * ← 3. ...en la que NO tenga una cuenta  
       FROM cuenta Q  
       WHERE Q.cliente_cod = C.codigo  
         AND Q.sucursal_cod = S.codigo));  
  
2. Esta SELECT saca las cuentas del cliente C en la sucursal S
```

Soluciones

-- Una correlación (un poquito mejor que la anterior)

```
SELECT codigo  
FROM cliente C  
WHERE NOT EXISTS  
(SELECT * ← 1. NO EXISTE una sucursal de Murcia...  
   FROM sucursal S  
   WHERE ciudad = 'MURCIA'  
   AND S.codigo NOT IN ← NOT IN en vez de NOT EXISTS  
     (SELECT sucursal_cod  
      FROM cuenta Q ← 2. ...en la que NO tenga unc  
      WHERE Q.cliente_cod = C.codigo));
```

La comparación que se hace en NOT IN evita la segunda comparación dentro de la subconsulta

EVALUACION

Sólo son obligatorios el SELECT y el FROM.

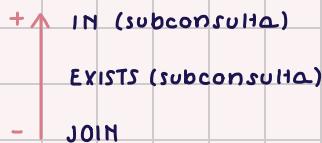
Orden de ESCRITURA
SELECT lista columnas o funciones
FROM lista tablas
WHERE condición para filas
GROUP BY lista columnas agrupación
HAVING condición para grupos
ORDER BY lista columnas ordenación

↓ Orden de EVALUACIÓN	
1. FROM lista tablas	Acceso a tablas, o reunión
2. WHERE condición para filas	Descarta las filas que incumplen la condición
3. GROUP BY lista columnas agrupación	Crea grupos de filas
4. HAVING condición para grupos	Descarta los grupos de filas que la incumplen
5. SELECT lista columnas y/o funciones	Para el conjunto de filas resultado, elige columnas indicadas, calcula operaciones y funciones
6. ORDER BY lista columnas ordenación	Ordena las filas del resultado

SQL ofrece flexibilidad, ya que una misma consulta se puede resolver de varias maneras.

Buenas prácticas

Se recomienda usar, en orden de elegancia y claridad:



Hemos de evitar el uso de SELECT * por temas de eficiencia (el SGBD accedería a metadatos)

Es bueno usar pseudónimos para las columnas cuando se realiza un JOIN

Usar sangrías y espaciados en la redacción de consultas.

Debemos evitar la correlación cuando sea posible. Deterioran mucho la eficiencia. Se puede traducir a una consulta usando JOIN

NO USAR el anti-JOIN →
es incorrecto (usar
NOT IN o WHERE).

SELECT e.nombre, e.apellido
FROM empleado e
JOIN familiar f
ON e.nss <> f.nssemp;

ANTI-JOIN
DETECTED
¡¡Aaaargh!!

e.nombre	e.apellido	e.nss	... nssemp	numero	nombre	... parentesco
MACARENO	SOZO	111 ...	123	1	JONAS	... HIJO
MACARENO	SOZO	111 ...	222	1	ELEUTERIO	... HIJO
MACARENO	SOZO	111 ...	321	1	RENATA	... HUA
MACARENO	SOZO	111 ...	321	2	RÓMULA	... ABUELA
MACARENO	SOZO	111 ...	321	3	TORCUATA	... ABUELA
JONÁS	SOLANO	123 ...	111	1	JULIANA	... MADRE
JONÁS	SOLANO	123 ...	111	2	SINFOROSA	... ABUELA
JONÁS	SOLANO	123 ...	222	1	ELEUTERIO	... HIJO
JONÁS	SOLANO	123 ...	321	1	RENATA	... HUA
JONÁS	SOLANO	123 ...	321	2	RÓMULA	... ABUELA
JONÁS	SOLANO	123 ...	321	3	TORCUATA	... ABUELA
EUSEBIO	MULETAS	222 ...	123	1	JONAS	... HIJO
EUSEBIO	MULETAS	222 ...	111	1	JULIANA	... MADRE
EUSEBIO	MULETAS	222 ...	111	2	SINFOROSA	... ABUELA
EUSEBIO	MULETAS	222 ...	321	1	RENATA	... HUA
EUSEBIO	MULETAS	222 ...	321	2	RÓMULA	... ABUELA
EUSEBIO	MULETAS	222 ...	321	3	TORCUATA	... ABUELA
RIGOBERTA	CALAVERA	321 ...	123	1	JONAS	... HIJO
RIGOBERTA	CALAVERA	321 ...	111	1	JULIANA	... MADRE
RIGOBERTA	CALAVERA	321 ...	111	2	SINFOROSA	... ABUELA
RIGOBERTA	CALAVERA	321 ...	222	1	ELEUTERIO	... HIJO
FILOMENA	RASCAS	234 ...	123	1	JONAS	... HIJO
FILOMENA	RASCAS	234 ...	111	2	JULIANA	... MADRE
...

se combina cada empleado
con TODOS los familiares
que no sean tuyos

En cuanto eficiencia:



(Aunque a veces un JOIN es
más eficiente)

Modificación de datos

Sentencias SQL que permiten introducir nuevas filas o modificarlas.

INSERT

Añade una fila completa a una tabla, incluye el nombre de la tabla y la lista de valores.

INSERT INTO empleado

```
VALUES ('CEFERINA', 'SALSIPUEDES', 444, '44C',
        TO_DATE('30-DIC-92', 'dd-MON-yy'),
        'YECLA', 'S', 3700, 'D4', NULL, 0);
```

Aquí, los valores se dan en el orden en el que están las columnas en la tabla.

Si queremos dar los valores en otro orden:

orden específico

```
INSERT INTO empleado (nombre, apellido, nss, dni, dep, salario, nssjefe,
                      ciudad, fechanacim, est_civil, cuantos_familiares)
```

```
VALUES ('CEFERINA', 'SALSIPUEDES', 444, '44C', 'D4', 3700, NULL,
        'YECLA', TO_DATE('30-DIC-1992', 'dd-MON-yyyy'), 'S', 0);
```

Es posible omitar valores de las columnas si estas aceptan NULL o si tienen valor por defecto.

El SGBD rechaza o acepta las inserciones según incumplan o no las restricciones asociadas a la tabla.

Es posible llenar una tabla extrayendo su contenido de la base de datos, mediante una SELECT.

* Sea una tabla INFO_DEP vacía, creada mediante esta sentencia
CREATE TABLE info_dep (
 nombre_dep VARCHAR2(25) NOT NULL,
 num_empleados NUMBER(3) DEFAULT 0 NOT NULL,
 suma_salarios NUMBER(8) NOT NULL
)

► Almacenará el nombre de cada departamento, cuántos empleados tiene y la suma de los salarios de sus empleados

```
INSERT INTO info_dep (nombre_dep, num_empleados, suma_salarios)
SELECT d.nombre, COUNT(*), SUM(salario)
FROM departamento d JOIN empleado e ON d.coddep = e.dep
GROUP BY d.nombre;
```

OJO! info_dep no será una tabla actualizada.

* Crear una tabla para almacenar datos de empleados nacidos antes de 1960

```
CREATE TABLE emp_veterano (nss, nombre, jefe, depto)
AS SELECT nss, nombre || '' || apellido, nssjefe, dep
FROM empleado
WHERE EXTRACT(YEAR FROM fechanacim) < 1960;
```

EMP_VETERANO			
nss	nombre	jefe	depto
111	MACARENO SOSO	NULL	D1
123	JONÁS SOLANO	111	D1
333	CASIANA FABERGÉ	123	D3

Otra manera de crear

tablas usando:

