

# BASES DE DATOS - A

DESARROLLO DE APlicaciones multiplataforma

DESARROLLO DE APlicaciones web

ILERNA

© Ilerna Online S.L., 2023

Maquetado e impreso por Ilerna Online S. L.

© Imágenes: Shutterstock

Impreso en España - Printed in Spain

Reservados todos los derechos. No se permite la reproducción total o parcial de esta obra, ni su incorporación a un sistema informático, ni su transmisión en cualquier forma o por cualquier medio (electrónico, mecánico, fotocopia, grabación u otros) sin autorización previa y por escrito de los titulares del copyright. La infracción de dichos derechos puede constituir un delito contra la propiedad intelectual.

**Ilerna Online S. L.** ha puesto todos los recursos necesarios para reconocer los derechos de terceros en esta obra y se excusa con antelación por posibles errores u omisiones y queda a disposición de corregirlos en posteriores ediciones.

2.<sup>a</sup> edición: marzo 2023

# ÍNDICE

## Bases de datos - A

<b>1. Introducción a las bases de datos (BBDD).....</b>	<b>6</b>
1.1. Evolución histórica de las BBDD .....	8
1.2. Ventajas e inconvenientes de las BBDD.....	11
1.3. Almacenamiento de la información.....	12
1.4. Sistemas Gestores de bases de datos (SGBD) .....	17
1.5. ANSI/X3/SPARC. Estándares y niveles .....	23
1.6. Modelos de BBDD. Tipos: jerárquico, red, relacional y orientado a objetos .....	25
1.7. Bases de datos centralizadas y distribuidas .....	29
<b>2. Modelo entidad-relación .....</b>	<b>36</b>
2.1. Concepto de modelo entidad-relación.....	38
2.2. Entidad: representación gráfica, atributos y tipos de claves....	39
2.3. Relación: representación gráfica, atributos, grado y cardinalidad .....	42
2.4. Diagrama entidad-relación .....	43
2.5. Modelo entidad-relación extendido.....	49
2.6. Guía para la construcción de un modelo entidad-relación .....	57
<b>3. Modelo relacional .....</b>	<b>58</b>
3.1. Terminología del modelo relacional .....	59
3.2. Concepto de relación. Propiedades y relaciones.....	60
3.3. Atributos y dominio de los atributos .....	60
3.4. Concepto y tipos de clave: candidatas, primarias, ajenas, alternativas.....	63
3.5. Valores nulos .....	66
3.6. Reglas de integridad: de entidad y referencial .....	66
3.7. Traducción del modelo entidad-relación al modelo relacional	68
<b>4. Normalización .....</b>	<b>74</b>
4.1. Concepto de normalización, dependencias funcionales y sus tipos .....	75
4.2. Las formas normales .....	78
4.3. Primera forma normal (1FN) .....	78
4.4. Segunda forma normal (2FN).....	81
4.5. Tercera forma normal (3FN) .....	82

4.6.	Forma normal Boyce-Codd (FNBC).....	83
4.7.	Otras formas normales (4FN, 5FN).....	83
4.8.	Desnormalización .....	83
<b>5.</b>	<b>Lenguajes de las BBDD. SQL .....</b>	<b>84</b>
5.1.	Tipos de lenguajes para gestionar los datos en un SGBDR corporativo .....	86
5.2.	Herramientas gráficas proporcionadas por el SGBD para la edición de la BD .....	87
5.3.	Lenguaje de definición de datos (DDL) .....	104
5.4.	Lenguaje de manipulación de datos (DML) .....	114
5.5.	Extensiones y otras cláusulas del lenguaje.....	125
5.6.	Herramientas de la BDD para optimizar consultas .....	127
<b>6.</b>	<b>Estrategias para el control de las transacciones y de la concurrencia .....</b>	<b>128</b>
6.1.	Concepto de integridad.....	129
6.2.	Concepto de transacción. Control.....	131
6.3.	Propiedades de las transacciones: atomicidad, consistencia, aislamiento y permanencia .....	132
6.4.	Estados de una transacción: activa, parcialmente comprometida, fallida, abortada y comprometida .....	133
6.5.	Sentencias de transacciones .....	134
6.6.	Problemas derivados de la ejecución concurrente de transacciones .....	135
6.7.	Control de concurrencia: técnicas optimistas y pesimistas ....	136
6.8.	Recuperación ante errores. Mecanismos para deshacer transacciones .....	138
<b>7.</b>	<b>Lenguajes de las BBDD para la creación de su estructura.....</b>	<b>140</b>
7.1.	Vistas y otras extensiones del lenguaje .....	141
	<b>Bibliografía / webgrafía .....</b>	<b>146</b>
	<b>Solucionario .....</b>	<b>147</b>



# 1

## INTRODUCCIÓN A LAS BASES DE DATOS (BBDD)



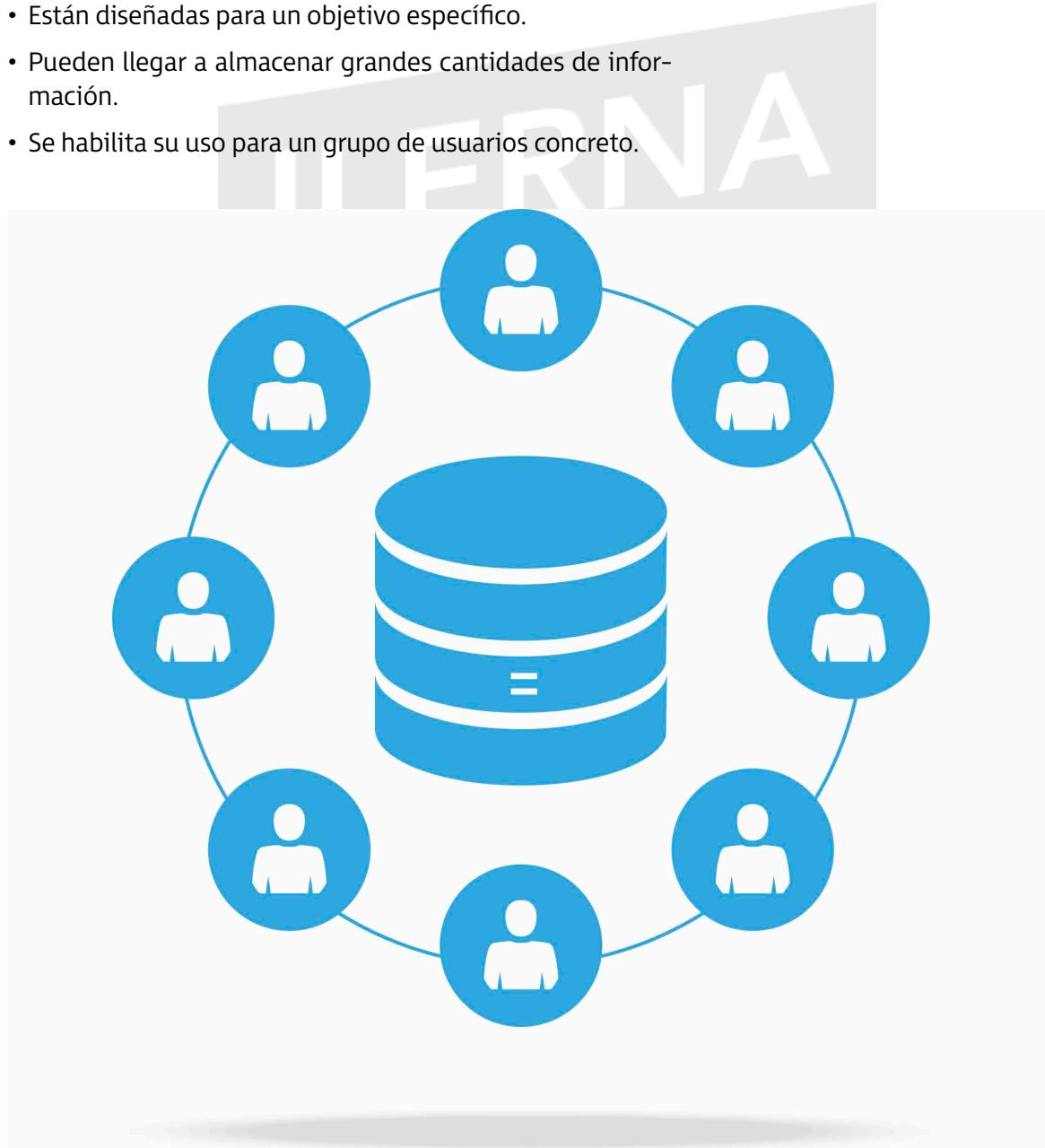
Las bases de datos (BBDD) son una parte esencial e imprescindible en el ámbito de los sistemas informáticos. Existen diversas perspectivas y definiciones sobre las BBDD, pero nos vamos a centrar en aquellas relativas a la informática.

