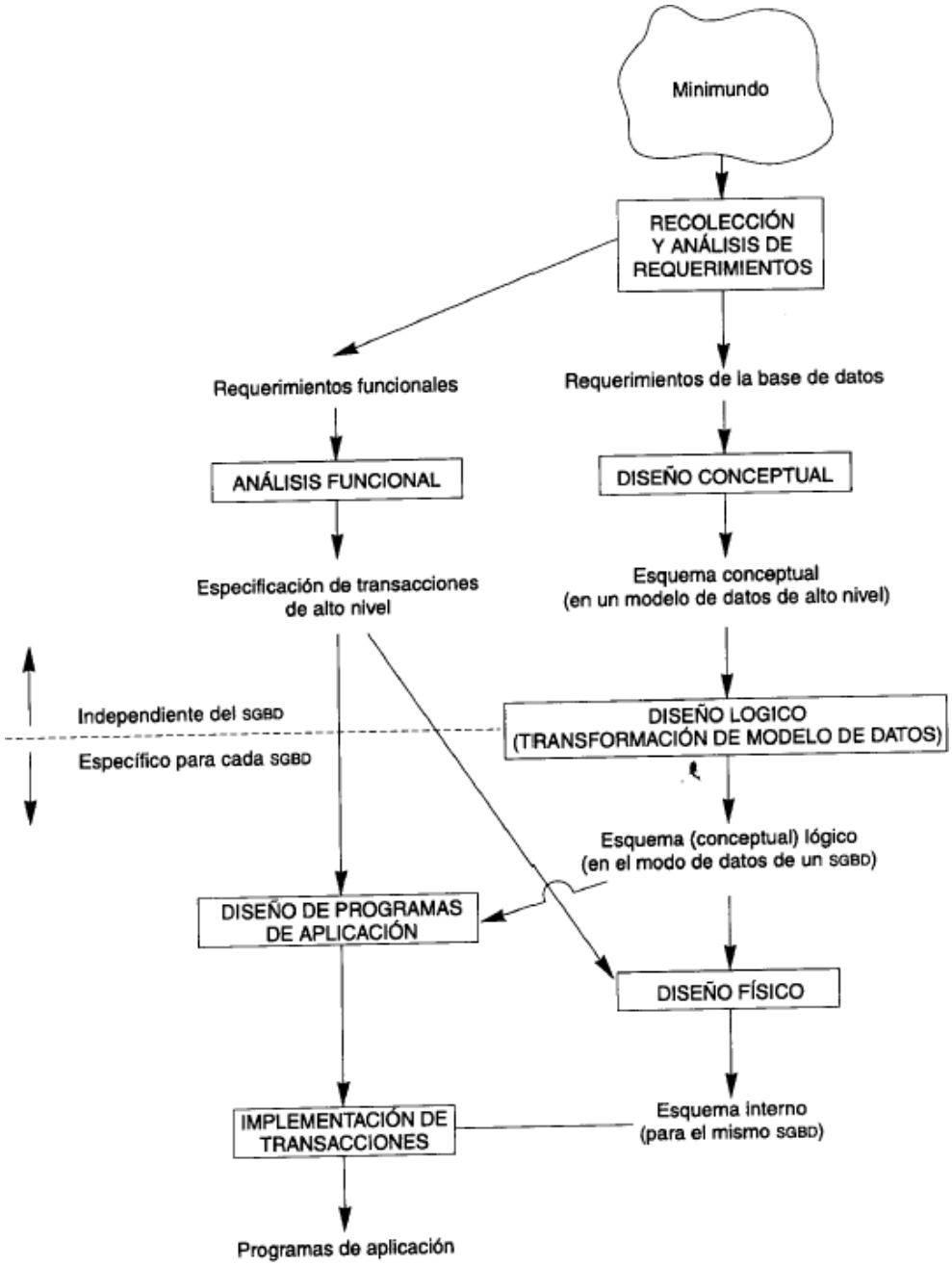
Diseño de Bases de Datos

Los pasos del diseño de una Base de Datos, representados en la siguiente figura, se pueden resumir en

- **Recolección y análisis de requerimientos**: En este paso recogemos información del sistema para el que debemos diseñar la Base de Datos.
- **Diseño conceptual**: Una vez recogidos todos los requisitos y conocido el problema, realizamos un primer esquema conceptual en algún lenguaje de alto nivel como es el *Modelo Entidad-Relación*
- **Diseño lógico**: El diseño conceptual debe ser ahora transformado en un diseño lógico, que es la transformación de un modelo conceptual a un modelo de datos concreto con el fin de poder representar el problema, más adelante, en algún software concreto. En nuestro caso usaremos el *Modelo Relacional*.
- **Diseño físico**: En este punto debemos aplicar el modelo lógico de datos del punto anterior sobre un SGBD concreto. Dependiendo del diseño físico escogido, tendremos un abanico de posibilidades en cuanto al software disponible. En nuestro caso hemos optado por un modelo relacional por lo que tendremos que escoger entre los SGBD relacionales disponibles. En este curso será *MySQL*.



Modelo Entidad/Relación

Es un modelo de datos que representa la realidad a través de *entidades*, que son objetos que existen y se distinguen de otros por sus características, que llamamos *atributos*. Además, estas entidades podrán o no estar relacionadas unas con otras a través de lo que se conoce como *relación*. Hay que tener en cuenta que se trata solamente de un modelo de representación, por lo que no tiene correspondencia real con ningún sistema de almacenamiento. Se utiliza en la etapa de Análisis y Diseño de una Base de Datos, por lo que habrá que convertirla a otro modelo antes de poder empezar a trabajar con ella.