```
CREATE TABLE <tabla> AS SELECT ..
```

DELETE

Elimina filas completas de una tabla (solo una en el FROM):

```
DELETE FROM empleado  
WHERE apellido = 'ROLLER';
```

```
DELETE FROM departamento  
WHERE nssdire = 111;
```

```
DELETE FROM empleado  
WHERE dep IN  
(SELECT coddep  
FROM departamento  
WHERE nombre = 'INVESTIGACION');
```

- 1º se accede a la tabla
- 2º se ejecuta el WHERE
- 3º se ejecuta el borrado

El WHERE servirá para seleccionar

las filas que se quieran borrar.

Si no hay WHERE, se eliminarán todas las filas. !!

UPDATE

Modifica valores de columnas de una o más filas de una tabla.

* Cambiar el nombre del departamento D4 por 'FORMACION' y asignarle un director

```
UPDATE departamento  
SET nombre = 'FORMACION', nssdire = 444  
WHERE coddep = 'D4';
```

- 1º se accede a la tabla
- 2º se ejecuta el WHERE
- 3º se ejecuta el SET

SET → modifica (valor, NULL, DEFAULT)

WHERE → selecciona

```
UPDATE empleado SET nssjefe = NULL  
WHERE apellido = 'FUERTES';
```

```
UPDATE empleado SET salario = DEFAULT  
WHERE nss = 333;
```

✳ Ejemplos ✳

* Asignar a todo director de departamento un salario igual a la media de los salarios de su departamento más el 25% de dicha media

```
UPDATE empleado d SET salario = ( SELECT AVG(salario) * 1.25  
                                    FROM empleado e  
                                    WHERE e.dep = d.dep )
```

1º selecciona sólo las filas de EMPLEADO que son directores

```
WHERE nss IN ( SELECT nssdire  
                  FROM departamento );
```

2º para cada una de esas filas, calcula la media de su departamento y le asigna el nuevo salario

Tema 9: Reglas de Integridad y otros

Reglas de integridad

Al crear un esquema de BD Relacional, estas restricciones quedan almacenadas junto con los datos. Formarán parte de las **metadatos**.

La **integridad** es la consistencia o corrección de los datos almacenados en la base de datos.

COMPONENTES

- **Nombre**: aparece en los mensajes de error.
- **Restricción**: expresión booleana
- **Respuesta a un intento de incumplimiento**

□ **Nombre** departamento_director_ok

Restricción NOT EXISTS (SELECT *
FROM departamento
WHERE nssdire NOT IN (SELECT nss
FROM empleado
WHERE dep = coddep))

Respuesta a un intento de incumplimiento
Siempre RECHAZAR

Un INSERT, un UPDATE o un
DELETE podría incumplir
esta 3^a regla

→ ejemplos

□ **Nombre** emp_fk_emp -- en EMPLEADO

Restricción FOREIGN KEY(nssjefe) REFERENCES EMPLEADO(nss)

□ Los valores de "nssjefe" deben existir en la columna "nss" de EMPLEADO

Respuesta a un intento de incumplimiento

En el INSERT: RECHAZAR; En el DELETE: SET NULL; En el UPDATE: CASCADE

CATEGORÍAS

1. De dominio (no las veremos)
2. De tabla
3. Generales

REGLAS DE INTEGRIDAD DE TABLA

Restricción asociada específicamente a una tabla. No existe si la tabla no existe.

Se especifica dentro de CREATE TABLE. Son comprobadas inmediatamente.

Pueden ser:

- Restricciones de columna
- Claves: primaria, alternativa o ajena.
- RI añadida con ALTER TABLE.

REGLAS DE INTEGRIDAD GENERALES

Se refieren a los **assertos**. Son restricciones que van más allá de dominio y tabla.

[BD de ventas de productos] Un cliente puede conseguir un 15% de descuento en sus compras siempre que sea cliente desde hace más de 3 años y haya comprado más de 2 productos al año. ← ejemplo

[BD de una universidad] Un alumno puede matricularse de un máximo de 9 créditos de libre elección.

Incluyen un predicado con la condición que se debe cumplir y puede involucrar varias columnas y tablas.

```
CREATE ASSERTION RI_libro_tiene_copias
CHECK (NOT EXISTS (SELECT *
                   FROM libro
                   WHERE isbn NOT IN (SELECT isbn
                                         FROM ejemplar)));
```

} nombre
} condición → en negativo

Son un elemento independiente de tablas y vistas.

Para eliminar un aserto: `DROP ASSERTION <nombre_aserto>`

Oracle no implementa la creación de asertos, pero podemos ejecutar la `SELECT` de la condición y si se cumple el aserto, nos devolverá una tabla vacía. !!

*Todo departamento debe tener al menos un empleado

► No existe departamento cuyo código no esté referenciado desde un empleado

```
CREATE ASSERTION RI4_empleado_en_departamento
CHECK (NOT EXISTS (SELECT * FROM departamento
                     WHERE coddep NOT IN (SELECT dep
                                         FROM empleado)));
```

Existen unas reglas llamadas **pseudo-reglas de integridad** que son:

- Especificación del tipo de datos
 - Definición de vista (más adelante)
- ↳ vienen implícitas

COMPROBACIÓN DE RESTRICCIONES

```
UPDATE empleado SET dep = 'D3' WHERE nss = 543; -- Gumersinda Mimos
```

► Comprobación de reglas de integridad:

- Tipo de datos de dep: el valor indicado es compatible
- Integridad referencial (clave ajena):
 - existe una fila en `DEPARTAMENTO` con `coddep = 'D3'`
- Aserto RI1a (entra al departamento D3: su salario debe ser $\geq 900\text{€}$)
- Aserto RI5... (salario inferior al del director de su nuevo departamento)

Cuando existe un ciclo referencial (una tabla referencia a otra que referencia a

la primera):