### CONCEPTO

Una **BBDD** se puede definir como una agrupación de datos relacionados entre sí, almacenados en formato digital, con una organización o estructura determinada, de modo que se permite el acceso a dicha información con cierta rapidez.

Entre las propiedades más importantes de las BBDD podemos destacar las siguientes:

- Representan elementos abstractos o concretos del mundo real.
- Están diseñadas para un objetivo específico.
- Pueden llegar a almacenar grandes cantidades de información.
- Se habilita su uso para un grupo de usuarios concreto.



## 1.1. EVOLUCIÓN HISTÓRICA DE LAS BBDD

Las BBDD como concepto genérico, es decir, como idea de un conjunto de datos o registros recopilados, organizados de algún modo y accesibles para un uso posterior, está directamente relacionado con el origen de la historia humana, dado que dicho origen se establece tras la invención de la escritura.

Se ha descubierto que muchos de los primeros escritos que se han hallado y estudiado, como por ejemplo los de la antigua civilización sumeria, servían como registro de las transacciones comerciales o como inventario. Por ejemplo, de la cantidad de arroz recogido tras una cosecha o de animales de ganadería que pertenecían a una persona tras una transacción comercial.

Toda esta información quedaba registrada de manera clara y organizada para poder realizar posteriores consultas en caso de necesitarse.

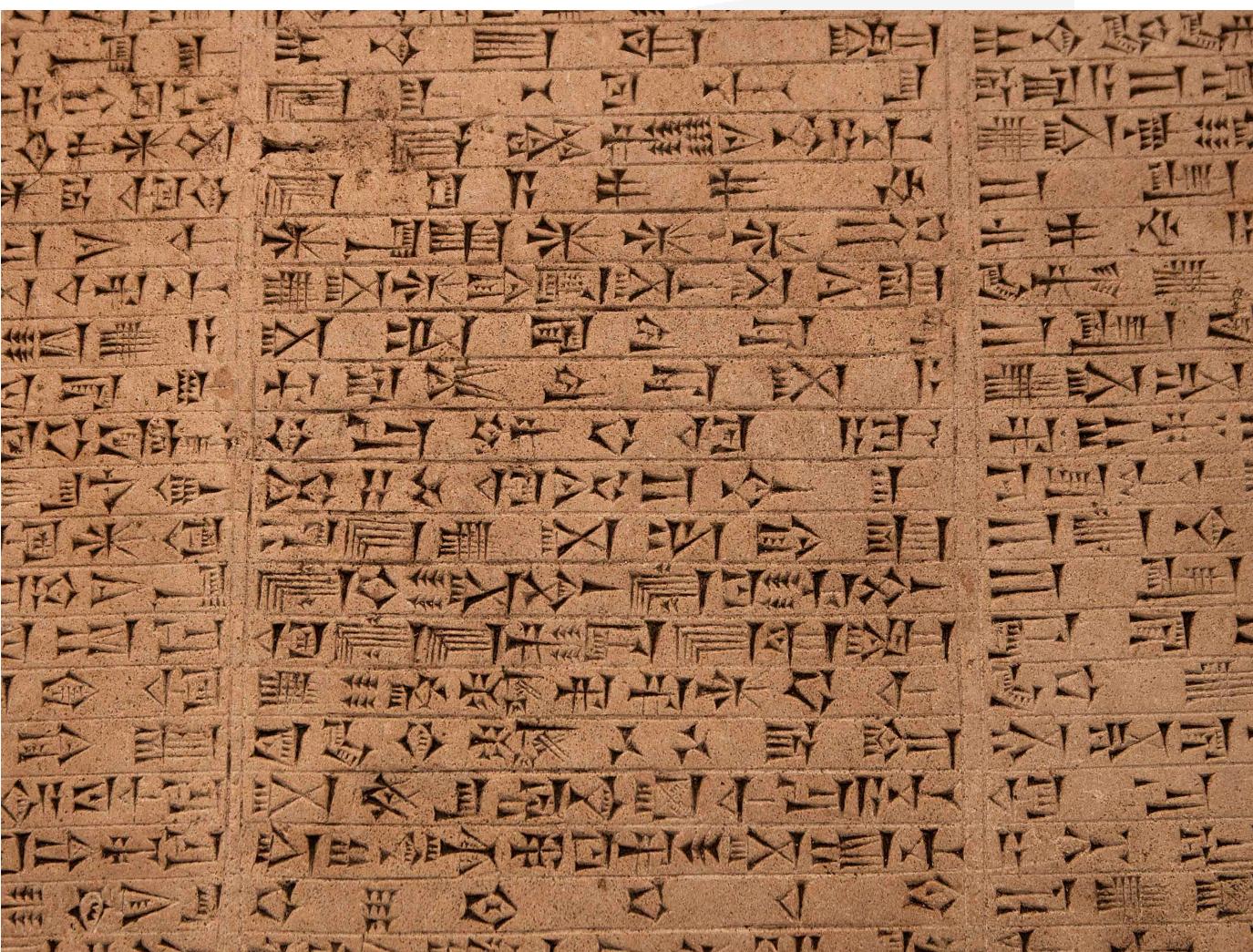


Imagen de una antigua tablilla cuneiforme, uno de los primeros sistemas de escritura.

Con el avance de la historia, este tipo de información fue registrándose cada vez con mejores tecnologías: se pasó del uso de tablillas de arcilla al uso de ordenadores digitales, pasando por tecnologías intermedias, como el papel y la tinta.

En este apartado, nos vamos a centrar en la evolución histórica de las BBDD desde del siglo XX. Es decir, desde la aparición de la informática y la digitalización de la información.

### 1900 – 1950

Se dan los primeros pasos de la informática con la invención del tubo de vacío, la construcción de la primera puerta lógica AND, la máquina de Turing, la primera máquina electromecánica (Harvard Mark I) o el diseño de la arquitectura de von Neumann.

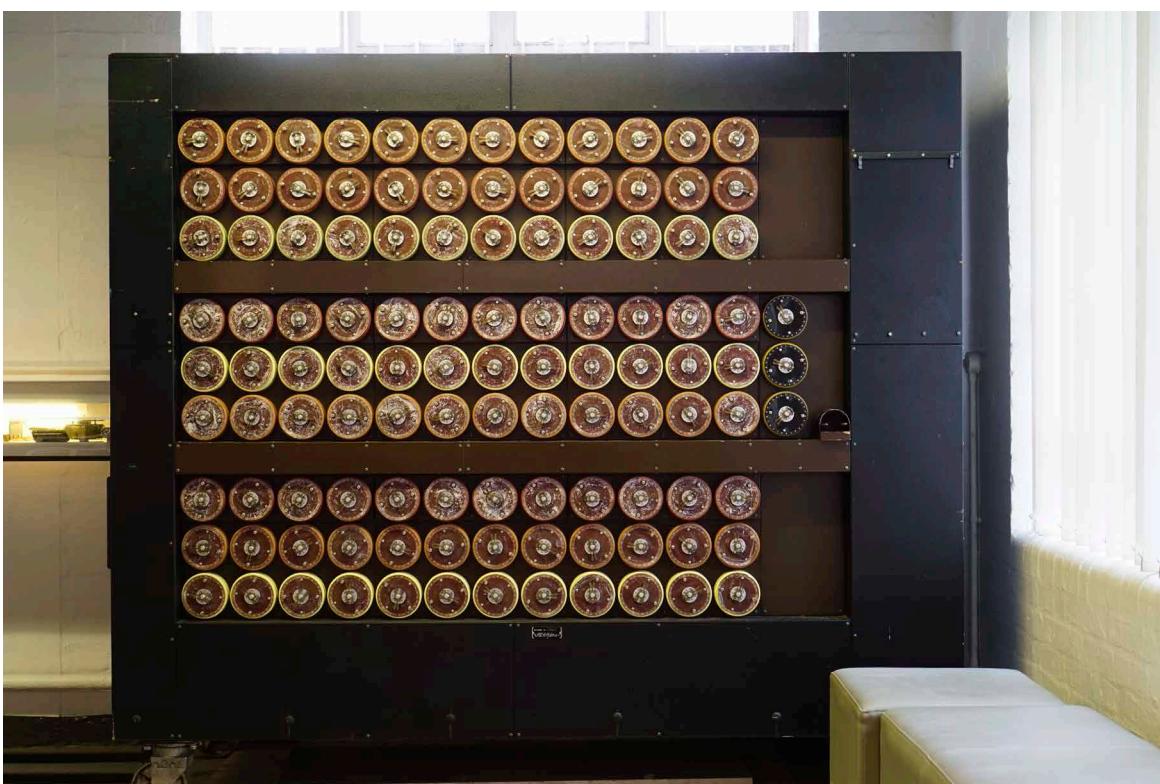


Imagen actual de la Máquina de Turing.