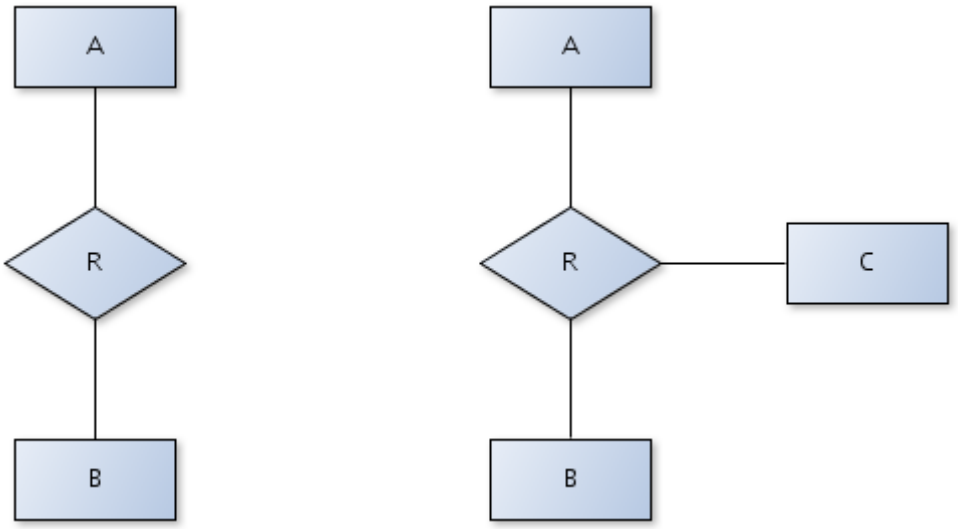
Una *entidad* es un objeto que existe en una realidad que queremos representar, por ejemplo, un alumno, que se distingue de otro por sus características como pueden ser: el nombre, los apellidos, el número de expediente, . . .

Las entidades se representan por el siguiente símbolo: Esas características que hacen que unas entidades se distingan de otras, son los *atributos*. El nombre, los apellidos y el número de expediente serían atributos de la entidad alumno. Los

atributos se representan por el siguiente símbolo: A su vez, podemos relacionar unas entidades con otras a través de lo que se conoce como *relación*. Por ejemplo, dos entidades alumno y asignatura podrían estar relacionadas entre sí puesto que un alumno *cursa* una asignatura (o varias). Conviene resaltar que una relación entre dos entidades no expresa obligatoriedad de relación sino posibilidad de relacionarse.

En este caso, no será necesario que todos los alumnos cursen una asignatura o que una asignatura sea cursada por todos los alumnos para que la relación se establezca. Por tanto, en este caso se establece que entre esas dos entidades existe una relación a la que podríamos llamar *cursa*. Las relaciones se representan por el siguiente símbolo:

Si consideramos que dos entidades A y B están relacionadas a través de una relación R, deberemos determinar lo que se conoce como *cardinalidad de la relación*, que determina cuantas entidades de tipo A se relacionan, como máximo, con cuantas entidades de tipo B. Además, resulta conveniente, en cada caso, calcular cuántas entidades de tipo A se relacionan, como mínimo, con cuantas entidades de tipo B (que normalmente será 0 ó 1). De esa manera podremos indicar la obligatoriedad o no de relación entre elementos de las entidades A y B.



Relación uno a uno

En esta relación una entidad de tipo A sólo se puede relacionar con una entidad de tipo B, y viceversa. Por ejemplo, si suponemos dos entidades *Curso* y *Aula*, relacionadas a través de una relación *Se Imparte*, podremos suponer que un *Curso* se imparte en una *Aula* y en una *Aula* sólo se puede impartir un *Curso*. Se representaría como sigue:

Relación uno a muchos

Indica que una entidad de tipo A se puede relacionar con un número indeterminado de entidades de tipo B, pero a su vez una entidad de tipo B sólo puede relacionarse con una entidad de tipo A. Si suponemos una entidad *Propietario* y otra entidad *Vehículo* relacionadas a través de una relación *Posee*, podremos suponer que un *Propietario* puede poseer varios *Vehículos*, mientras que cada Vehículo sólo puede pertenecer a un *Propietario*.

Quedaría representado de la siguiente manera:

Relación muchos a uno

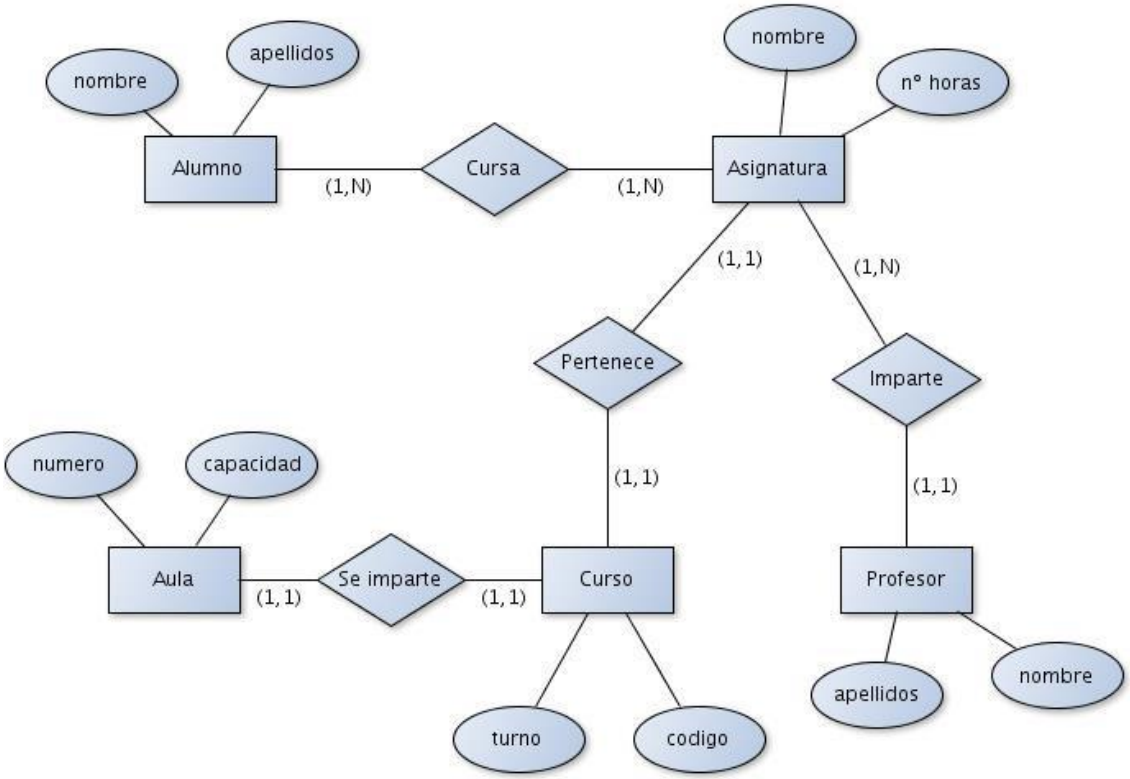
Significa que una entidad de tipo A sólo puede relacionarse con una entidad de tipo B, pero una entidad de tipo B puede relacionarse con un número indeterminado de entidades de tipo A. En realidad, se trata como una relación uno a muchos, pero el sentido de la relación es el inverso.

Relación muchos a muchos

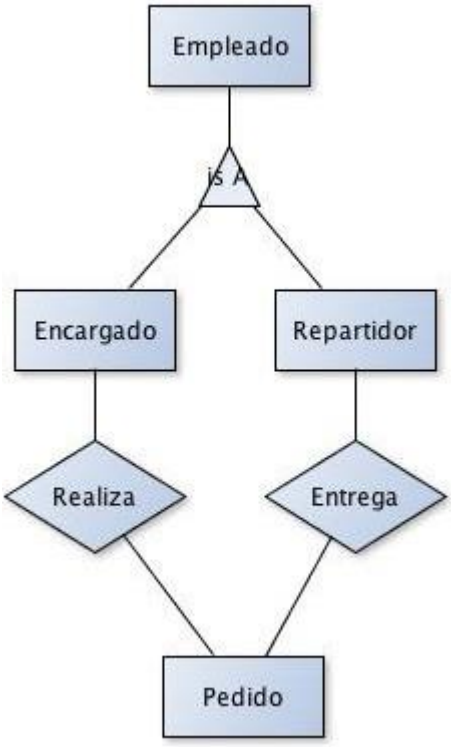
En este caso, tanto las entidades de tipo A y B, pueden relacionarse con un número indeterminado de entidades del otro tipo. Por ejemplo, si suponemos las entidades *Alumno* y *Asignatura* y una relación *Cursa*, podremos suponer que un *Alumno* cursa varias asignaturas mientras que una *Asignatura* la cursan varios *Alumnos*. Quedaría representado de la siguiente manera:

Diagrama Entidad/Relación

Se conoce como Diagrama Entidad/Relación (E/R) al diagrama resultante de modelar un mundo real siguiendo el modelo Entidad/Relación. Como resultado, se modelan todas las entidades con sus atributos, así como todas las relaciones existentes entre ellas, junto con su cardinalidad.


nombre
apellidos
nombre
n° horas
Alumno
(1,N)
Cursa
(1,N)
Asignatura
(1, 1)
(1,N)
Pertenece
Imparte
numero
capacidad
(1, 1)
(1, 1)
Aula
(1, 1)
Se imparte
(1, 1)
Curso
Profesor
apellidos
nombre
turno
codigo

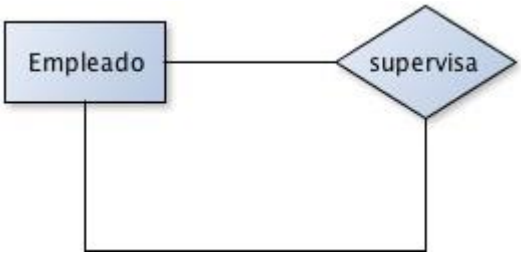
Herencia



También es posible representar otro tipo de relaciones entre objetos de nuestro sistema. La relación de herencia, representada como un triángulo (ver figura), expresa que un objeto es un subtipo de otro objeto. También se suele considerar al subtipo como una especialización del primero o al primero como una generalización del segundo.

En el caso del ejemplo, existen dos tipos de empleados que se relacionan de forma diferente con otros objetos del sistema, pero que a su vez pueden tener gran parte en común. Por ejemplo, trabajan de forma diferente, pero muchos de los datos personales que almacenaremos de ambos son comunes. Es por eso que el objeto *Empleado* se puede considerar una generalización de los dos tipos de trabajadores que hay en el sistema. Todos aquellos atributos y relaciones que tengan en común se podrán representar como atributos y funcionalidad del objeto *Empleado* y los atributos y relaciones que tengan como trabajadores especializados serán representados en el correspondiente objeto.