```
ALTER TABLE empleado
  DISABLE CONSTRAINT emp_fk_dep;
INSERT INTO empleado(...)
VALUES (...);
...
INSERT INTO empleado(...)
VALUES (...);
INSERT INTO departamento...
VALUES (...);
...
INSERT INTO departamento...
VALUES (...);
ALTER TABLE empleado
  ENABLE CONSTRAINT emp_fk_dep;
```

Desactivar una de las claves ajenas:
es necesario saber su nombre

Insertar filas en la tabla en la que se ha desactivado la clave ajena: las referencias a la otra tabla NO se comprueban (se han desactivado)

Insertar filas en la otra tabla: las referencias hacia la 1ª tabla sí se comprueban (no se han desactivado)

Reactivar la restricción de clave ajena: en este momento se comprueban las referencias (que se habían desactivado)

Vistas relacionales

Es una tabla obtenida como resultado de una consulta. Permite tratar a ese resultado como a una tabla.

CREATE VIEW personal
AS SELECT nss, nombre, nssjefe, dep
FROM empleado;

Está siempre actualizada, porque se crea cada vez que se consulta.

Una vista en sí no contiene información, si no que la deja ver.

Se usan para almacenar consultas complejas, presentar los datos con una perspectiva diferente, ocultar la complejidad de los datos, proporcionar seguridad, aislar a las aplicaciones de los cambios en la definición de las tablas base.

Podemos definir nuevos nombres para las columnas. Esto es obligatorio cuando alguna

CREATE VIEW familiar_empleado (empleado, familiar, parentesco)
AS SELECT e.nombre, f.nombre, f.parentesco
FROM empleado e JOIN familiar f ON e.nss = f.nssemp;

columna es una concatenación o cálculo.

Es posible generar nuevas columnas en la vista, aplicando operaciones sobre las columnas extraídas.

Además, una vista puede aparecer en el FROM para crear otra vista.

CREATE VIEW directivo (nss, nombre, coddep, departamento)
AS SELECT p.nss, p.nombre, d.coddep, d.nombre
FROM personal p JOIN departamento d
ON p.dep = d.coddep
WHERE p.nss IN (SELECT nssjefe
FROM empleado);

PERSONAL es una vista creada anteriormente

Vemos que una vista puede ser "tabla base" de otra vista

① CREATE VIEW personal
AS SELECT nss, nombre, nssjefe, dep
FROM empleado;

VISTAS EN ORACLE

Creación:

CREATE [OR REPLACE] [[NO] FORCE] VIEW nombre_vista
[(lista_nombres_columnas)]
AS consulta_definición
[WITH { READ ONLY
| CHECK OPTION } [CONSTRAINT nombre_restricción]]

① La CHECK OPTION la veremos después

OR REPLACE: reemplazar si ya existe

FORCE: crear la vista aunque no existan las tablas base o no haya permisos.

WITH READ ONLY: prohíbe hacer UPDATE,

INSERT y DELETE sobre las tablas base a través de la vista

Si lo que se desea es modificar la vista usamos la opción CREATE OR REPLACE.

Se pueden hacer consultas con vistas como si fueran tablas.

Sentencia de usuario	Traducción (lo que realmente se ejecuta)
<pre>SELECT * FROM veterano WHERE dep = 'D1';</pre>	<pre>SELECT nombre, dni, nss, fechanacim, dep FROM empleado WHERE fechanacim < TO_DATE('01/01/1970', 'dd/mm/yyyy') AND dep = 'D1';</pre>

CREATE OR REPLACE VIEW veterano

```
AS SELECT nombre, dni, nss,  
fechanacim, dep  
FROM empleado  
WHERE fechanacim <  
TO_DATE('01/01/1970',  
'dd/mm/yyyy');
```

También se pueden manipular datos a través de estas

Sentencia de usuario	Traducción (lo que realmente se ejecuta)
<pre>INSERT INTO veterano (nombre, apellido, nss, dni, fechanacim, salario, dep) VALUES ('EVA','EME',789,'78E', TO_DATE('09/07/1967', 'dd/mm/yyyy'), 980,'D4');</pre>	<pre>INSERT INTO empleado (nombre, apellido, nss, dni, fechanacim, salario, dep) VALUES ('EVA', 'EME', 789, '78E', TO_DATE('09/07/1967', 'dd/mm/yyyy'), 980, 'D4');</pre>

Por otro lado, actualizar a través de una vista definida sobre varias tablas base puede dar lugar a ambigüedad (que puede traducirse a 2 o más actualizaciones distintas). Ejemplo detallado en el ppt.

En general, podemos afirmar que:

- Una vista con una sola tabla base (teniendo una columna clave primaria o alternativa) **SÍ** es actualizable.
- Una vista definida sobre varias tablas con JOIN, **NO** lo es.
- Una vista definida con agrupación y funciones de agregados **NO** lo es

Si añadimos WITH CHECK OPTION a la creación de vistas, el SGBD comprobará cada INSERT y UPDATE que se haga sobre la vista.

```
CREATE VIEW precario  
AS SELECT nombre, apellido, nss, dni, salario, dep  
FROM empleado  
WHERE salario < 550  
→ WITH CHECK OPTION;
```

Así, tanto el INSERT como el UPDATE anteriores fallarían

Se comprueba que se cumple la definición de la vista.

En Oracle se le puede además dar nombre a la cláusula, para que aparezca en los mensajes de error.

```
CREATE OR REPLACE NOFORCE VIEW plantilla  
AS SELECT nombre, apellido, nss, dni, salario, dep  
FROM empleado  
WHERE salario < 3000 -- se ocultan datos de "jefazos y jefazas"  
WITH CHECK OPTION CONSTRAINT plantilla_const;
```

Las vistas son un elemento **virtual**, no ocupa espacio de almacenamiento. Sólo se almacena su definición (en el INFORMATION_SCHEMA).

Las vistas se **eliminan** con: `DROP VIEW vista;`

Si formaban parte de la base de otra vista, esa no se elimina pero se marca como inválida.

índices

Son un concepto que pertenece al **nivel físico**.

Vamos a asumir que cada tabla se guarda en un fichero. Cada fila en un registro.

Los ficheros ocupan bloques y páginas de memoria.

¿Cómo encontramos una fila o varias en una tabla?

Una opción es la **búsqueda secuencial**. Pero es muy ineficiente y requiere muchas lecturas de bloque.

Fichero EMPLEADO

nss	nombre	...	fechanacim	dep
123	JONÁS		10/10/1945	D1
321	RIGOBERTA		12/11/1974	D3
222	EUSEBIO		01/01/1969	D2
111	MACARENO		06/04/1944	D1
333	CASIANA		15/06/1943	D3
234	FILOMENA		18/07/1970	D1
543	GUMERSINDA		10/02/1980	NULL
444	CEFERINA		30/12/1992	D4
456	PRUDENCIO		22/01/1995	D4

Fichero con 5 bloques
Cada bloque contiene 2 registros

Un índice es una estructura de datos auxiliar que permite acelerar el acceso a las filas de una tabla con base en los valores de una o más columnas.

En la tabla empleado, la columna de **indexación** podría ser el nss.

Índice IDX_NSS_EMPLEADO	
nss	bloque
111	B2
123	B1
222	B2
234	B3
321	B1
333	B3
444	B4
456	B5
543	B4

1º Coger todos los valores del campo de indexación
2º Ordenarlos
3º Para cada uno de los valores, crear una **entrada de índice**. El campo "nss" contendrá el valor. El campo "bloque" contendrá un puntero al bloque de EMPLEADO en el que está el registro con tal valor en "nss"

Fichero EMPLEADO

bloque	nss	nombre	...
B1	123	JONÁS	
	321	RIGOBERTA	
B2	222	EUSEBIO	
	111	MACARENO	
B3	333	CASIANA	
	234	FILOMENA	
B4	543	GUMERSINDA	
	444	CEFERINA	
B5	456	PRUDENCIO	

↓
puntero al bloque

Se pueden crear varios índices sobre una misma tabla. Este ocupa poco.

Oracle crea índices automáticos con clave primaria y alternativas.

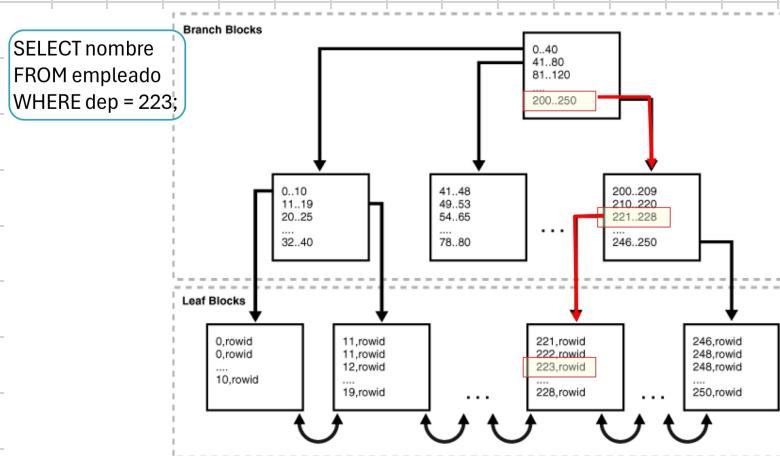
A veces crea índices con columnas no clave que se usan mucho en el WHERE y JOIN pero sólo si:

- La tabla tiene muchas datos
- Sus consultas más frecuentes recuperan un bajo porcentaje de datos (menos del 20%).