### Años 60

Aparecen sistemas por lotes (**batchprocessing**), que eran ejecuciones de programas en los que no era necesario que el usuario tuviese que estar controlando la máquina continuamente. Comienza la incipiente comercialización de la informática y surge *DatabaseTaskGroup* dentro de CODASYL. Por otro lado, IBM y American Airlines crean **SABRE**, un sistema de control de pasajeros y vuelos de avión en Estados Unidos.

### Años 70

Edgar Frank Codd crea y publica el modelo relacional de bases de datos. A raíz de esa publicación, nace el sistema de BBDD de **ORACLE**. Aparece el modelo entidad-relación, que facilita el diseño de las BBDD.

### Años 80

Fue a partir de los 80 cuando empezó a extenderse la informática y, con ella, las BBDD. Anteriormente, las aplicaciones que gestionaban las BBDD estaban diseñadas para ser gestionadas por personal muy cualificado. Para solventar este problema surgió la **estandarización**, con un nuevo lenguaje de programación denominado **SQL**. Este lenguaje produjo un estímulo en el ámbito de las BBDD relacionales.

Fue entonces cuando aparecieron los primeros ordenadores personales, capaces de manejar sistemas de gestión de BBDD a nivel de usuario.



*StructureQueryLanguage aparece en los años 80.*

### Años 90

A principios de los noventa, nos encontrábamos con una distribución bastante amplia de utilización de BBDD en todas las empresas, por lo que esta década se caracteriza las mejoras en el rendimiento de los sistemas gestores.

Microsoft desarrolla el popular sistema gestor de BBDD: **Microsoft Access**.

Los Sistemas de Gestión de bases de datos (SGBD) de la época tenían que servir, básicamente, para gestionar las distintas BBDD que utiliza cada usuario en un ordenador personal.

## Años 2000 y en adelante

Tres grandes empresas dominan el mercado de los sistemas gestores de BBDD: **ORACLE, IBM y Microsoft**.

Debido a los límites que presentaban las BBDD en épocas anteriores, nace la necesidad de incluir tecnologías con imagen y sonido a nuestro almacén de información. Por esta razón, los SGBD incluyeron el **tipo abstracto de datos** (TAD). Este tipo de dato no solamente comprende valores, sino también operaciones y propiedades.

Esta nueva forma de entender las BBDD nos lleva a la implementación de la **orientación a objetos** (OO). El éxito del paradigma de la **programación orientada a objetos (POO)** y la inclusión de las BBDD en las páginas web, hacen que los SGBD relacionales estén en plena transformación para adaptarse a estas nuevas tecnologías de reciente éxito y fuertemente relacionadas.

## 1.2. VENTAJAS E INCONVENIENTES DE LAS BBDD

### Ventajas

- La información contenida en la BBDD se procesa de forma independiente. Gracias a un buen diseño de relación entre los datos, conseguimos la menor redundancia posible.
- Mediante las BBDD podemos llegar a conseguir la unidad máxima de información partiendo de una pequeña cantidad de datos.
- Aseguran la integridad de los datos y devuelven unos resultados coherentes.
- Posibilitan una gran seguridad para la información.
- Reducen el espacio para el almacenamiento de la información comparada con otros medios como el papel físico.
- Se obtiene una forma muy eficiente de almacenar los datos.

### Desventajas

- Requieren de un mantenimiento por parte de un administrador cualificado.
- Requieren la adquisición y configuración de un *hardware*, lo cual resulta más costoso que otras alternativas como la información en papel. Por tanto, a corto plazo no ofrece rentabilidad.
- Necesita un espacio considerable en la memoria de la computadora, además del espacio requerido en el disco duro.

**ponte a prueba**

**¿A partir de qué año es posible gestionar los sistemas de gestión de bases de datos a nivel de usuario?**

- a) A partir de los años 80, con la aparición del sistema de base de datos de Oracle.
- b) A partir de los años 80, con la aparición del lenguaje SQL.
- c) A partir de los años 90, con la aparición del sistema gestor Microsoft Access.
- d) A partir de los años 2000, cuando aparecen los SGDB.

**¿Qué sistemas aparecen en los años 60?**

- a) Los sistemas de puerta lógica AND.
- b) Los sistemas Batchprocessing.
- c) Los sistemas SGBD.
- d) Los sistemas de estandarización.



## 1.3. ALMACENAMIENTO DE LA INFORMACIÓN

Las BBDD son hoy la técnica más eficiente para guardar información de todo tipo de una manera estructurada y accesible. En la sociedad actual, las BBDD están presentes en múltiples ámbitos. Por ejemplo, en Educación, se puede mantener un registro completo de todos los alumnos con sus respectivas calificaciones. En el de la Sanidad se usan BBDD para almacenar toda la información de los pacientes, junto con sus historiales médicos. La Banca la usa, por ejemplo, para guardar información de sus clientes y sus cuentas bancarias. Incluso solo en un teléfono móvil se pueden llegar a encontrar distintas BBDD para diversas aplicaciones.

En el ámbito de la informática, para almacenar información se utiliza un elemento clave llamado **fichero**. Los ficheros, también denominados archivos, son una secuencia de dígitos binarios que, siguiendo una estructura lógica determinada, almacenan una información concreta, como, por ejemplo, un informe, una imagen, una BD o una canción musical.

### Tipos de ficheros

Según como sea el método de acceso a un fichero, podemos diferenciar entre los siguientes tipos:

- **Ficheros planos:** se caracterizan por tener un orden contiguo de registros, de modo que para acceder a un registro concreto se han de recorrer secuencialmente, desde el principio, todos los registros que existan previamente.

- **Ficheros indexados:** en estos ficheros la información se divide en dos tipos: el índice y los registros. En el índice se establece una tabla de claves que indican la posición física de cada uno de los registros, para así acceder a cada uno más rápidamente.
- **Ficheros de acceso directo:** en este tipo se prescinde del índice y el campo clave de un registro pasa a ser la dirección física. De este modo, el posicionamiento es inmediato y normalmente se necesita que los registros tengan un tamaño fijo.
- **Otros:** existen otros tipos de ficheros, como los que usan las funciones *Hash*, esto es, un algoritmo matemático que se utiliza para generar la relación clave-posición de cada registro.

Las **operaciones básicas** que se pueden realizar en un fichero son:

<b>Abrir (open)</b>	Prepara el fichero para su posterior operación de lectura y/o escritura.
<b>Cerrar (close)</b>	Cierra el fichero, por lo que no se puede trabajar más con él.
<b>Leer (read)</b>	Obtiene la información del fichero.
<b>Escribir (write)</b>	Guarda información en el fichero.
<b>Posicionarse (seek)</b>	Posiciona el puntero para su posterior operación de lectura y/o escritura.
<b>Fin de fichero (eof)</b>	Marca el final del fichero.

### Conceptos clave de las BBDD

Tal y como hemos definido previamente, una BD se puede entender como una **colección de datos que se relacionan entre sí, en formato digital, con una organización determinada y accesible**.

Se utiliza para mantener la información de diversos objetos de una forma coherente y jerarquizada, de tal manera que, con la mínima cantidad de datos almacenados podamos sacar el mayor número de resultados posible. Para tal fin, vamos a organizar la información dentro de la BBDD en unas estructuras llamadas **tablas**, definidas posteriormente, que deben estar perfectamente relacionadas entre ellas.