Reflexividad

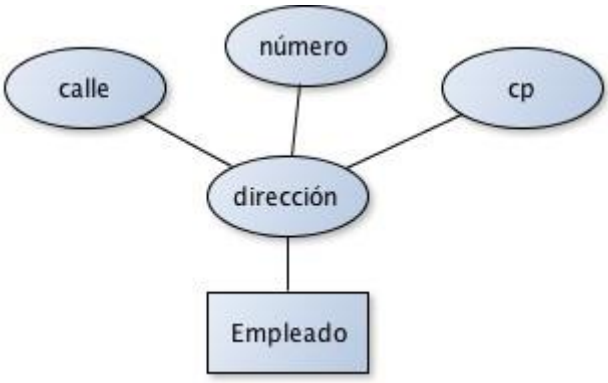


Es posible que la misma entidad ocupe ambos lados de una relación. En ese caso estamos frente a lo que se conoce como relaciones reflexivas. La cardinalidad de la relación indicará si todos los elementos de la relación están relacionados reflexivamente o bien sólo algunos están relacionados entre sí. En el caso de la figura podríamos suponer una empresa en la que algunos empleados hacen de supervisor de otros empleados.

Atributos multivaluados

Los atributos multivaluados son aquellos atributos que pueden contener una cantidad indeterminada de valores.

Atributos estructurados (o compuestos)



Los atributos estructurados o compuestos son aquellos atributos que pueden estar compuestos por otros atributos. Normalmente son atributos que pueden descomponerse, aunque dependiendo del contexto de la aplicación puede no interesar hacer esa descomposición y tratarlo como un atributo simple.

Atributos derivados

Los atributos derivados (o calculados) son aquellos atributos cuyo valor puede ser deducido realizando algunas operaciones con otros atributos de la misma entidad o de otras entidades. En algunas situaciones se podría considerar redundante (puesto que su valor se puede deducir) pero en otras puede resultar cómodo almacenarlo ya calculado puesto que se puede ahorrar mucho tiempo de cómputo si se trata de un valor de difícil y/o recurrente cálculo.

Comprobaciones sobre el Diagrama Entidad-Relación

1. Resulta cómodo que las entidades estén escritas en minúscula para hacer todas estas comprobaciones
2. Comprobar que nuestro diagrama no se ha convertido en un diagrama de flujo y no describe procesos, sino almacenes de datos
3. Comprobar que las Entidades son nombres de cosas y las relaciones son verbos
4. Comprobar que ninguna Entidad tiene como atributo algo que existe como Entidad (si ocurre, se deberían relacionar ambas Entidades)
5. Comprobar que varias entidades no comparten un mismo atributo estructurado que pueda ser considerado realmente como una Entidad
6. Evitar los ciclos (si aparece alguno, que puede ocurrir, comprobar que es necesario)

7. Si una relación tiene varios atributos, valorar si es posible que realmente deba ser una nueva Entidad (Comprar → Pedido, Alquilar → Alquiler, Reservar → Reserva, Enviar → Envío, . . .)
8. Comprobar que no hay colocada ninguna cardinalidad al revés: Se tiene que poder leer: un **Usuario Realiza** de 0 a N **Pedidos**. **Usuario** y **Pedido** son entidades y **Realizar** la relación entre ambas. En este caso, (0, N) debería estar escrito en el lado **Pedido** para que pudiera leerse correctamente
9. Continuará . . .

Ejemplos de diseño

- Diseñar un modelo Entidad/Relación (entidades, atributos y relaciones)

  Se quiere diseñar una Base de Datos para almacenar todos los datos de un campeonato de fútbol sala que se organiza este año en la ciudad. Aquellos que quieran participar deberán formar un equipo (nombre, patrocinador, color de la 1ª camiseta, color de la 2ª camiseta, categoría, . . .) e inscribirse en el campeonato. A medida que transcurran los partidos se irán almacenando los resultados de éstos, así como qué equipos lo jugaron, en qué campo se jugó, quién lo arbitró y alguna incidencia que pudiera haber ocurrido (en caso de que no ocurran incidencias no se anotará nada. Además, los participantes deberán rellenar una ficha de suscripción con algunos datos personales (nombre, apellidos, edad, dirección, teléfono, . . .)

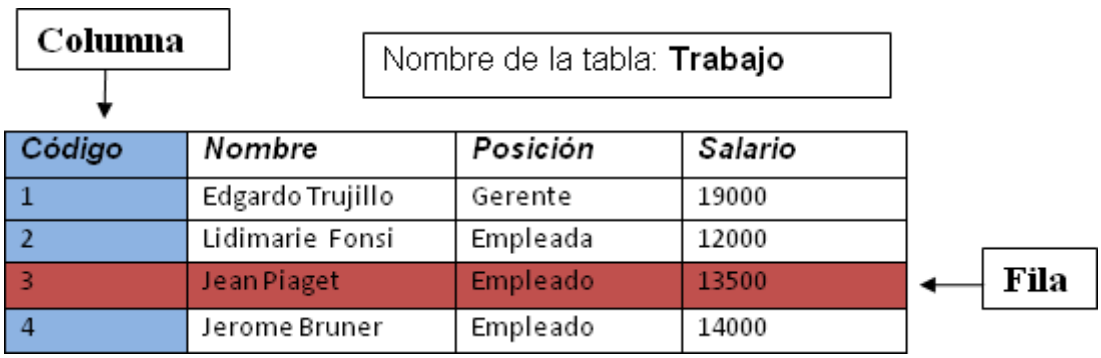
- Diseñar un modelo Entidad/Relación (ciclos y redundancia]

- Diseñar un modelo Entidad/Relación (atributos multivaluados, compuestos y derivados)

Modelo relacional

El modelo relacional es otro modelo de representación en el que los datos y sus relaciones se representan a través de tablas, y en el que los atributos se traducen en campos de esas tablas. Es el modelo de representación que siguen la gran mayoría de los SGBD relacionales (MySQL, SQL Server, Oracle, Ms Access, entre otros) en la actualidad, puesto que es el modelo de datos más extendido.