```
CREATE INDEX nombre_idx
ON nombre_tabla (columna [ASC | DESC]
[, columna [ASC | DESC]...])
...;
```

Son mantenidas por el SGBD, son independientes de los datos de la tabla.

Los más comunes son los índices en árbol B.



¿Qué es el rowid?

Identificador de fila. Contiene el fichero, el bloque y la posición, codificado en base 64.

AAAPzSAAEAAAABzAAA

Para eliminar un índice: `DROP INDEX nombre;`

Se hace cuando la tabla es pequeña o hay pocas entradas en el índice, o cuando no se esperan hacer más consultas con esa columna de indexación.

Si queremos eliminar un índice asociado a una clave hay que desactivar o eliminar la restricción de clave primero.

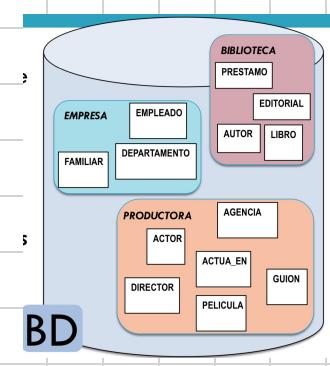
Esquema

Una base de datos puede contener varios esquemas.

Un esquema agrupa tablas y otros elementos.

```
CREATE SCHEMA nombre_de_esquema
AUTHORIZATION identificador_de_autorización
```

Un usuario puede crear una tabla en el esquema de otro si tiene permisos para ello.



En Oracle:

CREATE SCHEMA AUTHORIZATION cuenta;

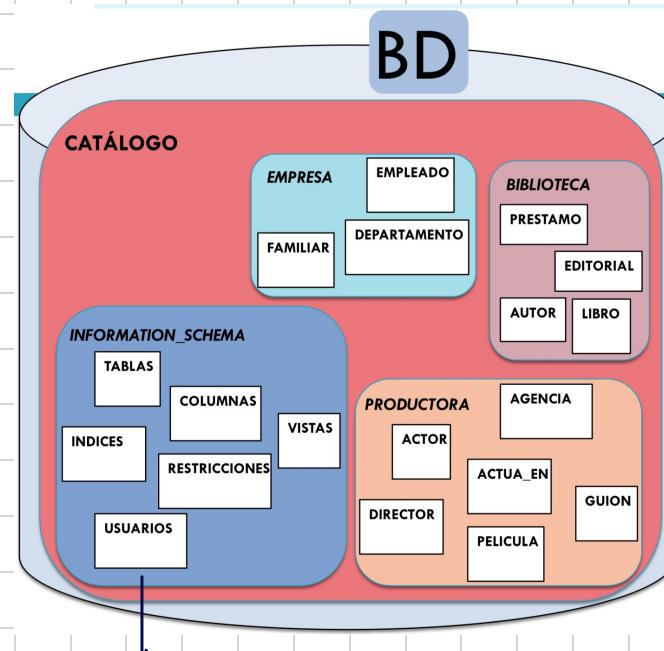
No hace literalmente nada (sólo un esquema por cuenta y no tiene nombre).

Tampoco permite eliminar esquemas, que en otro lenguaje sería:

DROP SCHEMA nombre [RESTRICT/CASCADE]

↓ ↓
sólo lo elimina elimina todo
si no tiene nada

Un **catalogo** recoge todos los esquemas de un mismo entorno SQL.



En Oracle es Data Dictionary
(metadatos)

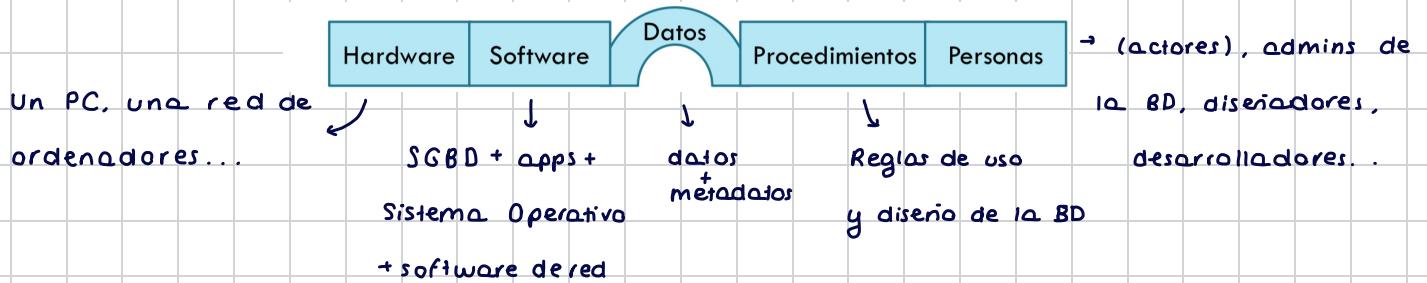
Puede haber claves ajena que refieran a tablas de distintos esquemas pero mismo catálogo.

Para acabar, recordemos.

BD = DATOS + METADATOS

Tema 10: Sistemas de bases de datos

Componentes

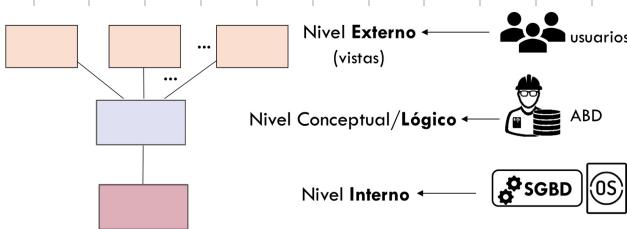


ACTORES

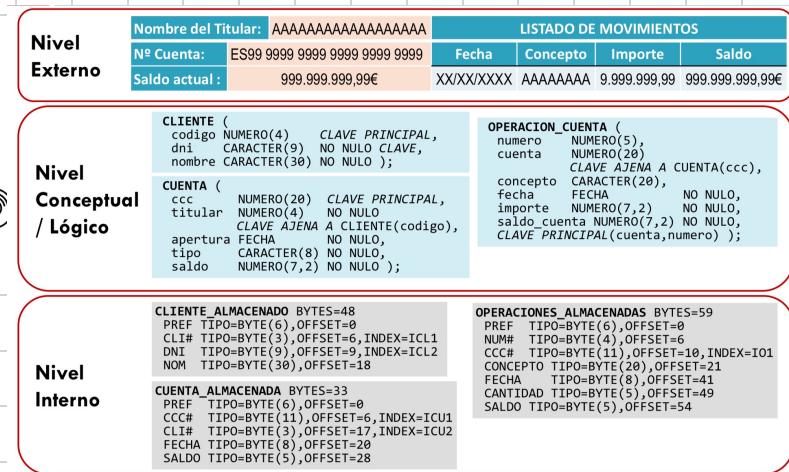
- Administrador de la base de datos (ABD):
Administra la BD, el SGBD y el software de aplicaciones o programas de acceso. Implementa la estructura de la BD y restricciones de los datos. Crea /modifica estructuras de almacenamiento. Controla la seguridad (permisos). Define las copias de seguridad. Garantiza el funcionamiento correcto del sistema.
- Diseñadores de la base de datos:
Recogen las necesidades y requisitos del cliente. Con ello crean **esquemas conceptuales (E-R)** identificando relaciones. Eligen las estructuras para representar los datos.
- Desarrolladores de Software:
 - Analistas de sistemas: determinan necesidades de procedimiento y especifican las operaciones.
 - Desarrolladores de aplicaciones: implementan las especificaciones.
- Usuarios finales:
Son los clientes.
 - Inexpertos: no son conscientes de la existencia del SGBD y acceden a la BD mediante sencillos programas de aplicación.
 - Avanzados: conocen el SGBD. Pueden usar SQL.

Arquitectura de 3 niveles ANSI - SPARC

La complejidad de un sistema de bases de datos se oculta mediante niveles de abstracción.

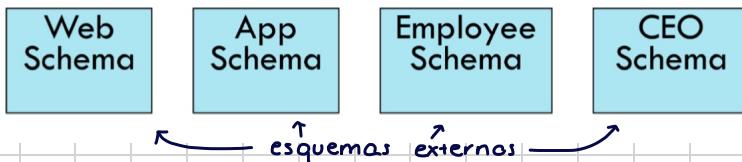


Con esto logramos aislar los distintos niveles y hacerlos independientes: un cambio en un aspecto físico de almacenamiento no debería afectar a la estructura de la BD.



NIVEL EXTERNO O DE VISTAS

Describe que parte de los datos es relevante para cada usuario o aplicación.



Para cada tipo de usuario se le da un esquema.

Pueden existir varias vistas del mismo esquema

NIVEL LÓGICO

Describe qué datos están almacenados en la BD y sus relaciones. Usa el esquema conceptual o lógico.

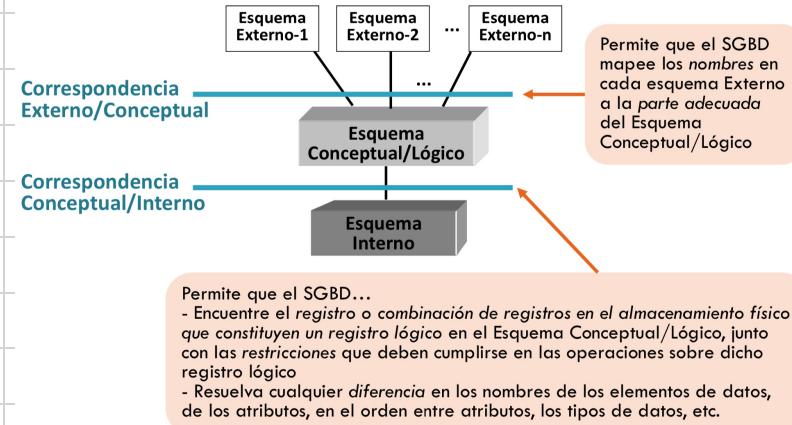
- Esquema conceptual → más cercano al usuario
- Esquema lógico → más cercano al SGBD

NIVEL INTERNO

Describe cómo los datos están almacenados en la BD. Usa el esquema interno: asigna el espacio para datos e índices, describe los tipos de registros almacenados, define estructuras de acceso... Muy cercano al nivel físico pero no trata con registros físicos.

Los 3 niveles de la A3N son descripciones de datos. Los datos reales están solo en el nivel físico.

El SGBD es el responsable de mantener la **correspondencia** entre niveles. Estas correspondencias están guardadas en el **INFORMATION_SCHEMA** (metadatos). Se debe comprobar la **consistencia** entre ellos.



Independencia de datos

"Capacidad de modificar el esquema de un nivel sin tener que cambiar el esquema del nivel inmediato superior".

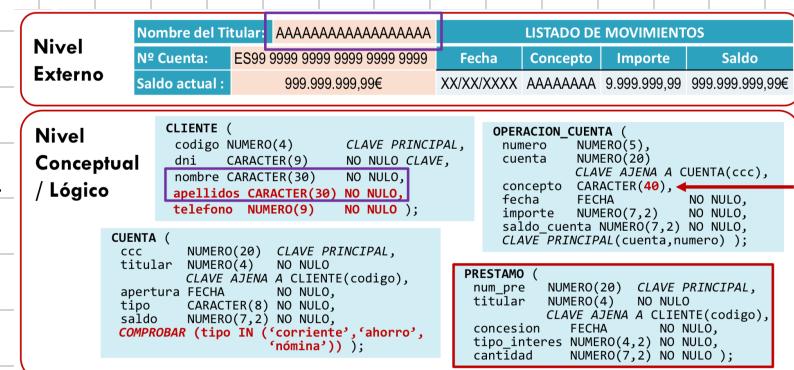
INDEPENDENCIA LÓGICA

Modificar el esquema lógico/conceptual sin alterar esquemas externos.



El esquema externo permanece inalterado.

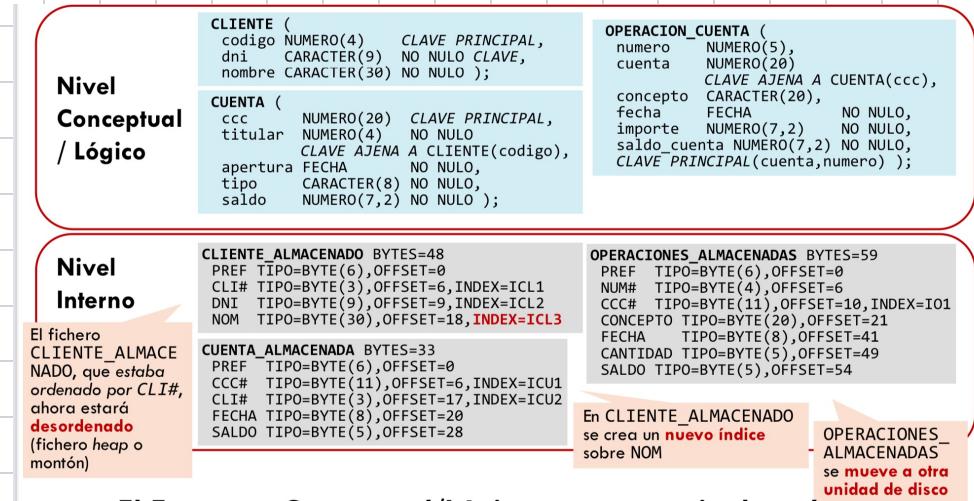
Sólo hay que modificar la correspondencia entre esquemas:
"Nombre del titular" ahora concatena "nombre" y "apellidos".



INDEPENDENCIA FÍSICA

Capacidad de modificar el esquema interno sin alterar el esquema lógico/conceptual.

Lo único que los usuarios notarían es una mejora en el rendimiento.



En resumen: gracias a la arquitectura de 3 niveles y a los metadatos; la modificación de un nivel no modifica un nivel superior, si no la correspondencia entre niveles.

El único inconveniente es que mantener las correspondencias actualizadas supone un coste en eficiencia.