A continuación, se definen de forma más detallada algunos de los conceptos usados en BBDD:

- **Dato:** expresa una información concreta sobre algo específico. Por ejemplo, la edad de una persona es un dato, su fecha de nacimiento es otro dato, la calificación de un alumno en un examen, etc.
- **Tipo de dato:** es la estructura en la que se está expresado un dato concreto. Por ejemplo, un dato podría ser de tipo numérico si expresa una edad o podría ser de tipo fecha. Un dato también podría ser de tipo *booleano*, si lo que se desea es expresar un valor binario, es decir, verdadero o falso.
- **Campo:** es un identificador que agrupa los datos de una misma propiedad o atributo en común de todos los registros de una tabla. Por ejemplo, en la tabla *Personas*, cada registro es una persona concreta y un campo de esos registros podría ser la *fecha de nacimiento* o el *apellido*. En función de la información que indica el campo, este tendrá un tipo de dato u otro.
- **Tabla:** corresponde a una entidad formada por varios campos y registros. Por ejemplo, podríamos crear la tabla *Persona*, que estuviera formada por los campos *nombre*, *apellidos*, *edad*, *sexo*, *DNI*, etc., y cada registro sería una persona distinta.

*Las tablas son el principal elemento del modelo relacional.*



- **Registro:** un registro es una agrupación de datos que hacen referencia a un mismo elemento. Por ejemplo, en la tabla *Persona* un registro podría ser: *María, Gómez, 25 años, Mujer, 47585858X*.
- **Campo clave:** es un campo que identifica de manera única cada registro. Por ejemplo, en la tabla *Persona* el campo clave podría ser el *DNI*, ya que el *DNI* no se repetirá entre los diferentes registros.
- **Consulta:** una consulta es una petición de búsqueda que se realiza cuando se quiere obtener alguna información. También se conoce como *query* y se diferencian consultas de peticiones, inserciones de datos, actualización o eliminación. Estas consultas pueden ser ejecutadas sobre una o varias tablas.
- **Índice:** es una estructura que agrupa los campos clave de cada tabla. De esta forma, si buscamos a través de este índice podemos encontrar los registros de una forma más rápida.
- **Vista:** es una tabla virtual que engloba campos de una o diferentes tablas.
- **Informe:** es un listado de campos y registros seleccionados con un formato que facilita su lectura. Por ejemplo, un informe puede ser un listado de los trabajadores de mayor edad de una empresa.

Los informes se diseñan en función del usuario al que van destinados.



- **Guiones o scripts:** son un conjunto de instrucciones que cumplen una cierta funcionalidad. Estas instrucciones suelen realizarse para el mantenimiento de los datos de las tablas.
- **Procedimiento:** un procedimiento es un tipo especial de script que está almacenado en la propia BD.

### Usos de las BBDD

Hoy en día, las BBDD se utilizan en infinidad de áreas. A continuación, veremos varios ejemplos en los que las BBDD se utilizan para gestionar información:

- **Bibliotecas:** almacenamos la información perteneciente tanto a libros (títulos, autores, editoriales, años de publicación...) como a usuarios, que reservan y utilizan dichos libros (nombres, apellidos, fecha de alquiler, fecha de devolución...).
- **Censo:** guardamos información demográfica de los habitantes (nombres, edades, sexo...) y de sus municipios, comarcas, regiones, países...
- **Científicas:** guardamos información recogida en todo tipo de proyectos y experimentos científicos para su posterior tratamiento.
- **Administrativas:** cualquier empresa necesita mantener la información totalmente ordenada, estructurada y accesible para su buen funcionamiento.

### Tipos de BBDD

Existen diversos tipos de BBDD, pero en función del criterio que usemos para categorizarlas, obtendremos distintas clasificaciones.

A continuación, vamos a listar los tipos de BBDD según su **modelo de datos** (cómo están estructurados internamente) y según su **ubicación**.

#### TIPOS DE BBDD SEGÚN SU MODELO DE DATOS

- **Jerárquicas:** se organizan en una estructura de diferentes niveles, como un árbol invertido. Esta se asemeja a la organización de árbol que usa el sistema operativo para almacenar la información en el disco.
- **En red:** la información se organiza en una estructura de enlaces y nodos. Cada registro contiene un enlace a otro registro. No hay una jerarquía entre los nodos.
- **Relacionales:** la información se organiza a través de tablas. Las tablas pueden estar interrelacionadas entre sí, de modo que ciertos datos de una tabla pueden depender de otras tablas.

- **Orientada a objetos:** la información se estructura en objetos, los cuales pertenecen a clases. Este tipo permite evitar la redundancia gracias a la reutilización de las clases.

Si nos guiamos por el criterio de la ubicación, podemos dividir las bases de datos en *centralizadas* y *distribuidas*. Este punto lo estudiaremos un poco más adelante.

## 1.4. SISTEMAS GESTORES DE BASES DE DATOS (SGBD)

### CONCEPTO

Un **sistema gestor de base de datos (SGBD)** es un programa o software que facilita a la persona encargada de la administración de las BBDD el tratamiento de la información que contiene. Puede realizar operaciones como diseño, consulta y modificación de dichos datos.

A continuación, vamos a ver algunos de los más utilizados:



### Funciones, componentes y tipos

Las funciones de los SGBD se pueden clasificar en tres tipos diferentes:

- **Definición:** especifican los tipos de datos, las estructuras y las restricciones que se van a almacenar, asegurando la cohesión e integridad de estos.
- **Construcción:** proceso de almacenamiento de datos en algún medio controlado por el SGBD. En esta función, podemos citar la posibilidad que deben tener estos siste-



### ponte a prueba

¿A qué tipo de fichero se hace referencia con esta afirmación: “Disponemos de un fichero lleno de registros, y para encontrar un registro en concreto debemos recorrer todos los registros de forma secuencia que se encuentran antes que este.”?

- Ficheros indexados.
- Ficheros secuenciales.
- Ficheros planos.
- Ficheros de lectura continua.

Dadas las siguientes opciones, indica cuál no podría ser un campo clave en una base de datos.

- Nombre de persona.
- DNI.
- Matrícula de un coche.
- ID de una película.

En referencia a la clasificación de las BBDD. Según su modelo de datos, ¿en cuántos tipos las podemos clasificar?

- Jerárquicas.
- En red.
- Relacionales.
- Todas las opciones anteriores son correctas.

mas de gestión para que dicha BBDD se pueda conectar con el exterior utilizando algún medio. El estándar más utilizado para estos casos es el protocolo ODBC, que comunica la BD con una aplicación externa.

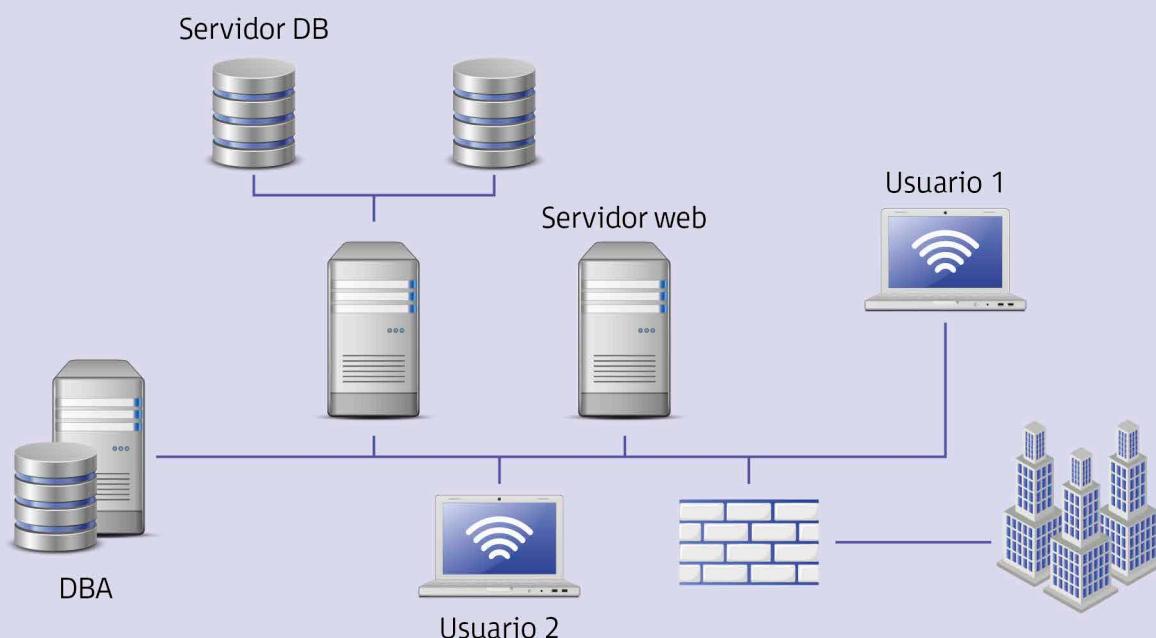
- **Manipulación:** incluye tareas como la manipulación y consulta de los datos almacenados para obtener una información.