Así, es necesario transformar nuestro modelo Entidad/Relación a un modelo relacional si queremos crear nuestra Base de Datos en algún SGBD relacional.



Código	*Nombre*	*Posición*	*Salario*
1	Edgardo Trujillo	Gerente	19000
2	Lidimarie Fonsi	Empleada	12000
3	Jean Piaget	Empleado	13500
4	Jerome Bruner	Empleado	14000

CLIENTE

NOMBRE	CEDULA	CUENTA	CIUDAD
CANO	7.245.310	C-101	CALI
PEREZ	1.352.851	C-121	PASTO
TORO	9.874.115	C-203	BOGOTA
LOPEZ	9.765.398	C-302	BUGA
SERNA	2.458.698	C-109	TADO
VEGA	4.111.119	C-230	LIMA
CANO	7.245.310	C-209	CALI
PEREZ	1.352.851	C-209	PASTO

CUENTA

CUENTA	SALDO
C-101	50.000
C-121	120.000
C-203	70.000
C-302	98.000
C-209	42.000
C-109	108.500
C-230	59.000

Paso del modelo E/R al modelo relacional

El paso de un modelo E/R a un modelo relacional se puede llevar a cabo, en gran parte, siguiendo una serie de reglas o pautas, que se enumeran a continuación:

- Toda **entidad** se transforma en una tabla
- Todo **atributo simple** o **derivado** se transforma en columna de una tabla
- Los **atributos estructurados** transforman los campos en los que se componen en nuevas columnas de la tabla
- El **identificador único** de la entidad se convierte en clave primaria
- Los **atributos multivaluados** generan una nueva tabla con tres columnas: un *id*, el *id* de la tabla de la que surgen propagado como *clave ajena* y el valor del campo multivaluado
- Toda **relación N:M** se transforma en una tabla que tendrá como clave primaria la concatenación de los atributos clave de las entidades que relaciona.
- En la transformación de **relaciones 1:N** existen dos posibilidades:
    - Transformarlo en una tabla. Se hace como si se tratara de una relación N:M. Es conveniente realizarlo así en el caso de que se prevea que en un futuro la relación puede transformarse en N:M y cuando la relación tiene atributos propios
    - Propagar la clave. Se propaga el atributo principal de la entidad que tiene de cardinalidad máxima 1 a la que tiene cardinalidad máxima N, haciendo desaparecer a la relación
- En la transformación de **relaciones 1:1** se tienen en cuenta las cardinalidades de las entidades que participan en ellas. Existen también dos soluciones:
    - Transformarlo en una tabla. Si las entidades poseen cardinalidades (0,1), la relación se convierte en una tabla
    - Propagar la clave. Si una de las entidades posee cardinalidad (0,1) y la otra (1,1), se propaga la clave de la entidad con cardinalidad (1,1) a la tabla resultante de la entidad de cardinalidad (0,1). Si ambas poseen cardinalidades (1,1), se puede propagar la clave de cualquiera de ellas a la tabla resultante de la otra
- Para los **atributos de las relaciones** existen dos casos:
    - Si la relación es 1:N, sus atributos se propagan a la tabla de lado N, junto con la clave del lado 1
    - Si la relación es N:M, sus atributos se transforman en columnas de la tabla generada por dicha relación

- Las relaciones de **herencia** se pueden transforman de varias maneras. La más sencilla será crear una tabla por entidad (de la que se hereda y las que heredan), cada entidad transforma sus atributos siguiendo las reglas anteriores, y al final, la tabla que resulta de la entidad base (de la que se hereda) propaga su clave como clave ajena en cada una de las tablas que resultan de las entidades que heredaban.

Las situaciones más particulares habrá que estudiarlas y aplicar algún *patrón de diseño* conocido, si lo hay, para generar el correspondiente modelo relacional. Estos casos no siempre se podrán reflejar en el correspondiente modelo Entidad-Relación puesto que algunos tienen que ver con exigencias técnicas o de tiempo, más que con el propio modelo de datos. Algunos casos particulares pueden ser:

- **Datos temporales**: Cómo representar en un modelo relacional datos que tienen *fecha de caducidad* (los precios de un producto del que queremos tener un histórico de precios, . . .)
- **Datos eliminados**: Cómo representar en un modelo relacional datos que se desean eliminar, pero que por alguna razón necesitamos que sigan estando almacenados en la base de datos (productos descatalogados, alumnos que terminan sus estudios, . . .)
- **Registro o auditoría**: Cómo podemos registrar las acciones de los usuarios o de la aplicación durante el ciclo de vida de la Base de Datos.
- **Bloqueo de registros**: Cómo podemos bloquear el acceso a un registro para evitar la modificación simultánea del mismo dato por más de un usuario desde la aplicación que conecta con la Base de Datos.

En cualquier caso, aplicar correctamente al modelo relacional resultante las reglas de normalización eliminará todas las anomalías que nuestro modelo pueda contener. Así, hay que tener en cuenta que el modelo relacional que hemos obtenido en este momento todavía puede no ser el definitivo y puede sufrir transformaciones (e incluso se pueden añadir nuevas tablas) como resultado de aplicar las reglas de normalización que se pasan a explicar en el siguiente punto.