Además, la independencia de datos incluye la independencia entre programas y datos.

Funciones del SGBD

1 ALMACENAMIENTO, RECUPERACIÓN Y ACTUALIZACIÓN DE DATOS

(Tema 11)

2 SERVICIO PARA PERMITIR INDEPENDENCIA DE DATOS

ASN ✓

3 METADATOS ACCESIBLES PARA EL USUARIO

INFORMATION_SCHEMA ✓

4 SERVICIOS DE INTEGRIDAD ✓

5 SOPORTE DE TRANSACCIONES

Transacción: secuencia de acciones que lee y/o actualiza el contenido de la BD.

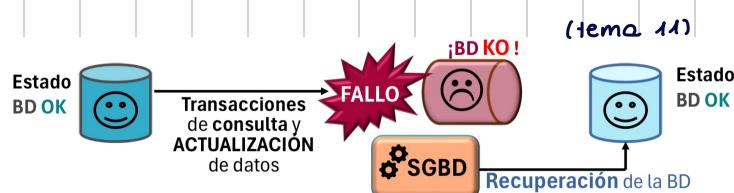
Sopor te: asegura que, o todos los cambios se han hecho correctamente, o que no se ha hecho ninguno de los cambios.

Se garantiza incluso en presencia de fallos y teniendo en cuenta la conurrencia de varios usuarios.

6 SERVICIO DE CONTROL DE CONCURRENCIA

Garantiza que la BD es actualizada correctamente cuando múltiples usuarios/aplicaciones modifican los datos a la vez

7 SERVICIO DE RECUPERACIÓN DE FALLOS



Recuperar las BD tras fallos:
system crash, errores hardware,
errores software...

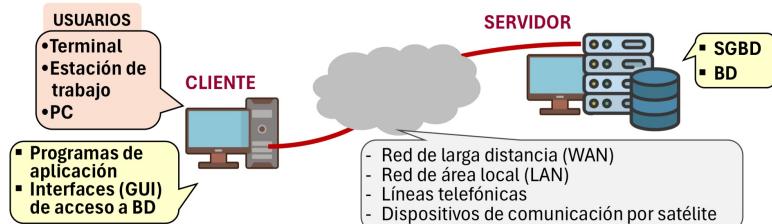
8 SERVICIO DE AUTORIZACIÓN (SEGURIDAD)

Control de acceso selectivo

1. Solo usuarios autorizados. → cuentas de usuario y contraseña
2. Solo a ciertas partes de la BD.] → restricciones para cada cuenta
3. Solo para realizar ciertas operaciones.

9 SOPORTE PARA COMUNICACIÓN DE DATOS

Garantiza que los usuarios accedan a una BD centralizada desde ubicaciones remotas.



10 UTILIDADES ADICIONALES

Herramientas de importación/exportación de datos, utilidades de monitorización y supervisión, análisis estadístico, herramientas de backup, etc.

TEMA II: Procesamiento de transacciones

Soporte de transacciones

Una transacción es una acción, o una secuencia de acciones, que lee y/o actualiza el contenido de la base de datos. Se ejecuta como una unidad: es una unidad lógica de procesamiento.

Ejemplo. transferencia bancaria

Transferir 200€ de una cuenta X a otra cuenta Y

1. Leer el saldo de X
2. Comprobar que el saldo de X es superior a 200
Si no, indicar que no se puede realizar la transferencia y finalizar
3. Restar 200 del saldo de X
4. Sumar 200 al saldo de Y
5. ...

Transferir 200€ de una cuenta X a otra cuenta Y

1. SELECT saldo FROM cuenta WHERE ccc = X;
2. SI (saldo < 200) ENTONCES ko_saldo_insuficiente(saldo);
3. UPDATE cuenta SET saldo=saldo - 200 WHERE ccc = X;
4. UPDATE cuenta SET saldo=saldo + 200 WHERE ccc = Y;
5. ...

¿Qué pasa si hay un error?

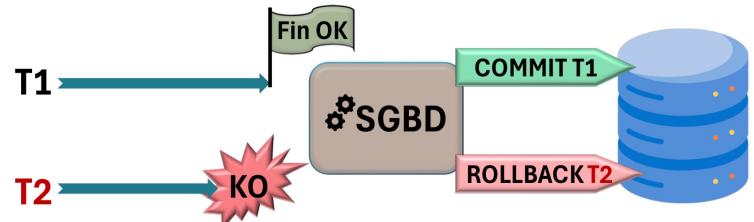
Una transacción es ATÓMICA, o se ejecuta completa, o no se ejecuta

SOPORTE DE TRANSACCIONES

Toda transacción termina con éxito o con fracaso.

↓ ROLLBACK

↓ COMMIT



PROPIEDADES ACID DE UNA TRANSACCION

A Atomicidad

Subsistema de Recuperación del SGBD

Responsable

C Consistencia

Programadores + Subsistema de Integridad del SGBD

I Isolation (aislamiento)

Subsistema de Concurrencia del SGBD

D Durabilidad

Subsistema de Recuperación del SGBD

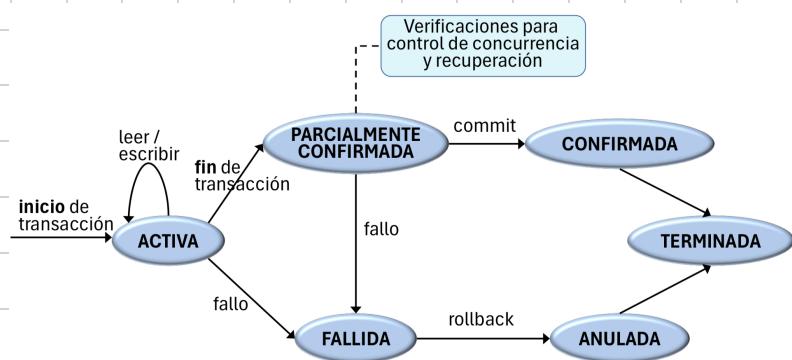
Atomicidad: ejecución "Todo o nada"

Consistencia: una transacción lleva la BD de un estado de consistencia a otro estado de consistencia.

Isolation (aislamiento): la ejecución de una transacción es independiente de las otras.

Durabilidad: los cambios se guardan en la BD permanente.

Estados de una transacción:



Inicio de una transacción → al ejecutar una sentencia SQL incluida o incluida en un programa

Finalización de una transacción → al ejecutar explícitamente COMMIT o ROLLBACK, al terminar el programa en ejecución.

SOPORTE DE TRANSACCIONES (EN ORACLE)

Una transacción inicia en Oracle cuando :

- No hay una transacción en proceso y se ejecuta una sentencia LMD: INSERT, UPDATE, DELETE, SELECT
- Cuando se ejecuta una sentencia LDD: CREATE, ALTER, DROP, RENAME, COMMENT
(si hay ya una transacción en proceso, Oracle la finaliza con COMMIT y comienza la nueva)

Se finaliza con:

- COMMIT explícito: ejecución de la sentencia COMMIT.
- COMMIT implícito: el programa finaliza sin errores Cada sentencia LDD es tratada como una transacción.
- ROLLBACK explícito
- ROLLBACK implícito → errores, cerrar la ventana de la BD directamente

Conexión a SQL Developer	(inicio de sesión interactiva)
SELECT ...;	Inicia T1
SELECT ...;	
UPDATE...;	
DROP TABLE...;	Finaliza T1 (COMMIT) Inicia T2 y finaliza T2 (COMMIT)
INSERT INTO...;	Inicia T3
UPDATE...;	
SELECT...;	
INSERT INTO...;	
ROLLBACK;	Finaliza T3 (KO) Deshace TODA modificación de T3
SELECT ...;	Inicia T4
DELETE ...;	
INSERT INTO...;	
INSERT INTO...;	
COMMIT;	Finaliza T4 (COMMIT)
Desconexión del SQL Developer	

Concurrencia de transacciones

Los sistemas de procesamiento de transacciones requieren alta disponibilidad y respuesta rápida.

T1: Transferir 200€ de la cuenta X a otra cuenta Y

1. SELECT saldo
FROM cuenta
WHERE ccc = X;
2. SI (saldo<200) ENTONCES
ko_saldo_insuficiente(saldo);
3. UPDATE cuenta
SET saldo=saldo - 200
WHERE ccc = X;
4. UPDATE cuenta
SET saldo=saldo + 200
WHERE ccc = Y;

T2: Ingresar 100€ en la cuenta X

1. SELECT saldo
FROM cuenta
WHERE ccc = X;