Podemos mencionar las herramientas complementarias que ofrecen estos sistemas de gestión para las estadísticas sobre consultas, actualizaciones e incidencias ocurridas en las BBDD. Los SGBD deben estar implementados para que estas operaciones se realicen de forma sencilla y ofrezcan un gran rendimiento.

**Según su capacidad y potencia**, los podemos clasificar en dos grandes grupos:

- **SGBD ofimáticos:** destinados a manipular BBDD de uso doméstico o pequeñas empresas. Presentan una interfaz intuitiva y sencilla. El administrador de este sistema no tiene que ser un usuario experto. Por ejemplo, **Microsoft Access**.
- **SGBD corporativos:** implementados para manipular BBDD con una capacidad mucho mayor que los anteriores. Suelen ser implantados en grandes o medianas empresas con una gran carga de datos y un volumen alto de transacciones, de tal forma que requieren de un servidor con altas prestaciones. El ejemplo más extendido de estos SGBD es **ORACLE**.

Esquema de un SGBD.



Según su **licencia de uso**, podemos dividir los SGBD en dos grupos:

- **SGBD comerciales:** son aquellos que requieren una licencia de uso, de tal modo que el usuario debe abonar un pago establecido para tener el derecho al uso o redistribución del SGBD. Además, un SGBD comercial puede ser de código cerrado, lo que significa que el usuario del SGBD no tiene acceso al código fuente y, por tanto, no puede modificarlo.

Normalmente este tipo de software suele tener una garantía de mantenimiento así como una cierta estabilidad en su funcionamiento. Algunos de los SGBD comerciales más comunes son Oracle, Microsoft Access o Microsoft SQL Server.

- **SGBD libres:** son aquellos que no requieren de una licencia de uso por parte del usuario, ni para la utilización ni para la distribución. Es común que los SGBD libres, más allá de su uso gratuito, dispongan de código abierto, de modo que el usuario pueda modificar el mismo SGBD. Algunos de los SGBD libres más comunes son MySQL, PostgreSQL y MariaDB.

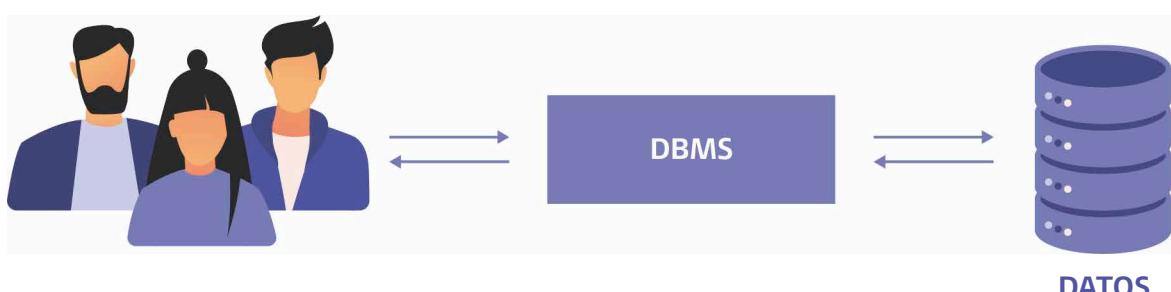
### Objetivos del SGBD

Los SGBD o *Data Base Management System*, en inglés, son programas de software que brindan a los distintos usuarios la posibilidad de procesar, describir, administrar y recuperar aquellos datos almacenados en las BBDD. Ofrecen una serie de programas y procedimientos que permiten a los usuarios llevar a cabo las diferentes tareas, teniendo siempre especial cuidado con la seguridad de los datos.

**¿SABÍAS QUE...?**

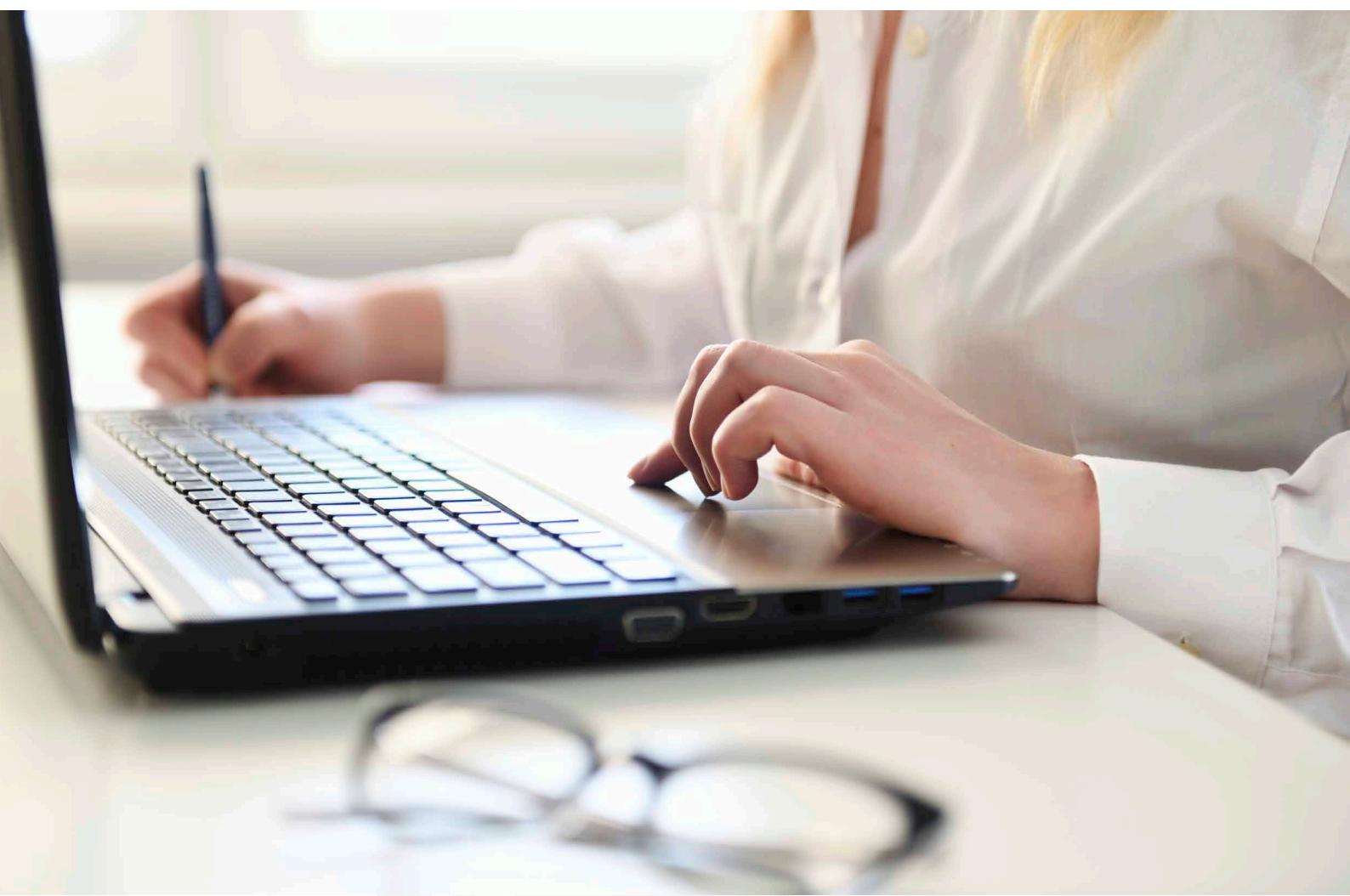
**MySQL®**

Oracle es una empresa que ofrece un SGBD de tipo comercial (también ofrece otro libre) y MySQL está más enfocado a los SGBD libres.