Ejemplos de transformaciones a modelo relacional

- Transformar relaciones 1 a 1 al modelo relacional
- Transformar relaciones N a M al modelo relacional
- Transformar relaciones 1 a N al modelo relacional
- Transformar una herencia al modelo relacional
- Transformar relaciones reflexivas al modelo relacional
- Transformar atributos al modelo relacional
- Ejemplo completo de diseño y transformación de modelo Entidad/Relación a modelo relacional

Normalización de modelos relacionales

Una vez obtenido el esquema relacional resultante del esquema entidad/relación que representa la base de datos, normalmente tendremos una buena base de datos. Pero otras veces, debido a fallos en el diseño o a problemas indetectables, tendremos un esquema que puede producir una base de datos que incorpore estos problemas:

Redundancia. Se llama así a los datos que se repiten continua e innecesariamente por las tablas de las bases de datos. Cuando es excesiva es evidente que el diseño hay que revisarlo, es el primer síntoma de problemas y se detecta fácilmente.

Ambigüedades. Datos que no clarifican suficientemente el elemento al que representan. Los datos de cada fila podrían referirse a más de un ejemplar de esa tabla o incluso puede ser imposible saber a qué ejemplar exactamente se están refiriendo. Es un problema muy grave y difícil de detectar.

Pérdida de restricciones de integridad. Normalmente debido a **dependencias funcionales.** Más adelante se explica este problema. Se arreglan fácilmente siguiendo una serie de pasos concretos.

Anomalías en operaciones de modificación de datos. El hecho de que al insertar un solo elemento haya que repetir tuplas en una tabla para variar unos pocos datos. O que eliminar un elemento suponga eliminar varias tuplas necesariamente (por ejemplo, eliminar un cliente suponga borrar seis o siete filas de la tabla de clientes, sería un error muy grave y por lo tanto un diseño terrible).

El principio fundamental reside en que las tablas deben referirse a objetos o situaciones muy concretas, relacionados exactamente con elementos reconocibles por el sistema de información de forma inequívoca. Cada fila de una tabla representa inequívocamente un elemento reconocible en el sistema. Lo que ocurre es que conceptualmente es difícil agrupar esos elementos correctamente.

En cualquier caso, la mayor parte de problemas se agravan si no se sigue un modelo conceptual y se decide crear directamente el esquema relacional. En ese caso, el diseño tiene una garantía casi asegurada de funcionar mal.

Cuando aparecen los problemas enumerados, entonces se les puede resolver usando reglas de normalización. Estas reglas suelen forzar la división de una tabla en dos o más tablas para arreglar ese problema.

Formas normales

Las formas normales se corresponden a una teoría de normalización iniciada por el propio **Codd** y continuada por otros autores (entre los que destacan **Boyce** y **Fagin**). Codd definió en 1970 la primera forma normal, desde ese momento aparecieron la segunda, tercera, la Boyce-Codd, la cuarta y la quinta forma normal.

Una tabla puede encontrarse en primera forma normal y no en segunda forma normal, pero no al contrario. Es decir, los números altos de formas normales son más restrictivos (la quinta forma normal cumple todas las anteriores).

La teoría de formas normales es una teoría absolutamente matemática, pero en el presente manual se describen de forma más intuitiva.

Hay que tener en cuenta que muchos diseñadores opinan que basta con llegar a la forma Boyce-Codd, ya que la cuarta, y sobre todo la quinta, forma normal es polémica. Hay quien opina que hay bases de datos peores en quinta forma normal que en tercera. En cualquier caso, debería ser obligatorio para cualquier diseñador llegar hasta la forma normal de Boyce-Codd.

primera forma normal (1FN)

Es una forma normal inherente al esquema relacional. Es decir, toda tabla realmente relacional la cumple.

Se dice que una tabla se encuentra en primera forma normal si impide que un atributo de una tupla pueda tomar más de un valor. La tabla:

TRABAJADOR		
DNI	**Nombre**	**Departamento**
12121212A	Andrés	Mantenimiento
12345345G	Andrea	Dirección Gestión

Visualmente es una tabla, pero no una tabla relacional (lo que en terminología de bases de datos relacionales se llama **relación**). No cumple la primera forma normal.

Sería primera forma normal si los datos fueran:

TRABAJADOR		
DNI	**Nombre**	**Departamento**
12121212A	Andrés	Mantenimiento
12345345G	Andrea	Dirección
12345345G	Andrea	Gestión

Esa tabla sí está en primera forma normal.

segunda forma normal (2FN)

Esta formal normal sólo debe ser considerada para aquellas tablas en las que la clave principal sea compuesta. Si no fuera así, la tabla estaría, de forma directa, en segunda forma normal.

Decimos que una tabla está en *segunda forma normal* si se cumplen las siguientes condiciones:

- Está en 1FN
- Todo atributo secundario (que no pertenezca a la clave principal) tiene una dependencia funcional total de la clave principal, y no de una parte de ella

Se dice un atributo B depende funcionalmente de A (A→B) si cada valor de A se corresponde con un único valor de B. Visto de otra manera, si dado A puedo obtener B. Un caso típico podría ser DNI → Nombre, puesto que dado un DNI puedo obtener, de forma unívoca, el nombre de la persona

Para convertir una tabla que no está en 2FN se creará una tabla con la clave y todas sus dependencias funcionales totales y otra tabla con la parte de la clave que tiene dependencias con los atributos secundarios.

En el ejemplo podemos ver como el campo *TelefonoProveedor* no depende totalmente de la clave *(NombreProducto, NombreProveedor)*, sino únicamente del campo *NombreProveedor*.