2. solicitar_confirmacion();
3. UPDATE cuenta
SET saldo=saldo + 100
WHERE ccc = X;

Si no existiera un control de la concurrencia la ejecución de las transacciones se intercalaría.

Esto puede provocar conflictos e interferencias.

Objetivo del SGBD: planificar las transacciones para que el efecto sea equivalente a ejecutarlas en serie.

El SGBD usa técnicas como el bloqueo (locking):

BLOQUEO

Si una transacción necesita leer un elemento X, antes debe obtener un **bloqueo compartido** sobre X. (si no lo consigue, espera).

Si una transacción desea escribir un elemento Y, antes debe conseguir un **bloqueo exclusivo** sobre Y.

Una vez realizada la operación, la transacción desbloquea el elemento de datos.

Un COMMIT o ROLLBACK libera los bloques.

El **nivel de aislamiento** de una transacción T define cuál es el grado de interacción de T con el resto de las transacciones.

Los SGBD implementan los niveles de aislamiento usando bloques.

Definir un nivel:

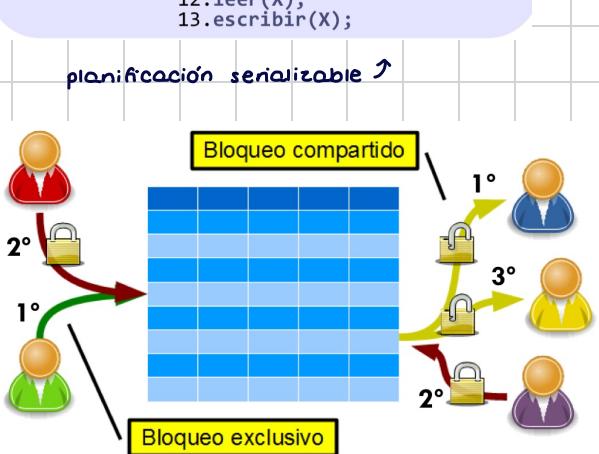
SET TRANSACTION ISOLATION LEVEL nivel;

(1º sentencia de la transacción)

Valores posibles:

Nivel de Aislamiento	Lectura sucia	Lectura no repetible	Lectura Fantasma
READ UNCOMMITTED	Possible	Possible	Possible
READ COMMITED	No	Possible	Possible
REPEATABLE READ	No	No	Possible
SERIALIZABLE	No	No	No

← problemas que permiten o evitan



- Lectura sucia (dirty read): una transacción T1 actualiza ciertos datos, después otra transacción T2 lee dichos datos. Si T1 hace ROLLBACK los datos vuelven al estado original. Los datos leídos por T2 no serán válidos.
- Lectura no repetible (Non-repeatable Read): después de que una transacción T1 lee ciertos datos de la BD, otra transacción T2 los actualiza y se confirma. Cuando T1 lee de nuevo los mismos datos encontrará que han cambiado: es una lectura no repetible, los mismos datos leídos 2 veces son diferentes.
- Lectura fantasma (Phantom Read): Una transacción T1 ejecuta una consulta, otras transacciones insertan nuevas filas y se confirman. La T1 ejecuta la misma consulta y el resultado muestra las nuevas filas.

Read uncommitted

los bloqueos de lectura y escritura son liberados en cuanto finaliza la sentencia

T1	T2 (READ UNCOMMITTED)
	SELECT SUM(salario) -- suma1 FROM empleado;
UPDATE empleado SET salario = salario * 1.05 WHERE nss IN (SELECT nssemp FROM familiar);	
	SELECT AVG(salario) T2 ve el UPDATE de T1 (Dirty Read) y calcula la media con los nuevos datos FROM empleado;
COMMIT;	
	SELECT SUM(salario) El resultado de la suma es diferente a la suma1 (Nonrepeatable Read) FROM empleado;

Read committed

Se leen datos modificados por transacciones confirmadas.

La lectura se libera al acabar la SELECT.

T1	T2 (READ COMMITTED)
	SELECT SUM(salario) -- suma1 FROM empleado;
UPDATE empleado SET salario = salario * 1.05 WHERE nss IN (SELECT nssemp FROM familiar);	
COMMIT;	
	SELECT AVG(salario)FROM empleado;
	SELECT SUM(salario) El resultado de esta suma es DIFERENTE a la suma1 (Nonrepeatable Read) FROM empleado;

Repeatable read

Se realizan siempre lecturas 'repetibles': los mismos valores para lecturas de los mismos datos. Todo bloqueo se libera al acabar la transacc.

T1 (READ UNCOMMITTED)	T2 (REPEATABLE READ)
	SELECT SUM(salario) -- suma1 FROM empleado;
UPDATE empleado SET salario = salario * 1.05 WHERE nss IN (SELECT nssemp FROM familiar);	
T1 NO puede completar el UPDATE porque T2 tiene bloqueadas las filas de EMPLEADO el prompt no aparece... (T1 espera)	T2 calcula la media con los mismos datos que la suma1 SELECT AVG(salario)FROM empleado;

Serializable

Aislamiento absoluto. Igual que el Repeatable Read pero además se hacen **bloqueos de rango**: se bloquean los datos seleccionados con cada **SELECT... FROM... WHERE**

Recuperación de fallos

El SGBD ofrece un mecanismo para recuperar la base de datos en caso de que resulte afectada de alguna manera por fallos de procesamiento.

TIPOS DE FALLOS

- Fallo local: solo afecta a la transacción fallida. Se pierden datos de T en memoria principal.
 - **Previsto:** excepción tratada. (1)
 - **No previsto:** error de programación, división por 0, desbordamiento... (2)
 - **Por imposición de concurrencia:** incumple la serialización, romper interbloqueo... (3)
- Fallo global: afecta a todos las transacciones en ejecución. Se pierden datos en memoria y quizá almacenamiento secundario.
 - **Fallo del sistema.** (4)
 - **Fallo de los medios de almacenamiento.** (5)
 - **Fallos físicos y catástrofes.** (6)

Recuperar la BD es restaurar un estado previo al fallo.

ESTRUCTURAS DE MEMORIA

Caché de Base de Datos: uno o más búferes en memoria principal. Es un área común a todas las transacciones. El volcado de la caché de BD al disco se hace tras ejecutar un comando como COMMIT o cuando esta se llena.

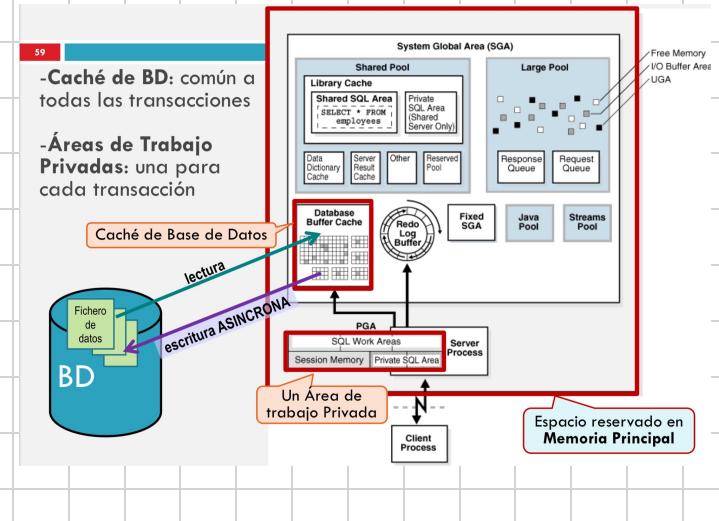
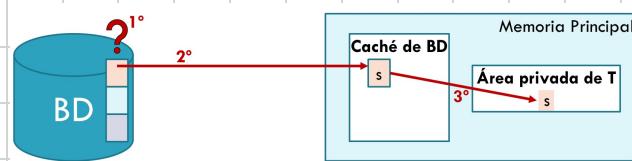
Áreas de trabajo privadas: cada T tiene una zona asignada en memoria principal. Se crea al iniciarse T y se elimina cuando T finaliza.

Lectura de la base de datos:

```
UPDATE empleado  
SET salario = salario * 1.1  
WHERE nss = 111;
```

leer(nss=111, salario)
salario = salario * 1.1
escribir(nss=111, salario)

representación visual:

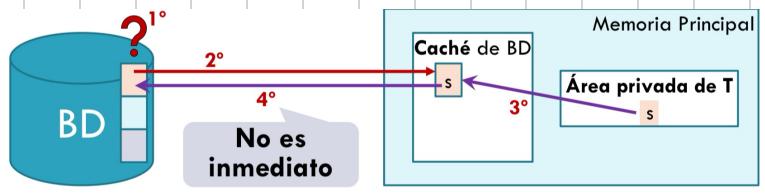


Escritura en la base de datos:

```
UPDATE empleado  
SET salario = salario * 1.1  
WHERE nss = 111;
```

leer(nss=111, salario) → salario = salario * 1.1
escribir(nss=111, salario)

El paso 1º solo se hace si el bloque no está en caché. (y el 2º)



Mta T puede modificar la BD antes de emitir COMMIT, consolidando algunas datos en disco, por ejemplo, porque fuera necesario hacer hueco en la caché. Son modificaciones 'no comprometidas'.

ARMAS PARA RECUPERACIÓN DE FALLOS

Una BD se recupera con redundancia y acciones de recuperación.

- Facilidades de **backup**, para realizar copias periódicas de la base de datos REDUNDANCIA
- Mecanismo de **bitácora**, para mantener la pista del estado actual de las transacciones y los cambios de los datos
- Un protocolo de **checkpoints** ACCIONES DE RECUPERACIÓN
- Un Gestor de Recuperación (Recovery Manager; subsistema del SGBD), que permite al sistema restaurar la BD a un estado consistente tras un fallo, mediante **técnicas y estrategias de recuperación**

Fichero bitácora: almacena detalles sobre las operaciones efectuadas por las transacciones.

Se mantiene en el almacenamiento secundario. Se pueden mantener varias copias. Se pueden realizar copias de seguridad periódicas. Existe un búfer de bitácora (~cache').

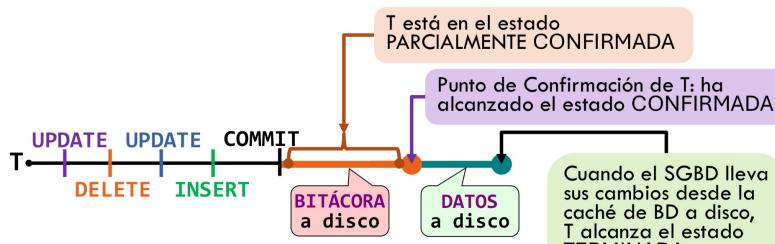
El fichero contiene entradas de estos tipos:

```
<INICIAR,T>  
<LEER,T,X>  
<ESCRIBIR,T,X, valor_anterior, valor_nuevo>  
<COMMIT,T>  
<ROLLBACK,T>  
<PUNTO DE CONTROL,...>
```

Si ocurre un fallo y en la bitácora no está la entrada <COMMIT,T> se debe deshacer T. Si existe esa entrada entonces T se rehace para asegurarnos que todos los cambios han sido llevados a disco.

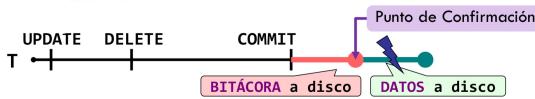
bitácora adelantada

NO se puede guardar en disco los datos modificados por T hasta que se haya escrito en disco toda la entrada de la bitácora para T (Write-Ahead-Logging).



En el punto de confirmación está asegurado que las modificaciones sobre los datos serán permanentes, sea cual sea el momento de volcar los cambios en el disco.

Situaciones:



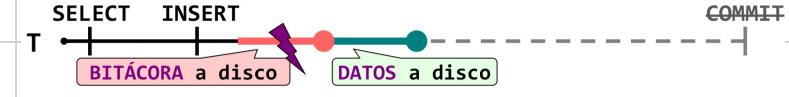
Rehacer T



Deshacer T



Deshacer T



Deshacer T

ESTRATEGIAS DE RECUPERACIÓN

- Tras un fallo de tipo 5 o 6:

Se restaura la última copia de seguridad y se reconstruye a un estado más actual usando la bitácora en disco.

- Tras un fallo tipo 1-4:

Deshacer las operaciones de las transacciones en curso. Rehacer, si es necesario, las transacciones confirmadas.

Protocolo de Checkpoints:

Cada m segundos, o tras n entradas <COMMIT,Ti> se coloca un punto de control en la bitácora. Supone:

- 1 - Suspender ejecución de las transacciones
- 2 - Forzar la escritura en disco del búfer de bitácora
- 3 - Forzar la escritura en disco de la caché de la BD.
- 4 - Escribir <PUNTO DE CONTROL> en la bitácora

5 - Escribir en un fichero especial de arranque la dirección de la entrada <PUNTO DE CONTROL>

6 - Reanudar la ejecución de transacciones.

La entrada <PUNTO DE CONTROL> contiene identificadores de las transacciones activas y la dirección de sus primeras y últimas entradas.

Algoritmo deshacer/rehacer:

Se accede a la última entrada <PUNTO DE CONTROL> Contiene la lista de transacciones activas. $A = \{T_2, T_3, T_1\}$

Se crea la lista de confirmadas. $C = \emptyset$

- Cada <COMMIT, T> mueve T de A a C

Al terminar de recorrer la bitácora:

- $A = \{T_1, T_3\}$ y $C = \{T_2\}$

Recuperación.

- Deshacer las de A.
- Rehacer las de C.

Siempre se debe deshacer primero y rehacer después.

- Para deshacer, se deshacen las operaciones en el orden inverso al de anotación en bitácora. Se van deshaciendo todas las T activas de forma intercalada.
- Para rehacer las operaciones, se hacen en el mismo orden al de la bitácora.

Se van rehaciendo todas las confirmadas intercaladamente.

Fichero Bitácora
...
<INICIAR, T4>
<ESCRIBIR, T4, ...>
<INICIAR, T2>
<ESCRIBIR, T2, ...>
<INICIAR, T1>
<COMMIT, T4>
<INICIAR, T3, ...>
<PUNTO DE CONTROL...>
<ESCRIBIR, T1, ...>
<ESCRIBIR, T3, ...>
<LEER, T1, ...>
<ESCRIBIR, T3, ...>
<LEER, T2, ...>
<ESCRIBIR, T1, ...>
<COMMIT, T2>
<LEER, T3, ...>