Para que los SGBD puedan mantener la seguridad e integridad de todos los datos, deben proporcionar una serie de herramientas a los usuarios, entre las que podemos encontrar:

- Creación y especificación de los datos: crean la estructura física que se requiera en cada unidad.
- Manipulación de los datos de las BBDD: añaden, modifican, suprimen o consultan los datos.



- Recuperación: se lleva a cabo mediante la creación de copias de seguridad.
- Gestión de la comunicación de la BBDD.
- Creación de aplicaciones.
- Instalación de la BBDD.
- Exportación e importación de datos.

### Tipos de usuarios de BBDD: informáticos y no informáticos

Podemos diferenciarlos de la siguiente manera:

- **Informáticos:** usuarios sofisticados que se comunican con el sistema mediante consultas, incluyendo a los especializados, capaces de diseñar las aplicaciones de la BBDD en diferentes sistemas. Los administradores de la BBDD también forman parte de los informáticos.
- **No informáticos:** aquellos que simplemente interactúan con el programa mediante interfaces de formularios rellenando datos.

## Administrador de la base de datos (DBA): funciones y responsabilidades

Nos referimos a administradores de la base de datos para referirnos a aquellas personas que tienen el control sobre la BBDD en cuestión. Entre sus funciones principales podemos encontrar:

- Definir el esquema de la BBDD.
- Definir sus estructuras de almacenamiento.
- Modificar dicho esquema y su organización física.
- Asignar autorización para su uso.
- Especificar restricciones de integridad.

## Tipos de lenguajes de BBDD

Podemos diferenciar cuatro tipos de lenguajes, que detallamos a continuación:

- **DDL (Data Definition Language)**: es el lenguaje de definición de datos y se utiliza para crear la estructura de una BBDD y, también, para diseñar las vistas del nivel externo. Genera una serie de tablas que se almacenan en un archivo al que llamamos *diccionario de datos*.
- **DML (Data Manipulation Language)**: es el lenguaje de manipulación de datos y se utiliza para realizar operaciones sobre los datos, como insertar, modificar, borrar y consultar. Pueden ser de dos tipos:
  - **Procedimentales**: cuando tenemos que especificar de qué forma se obtienen los datos.
  - **No procedimentales**: cuando solo tenemos que especificar qué datos son los que se van a necesitar.
- **DCL (Data Control Language)**: es el lenguaje que nos permite crear permisos y roles y, de ese modo, controlar el acceso a la BBDD. Utiliza GRANT para dar privilegios y REVOKE para retirarlos.
- **TCL (Transactional Control Language)**: permite administrar las transacciones que ocurren en la BBDD. Emplea COMMIT para guardar el trabajo realizado y ROLLBACK para deshacer lo realizado desde el último COMMIT.

## Diccionario de datos: concepto, contenido, tipos y uso

Un diccionario de datos es el lugar donde se va a depositar la información referente a los datos que forman la BBDD. Contiene las características lógicas de los sitios donde se van a almacenar los datos del sistema.

El diccionario proporciona información sobre la **estructura lógica y física de la BD**:

- **Define todos los objetos** de la BD: tablas, vistas, funciones, procedimientos, etc.
- **Define el espacio** que tiene asignado y va a ser utilizado por los objetos.
- **Define los valores** por defecto de las columnas de las tablas.
- **Define los privilegios** asignados.
- Contiene información sobre **restricciones** de integridad.

Aquí podemos ver un ejemplo:

**Nombre:** Empleado **Fecha de creación:** 27/09/2018

**Descripción:** Información de cada empleado

Campo	Tipo	Tamaño	Descripción
<b>idEmpleado</b>	Number	20	Campo primario para identificar al cliente
<b>nombre</b>	Varchar	60	Nombre del cliente
<b>apellidos</b>	Varchar	95	Apellido del cliente
<b>salarioE</b>	Float	10	Cantidad salarial anual
<b>departamentoE</b>	Number	5	Número del departamento al que pertenece
<b>fechaEntrada</b>	Date	10	Fecha de incorporación del empleado

**Relaciones:** *departamentoE* con tabla *Departamento*

**Campos clave:** *idEmpleado*

Además, un diccionario de datos debe cumplir una serie de **características**, como son:

- Soportar descripciones del modelo conceptual.
- Apoyar la transferencia de información.

- Estar integrado dentro de DBMS.
- Actualizar los cambios en la descripción de la BBDD.
- Estar almacenado mediante un acceso directo para tener una fácil recuperación de información.

Entre los **tipos** de diccionarios de datos, podemos encontrar los siguientes:

- **Off-Line**: su función es mantener el diccionario.
- **On-Line**:
  - Funciona con el compilador.
  - No deja que el programador defina los datos, sino que los define directamente.
  - Se asegura de que los datos existen en el diccionario.
  - Añade la definición de los datos.
- **In-Line**: no añade la definición de los datos hasta que no se ejecuta.

## 1.5. ANSI/X3/SPARC. ESTÁNDARES Y NIVELES

En 1975, el organismo ANSI-SPARC (*American National Standards Institute – Standards Planning And Requirements Committee*) creó un diseño abstracto para estandarizar la manera en que los Sistemas de Gestión de bases de datos (SGBD) administraban las BBDD.

Este diseño constaba de una arquitectura de tres niveles: el externo, el conceptual y el interno. De este modo, se conseguía separar la visión que los usuarios tenían de la BD con los detalles internos de su estructura.

Esta arquitectura nunca llegó a estandarizarse formalmente, sin embargo, hoy en día la mayoría de los SGBD usan como referente esta propuesta.

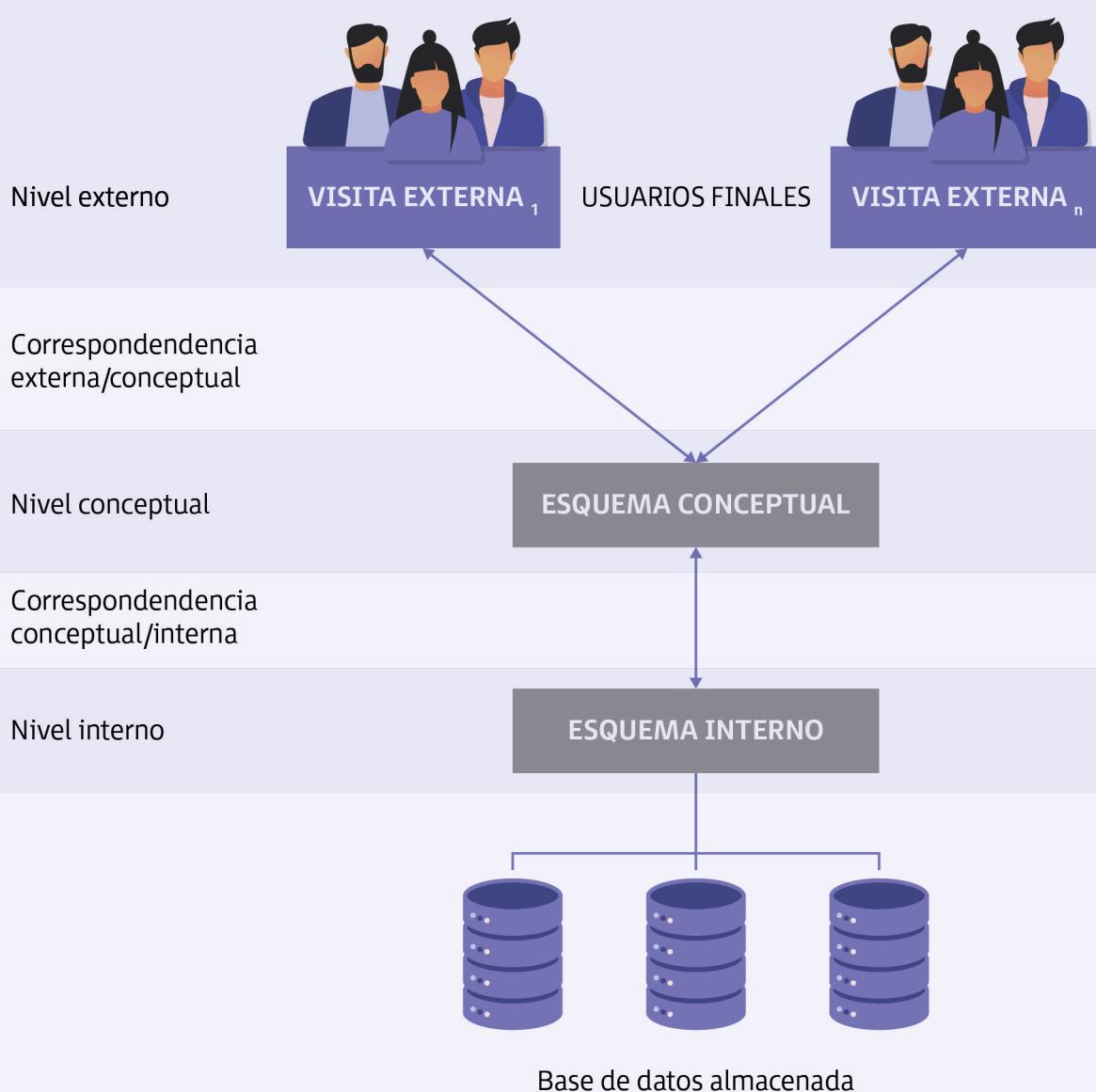
- **Nivel externo o de visión**: es el nivel enfocado al usuario de la BD. En este nivel se muestra al usuario la información que le es pertinente y en el formato adecuado. En este nivel se oculta al usuario la información que no tiene permitido ver y, además, se omiten los detalles técnicos sobre la información a la que sí tiene acceso.
- **Nivel conceptual**: es el nivel que determina la organización de los datos en la BD. En este nivel está determinado la información que se almacena en la BD y qué relaciones e interrelaciones existen entre los mismos datos. El administrador de la BD debería ser el único usuario con acceso