NombreProducto	NombreProveedor	Categoria	TeléfonoProveedor
Diarios	Exotic Kiosk	Prensa	968582222
Revistas	Exotic Kiosk	Prensa	968582222
Habas	Tnj export	Alimentacion	975869999
Botes	Tnj export	Bebidas	975869999

IdProducto	NombreProduct	Categoria	IdProveedor
1	Diarios	Prensa	1
2	Revistas	Prensa	1
3	Habas	Alimentación	2
4	Botes	Alimentación	2

IdProveedor	NombreProveedor	TeléfonoProveedor
1	Exotic Kiosk	968582222
2	Tnj Export	975869999

tercera forma normal (3FN)

Se dice que una tabla está en *tercera forma normal* si:

- Está en 2FN
- No existen atributos no primarios (que no pertenezcan a la clave) que son transitivamente dependientes de cada posible clave de la tabla. Es decir, un atributo secundario sólo puede ser conocido a través de la clave principal y no por medio de un atributo no primario.

Para convertir una tabla que no está en 3FN se realizará una proyección de la clave a los elementos que no tengan dependencia funcional transitiva y otra tabla con una nueva clave a los elementos que anteriormente tenían esta dependencia.

En el ejemplo, es posible conocer la edad del inscrito a través del número de licencia, y dada la edad podemos conocer su categoría, tenemos una dependencia funcional transitiva. Lo importante es tener en cuenta que la categoría depende de un atributo que no forma parte de la clave. Para normalizar, debemos descomponer esa tabla en las tablas *Atletas* y *Categorías*

Licencia	Nombre	Club	Edad	Categoria
189585	Pepe	Alcoy	12	Junior
205888	Tomas	Jaca	10	Cadete
748523	Andres	Almansa	9	Alevín

Licencia	Nombre	Club	Edad
189585	Pepe	Alcoy	12
205888	Tomas	Jaca	10
748523	Andres	Almansa	9

Edad	Categoria
9	Alevín
10	Cadete
12	Junior

forma normal de Boyce-Codd (FNBC o BCFN)

Ocurre si una tabla está en tercera forma normal y además todo determinante es una clave candidata. Ejemplo:

ORGANIZACIÓN		
Trabajador	Departamento	Responsable
Alex	Producción	Felipa
Arturo	Producción	Martín
Carlos	Ventas	Julio

Carlos	Producción	Felipa
Gabriela	Producción	Higinio
Luisa	Ventas	Eva
Luisa	Producción	Martín
Manuela	Ventas	Julio
Pedro	Ventas	Eva

La cuestión es que un trabajador o trabajadora puede trabajar en varios departamentos. En dicho departamento hay varios responsables, pero cada trabajador sólo tiene asignado uno. El detalle importante que no se ha tenido en cuenta, es que elo la responsable sólo puede ser responsable en un departamento.

Este detalle último produce una dependencia funcional ya que:

Responsable→Departamento

Por lo tanto, hemos encontrado un determinante que no es clave candidata. No está por tanto en FNBC. En este caso la redundancia ocurre por mala selección de clave. La redundancia del departamento es completamente evitable. La solución sería:

PERSONAL	
Trabajador	**Responsable**
Alex	Felipa
Arturo	Martín
Carlos	Julio
Carlos	Felipa
Gabriela	Higinio
Luisa	Eva
Luisa	Martín
Manuela	Julio
Pedro	Eva

RESPONSABLES	
Responsables	Departamento
Felipa	Producción
Martín	Producción
Julio	Ventas
Higinio	Producción
Eva	Ventas

En las formas de Boyce-Codd hay que tener cuidado al descomponer ya que se podría perder información por una mala descomposición

cuarta forma normal (4FN). dependenciasmultivaluadas

dependencia multivaluada

Para el resto de formas normales (las diseñadas por **Fagin**, mucho más complejas), es importante definir este tipo de dependencia, que es distinta de las funcionales. Si las funcionales eran la base de la segunda y tercera forma normal (y de la de Boyce-Codd), éstas son la base de la cuarta forma normal.

Una dependencia multivaluada de X sobre Y (es decir X->>Y), siendo X e Y atributos de la misma tabla, ocurre cuando Y tiene un conjunto de valores bien definidos sobre cualquier valor de X. Es decir, dado X sabremos los posibles valores que puede tomar Y.

Se refiere a posibles valores (en plural) y se trata de que los valores de ese atributo siempre son los mismos según el valor de un atributo y no del otro.

Ejemplo:

Nº Curso	Profesor	Material
17	Eva	1
17	Eva	2
17	Julia	1
17	Julia	2
25	Eva	1
25	Eva	2
25	Eva	3

La tabla cursos, profesores y materiales del curso. La tabla está en FNBC ya que no hay dependencias transitivas y todos los atributos son clave sin dependencia funcional hacia ellos. Sin embargo, hay redundancia. Los materiales se van a repetir para cualquier profesor dando cualquier curso, ya que los profesores van a utilizar todos los materiales del curso (de no ser así no habría ninguna redundancia).

Los materiales del curso dependen de forma multivaluada del curso y no del profesor en una dependencia multivaluada (no hay dependencia funcional ya que los posibles valores son varios). Para el par *Nº de curso* y *Profesor* podemos saber los materiales; pero lo sabemos por el curso y no por el profesor.

cuarta forma normal

Ocurre esta forma normal cuando una tabla está en forma normal de Boyce Codd y toda dependencia multivaluada no trivial es una dependencia funcional. Son triviales aquellas dependencias multivaluadas en las que el conjunto formado por eldeterminante y el implicado no forman la clave primaria de la tabla y además el implicado no forma parte del determinante: es decir si $X \twoheadrightarrow Y$ y además $Y \not\subset X$ y X,Y no es la clave de la tabla, tenemos una dependencia multivaluada no trivial (como ocurre en el ejemplo anterior).