a este nivel. Este nivel es un nivel lógico, por tanto es independiente al software y hardware utilizado.

- **Nivel interno o físico:** es el nivel que determina cómo están almacenados los datos físicamente en el sistema informático. Este nivel concreta los detalles de almacenamiento de cada conjunto de datos, tales como el tipo de dato para cierto campo, el método de acceso a una tabla, etc.

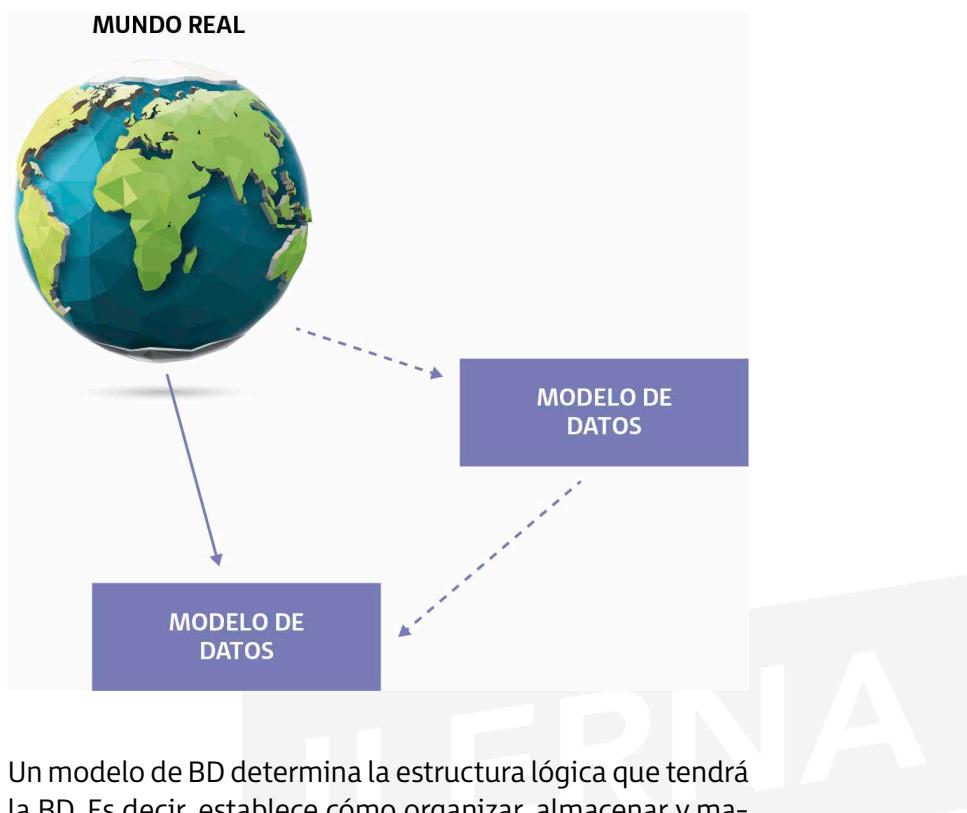
Esta arquitectura permite a que el DBA cambie sin que los usuarios lo vean.

No hay que confundir la arquitectura de tres niveles de una BD en funcionamiento con los pasos o fases que se deben realizar para la creación de una BD de inicio al final (recopilación de la información, creación del modelo entidad-relación, paso al modelo relacional e implementación del modelo físico).



*Representación niveles de abstracción de la arquitectura DBMS.*

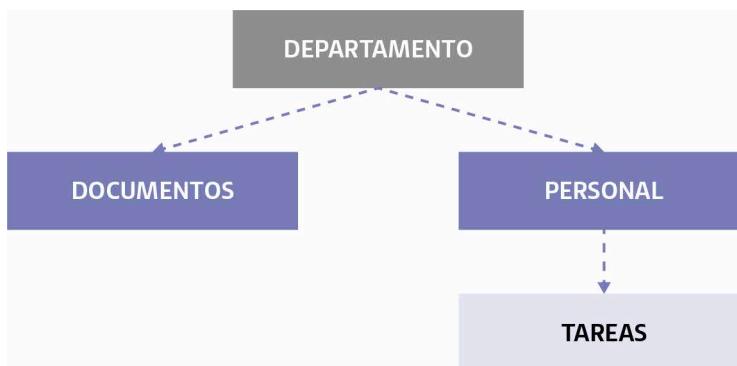
## 1.6. MODELOS DE BBDD. TIPOS: JERÁRQUICO, RED, RELACIONAL Y ORIENTADO A OBJETOS



Un modelo de BD determina la estructura lógica que tendrá la BD. Es decir, establece cómo organizar, almacenar y manipular los datos. Además, también indica qué operaciones se pueden llevar a cabo con los datos.

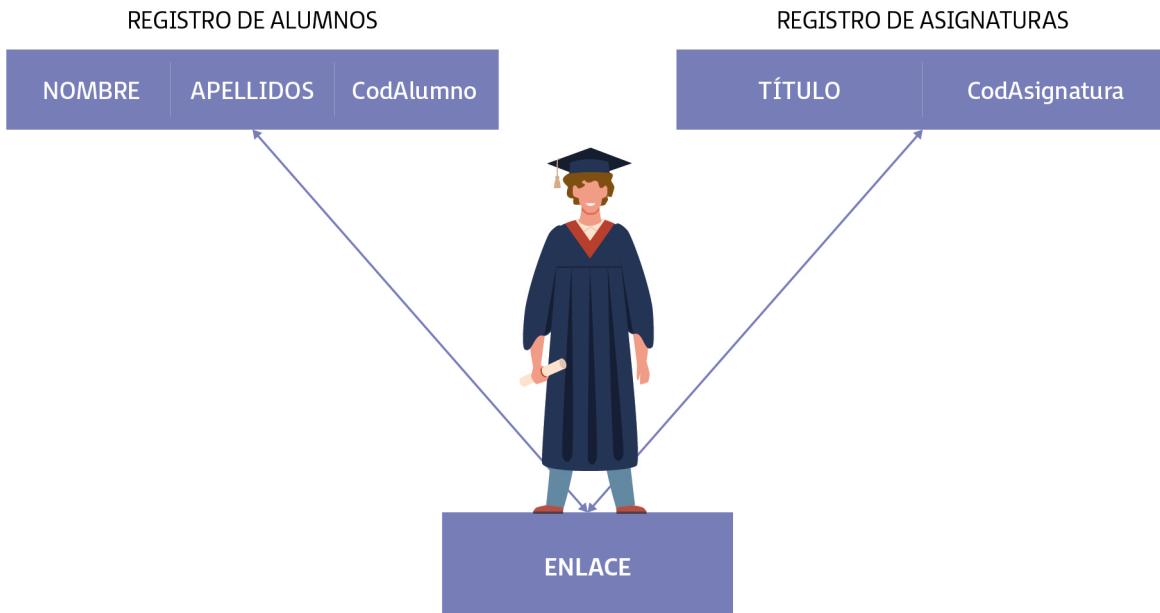
A continuación, se detallan los modelos lógicos más comunes en BBDD:

- **Modelo jerárquico:** la información se organiza de manera jerárquica, utilizando una relación entre las diferentes entidades, siguiendo el tipo padre-hijo. Existe una serie de nodos que contienen atributos que se relacionan con nodos-hijos, de manera que puede haber más de un hijo para el mismo padre, pero un hijo no puede tener dos padres.