Para la tabla anterior la solución serían dos tablas:

Nº Curso	Material
17	1
17	2
25	1
25	2
25	3

Nº Curso	Profesor
17	Eva
17	Julia
25	Eva

Un teorema de Fagin indica cuando hay tres pares de conjuntos de atributos X, Y y Z si ocurre X->>Y y X->>Z (Y y Z tienen dependencia multivaluada sobre X), entonces las tablas X, Y y X, Z reproducen sin perder información lo que poseía la tabla original. Este teorema marca la forma de dividir las tablas hacia una 4FN

quinta forma normal (5FN)

dependencias de JOIN o de reunión

Una proyección de una tabla es la tabla resultante de tomar un subconjunto de los atributos de una tabla (se trata de la operación proyección, Π, del álgebra relacional). Es decir una tabla formada por unas cuantas columnas de la tabla original.

La operación JOIN procedente también del álgebra relacional, consiste en formar una tabla con la unión de dos tablas. La tabla resultante estará formada por la combinación de todas las columnas y filas de ambas, excepto las columnas y filas repetidas.

Se dice que se tiene una tabla con dependencia de unión (o de tipo JOIN) si se puede obtener esa tabla como resultado de combinar mediante la operación JOIN varias proyecciones de la misma.

quinta forma normal o forma normal de proyección-unión

Ocurre cuando una tabla está en 4FN y cada dependencia de unión (JOIN) en ella es implicada por las claves candidatas.

Es la más compleja y polémica de todas. Polémica pues no está claro en muchas ocasiones está muy claro que el paso a 5FN mejore la base de datos. Fue definida también por Fagin.

Es raro encontrarse este tipo de problemas cuando la normalización llega a 4FN. Se deben a restricciones semánticas especiales aplicadas sobre la tabla. Ejemplo:

Proveedor	Material	Proyecto
1	1	2
1	2	1
2	1	1
1	1	1

Indican códigos de material suministrado por un proveedor y utilizado en un determinado proyecto. Así vista la tabla, no permite ninguna proyección en la que no perdamos datos.

Pero si ocurre una restricción especial como por ejemplo: *Cuando un proveedor nosha suministrado alguna vez un determinado material, si ese material aparece en otro proyecto, haremos que el proveedor anterior nos suministre también ese material para el proyecto.*

Eso ocurre en los datos como el proveedor número 1 nos suministró el material número 1 para el proyecto 2 y en el proyecto 1 utilizamos el material 1, aparecerá la tupla proveedor 1, material 1 y proyecto 1. Si un nuevo proyecto necesitara el material 1, entonces habrá que pedirlo a los proveedores 1 y 2 (ya que en otros proyectos les henos utilizado)

La dependencia de reunión que produce esta restricción es muy difícil de ver ya que es lejana. Para esa restricción esta proyección de tablas sería válida:

Proveedor	Material
1	1
1	2
2	1

Material	Proyecto
1	2
2	1
1	1

Proveedor	Proyecto
1	2
1	1
2	1

Esa descomposición no pierde valores en este caso, sabiendo que si el proveedor nos suministra un material podremos relacionarle con todos los proyectos que utilizan ese material.

Resumiendo, una tabla no está en quinta forma normal si hay una descomposición de esa tabla que muestre la misma información que la original y esa descomposición no tenga como clave la clave original de la tabla.

forma normal de dominio clave (FNDC)

Se la conoce más con sus siglas en inglés **DKNF**. Se trata de una forma normal enunciada también por **Fagin** en 1981 al darse cuenta de los problemas de redundancia que ocurrían con algunos dominios.

En este caso no se basó en dependencias entre los datos, sino que se basó en *restricciones de dominio y restricciones de clave*.

- **Restricciones de dominio**. Se trata de la restricción que hace que un determinado atributo obtenga sólo ciertos valores, los que estén de acuerdo a la definición de un determinado dominio.
- **Restricción de clave**. Es la restricción que permite que un atributo o un conjunto de atributos forme una clave candidata.

Fagin dice que una tabla está en FNDC si toda restricción sobre la tabla es consecuencia lógica de aplicar las restricciones de dominio y clave sobre la misma. Fagin demostró que si esto ocurría la tabla incluso estaba en 5FN.

Ejemplo:

Alumno	Nivel logrado	Nota
Antonio	Muy alto y eficiente	9
Marisa	Muy alto aunque sin trabajo constante	9
Mario	Alto con trabajo habitual	7
Luisa	Medio aunque con trabajo	5

Observando los datos de la tabla se observa que cuando la nota es superior a 9, en el nivel aparece la palabra alto, cuando es un 7 o un 8 medio, y un 5 o 6 sería medio. Es decir, tenemos restricciones que no son ni de dominio ni de clave en esa tabla. Lo lógico sería:

Alumno	Nivel de trabajo	Nota
Antonio	Eficiente	9
Marisa	Trabajo constante	9
Mario	Trabajo habitual	7
Luisa	Trabajo medio	5

Nivel académico	Nota mínima	Nota máxima
Muy alto	9	10
Alto	7	8,99
Medio	5	6,99
Bajo	0	4,99

No se pierde información al diseñar las tablas de esta forma y de hecho es más eficiente para la base de datos.