*Esquema jerárquico.*

- **Modelo en red:** este modelo estructura la información en registros (nodos) y enlaces. En los registros se van a almacenar los datos, mientras que en los enlaces se podrán relacionar. Este modelo ofrece la posibilidad de tener más de un parente.



*Esquema en red.*

- **Modelo relacional:** fue propuesto por Edgar Frank Codd en los años 70. Este modelo organiza la información de la BD con el uso de relaciones y tuplas. Una tupla se puede entender como un registro y una relación se puede entender como una tabla. Una tabla contiene varios registros.

Este modelo garantiza la normalización de la BD, evita duplicidades de registros (es decir, evita que el valor de un campo clave sea el mismo en dos tuplas distintas) y permite la integridad referencial entre tuplas de diferentes tablas.

- **Modelo entidad-relación (ER):** es el modelo más usado hoy en día, debido a la buena evolución que ha experimentado. No obstante, posee distintas variantes y esto provoca que su representación no es totalmente estándar todavía.

De las variantes, podemos destacar el modelo **entidad-relación extendido (ERE)**, que complementa algunas carencias del modelo original.

Se debe diferenciar bien el modelo ER con las BBDD relacionales. Los esquemas entidad-relación pueden ser usados con cualquier SGBD.

- **Modelo orientado a objetos:** este modelo utiliza programación orientada a objetos (POO), ya que ofrece la posibilidad de cohesionar datos y procedimientos. De



esta manera, se pueden diseñar estructuras que poseen datos (atributos) en las que se permite definir los diferentes procedimientos (operaciones) que se van a realizar con los atributos.

La POO se ideó para BBDD adaptadas a estos lenguajes. Las BBDD orientadas a objetos siguen esta misma filosofía. De este modo, se pretende que este modelo de BBDD pueda solucionar las limitaciones que tienen las relacionales.

Las BBDD orientadas a objetos son de las que más están creciendo en los últimos años.

### Reglas de integridad de los datos

Es fundamental que los SGBD puedan garantizar que se va a trabajar de forma segura, apostando siempre por la integridad de los datos. Es cierto que la redundancia de estos puede provocar un mal funcionamiento e, incluso, puede que los datos sufran pérdidas de integridad.

Algunos de los motivos principales por los que se puede perder el valor inicial de los datos pueden ser los errores provocados por los programas, los usuarios o una avería en el sistema, entre otros.

Existen diferentes **reglas de integridad** que se deben tener siempre en cuenta para garantizar un buen funcionamiento:

- **Reglas de integridad del modelo:** deben de cumplirse por el SGBD para que no se pierda integridad cada vez que se realicen actualizaciones de los programas.

- **Reglas de integridad del usuario:** deben de cumplirse por los usuarios ya que en caso de producirse algún error en el que se pierda información, el SGBD está capacitado para dar todas las herramientas necesarias para poderlos reconstruir y, de esta forma, conseguir que no se pierda integridad en el sistema.

### Modelo distribuido: ventajas e inconvenientes, técnicas de fragmentación, distribución de datos y esquemas de asignación y replicación de datos

Un **modelo de base de datos distribuido (BDD)** constituye un grupo de diferentes máquinas conectadas a través de una red. Cada procesador de los que se utilizan tiene sus dispositivos de entrada y salida y estos intercambian mensajes para conseguir un objetivo común.

Sus **ventajas** principales son:

- **Mejora del rendimiento**, ya que permite que varias máquinas conectadas en la red trabajen a la vez con diferentes usuarios.
- Aunque uno de los equipos falle, **los demás pueden seguir trabajando**.
- Todos los equipos comparten el **mismo estado**, aunque cada uno trabaje de forma independiente.

Aunque también es importante señalar algunos de los **inconvenientes** que presentan:

- Al estar todos los equipos interconectados en la red, puede que aumente la **pérdida de mensajes y la saturación**.
- **Menor confidencialidad** de los datos.

En un modelo distribuido podemos encontrar diferentes técnicas utilizadas para la **fragmentación**:

- **Horizontal:** se basa fundamentalmente en particionar tuplas en subconjuntos. De esta manera, cada subconjunto debe contener aquellos datos que son comunes. Así, cada fragmento, se puede definir como una operación de selección.
- **Vertical:** la fragmentación vertical se basa en subdividir los atributos en grupos. De esta manera, podemos obtener los distintos fragmentos si proyectamos la relación global sobre todos los grupos. Esta fragmentación es exitosa si se puede localizar a cada atributo en, al menos, otro atributo del fragmento en cuestión.
- **Mixta:** en este tipo de fragmentación se van a combinar la horizontal con la vertical o viceversa.

## 1.7. BASES DE DATOS CENTRALIZADAS Y DISTRIBUIDAS

Una **base de datos centralizada** es una base de datos que está almacenada de manera íntegra en un solo lugar, es decir, en una misma máquina. Sin embargo y tal y como se comentaba en el apartado anterior, una **base de datos distribuida (BDD)** es un conjunto de BBDD que se encuentran lógicamente relacionadas. Esto significa que se distribuyen en diferentes sitios, de forma que se necesita una interconexión de red para comunicarse. A continuación, detallamos con más precisión cada una de ellas:

### Base de datos centralizada (BDC)

<b>Definición</b>	Las BBDD centralizadas son aquellas que se encuentran <b>almacenadas en una única computadora</b> , por lo que el sistema informático no interacciona con ninguna otra máquina.  Ejemplos de estos sistemas pueden ser BBDD básicas de un solo usuario o BBDD de alto rendimiento, implantadas en grandes sistemas.
<b>Características</b>	Almacena todos los componentes en una única máquina.  No tiene demasiados elementos de procesamiento.  Componentes: datos, software de gestión de BBDD y dispositivos de almacenamiento.
<b>Ventajas</b>	No presenta redundancia ni inconsistencia, ya que se focaliza todo en un sistema central. Si se tratara de una BD no centralizada, existiría redundancia de la información y, por tanto, problemas con el espacio de almacenamiento.  Puede aplicar restricciones de seguridad.  Rendimiento óptimo al procesar datos.  Se evita la inconsistencia de los datos, ya que solo existe una sola entrada para cada dato almacenado.
<b>Inconvenientes</b>	Ante un problema, la recuperación de datos es complicada.  No existe la posibilidad de repartir las tareas al intervenir solamente una máquina.  Si un sistema falla, perdemos la disponibilidad de la información.  Los <i>mainframes</i> (ordenadores centrales) ofrecen una relación entre el precio y el rendimiento bastante costosos.

### Base de datos distribuida (BDD)

<b>Definición</b>	Una BDD es aquella en la que interviene un conjunto de múltiples BBDD relacionadas. Se encuentra en diferentes espacios lógicos y geográficos, pero está interconectada por una red. Estas BBDD son capaces de procesar de una forma autónoma, es decir, pueden trabajar de forma local o distribuida.
<b>Características</b>	Autonomía: los componentes, el SO y la red son independientes y cada uno realiza las diferentes operaciones desde su propio sitio. No necesita ni depende de una red central para obtener un servicio. Presenta la posibilidad de leer y escribir datos ubicados en lugares diferentes de la red. Puede convertir transacciones de usuarios en instrucciones para manipular datos.
<b>Ventajas</b>	Acceso rápido. Al intervenir varios nodos el procesamiento es más rápido. Los nuevos nodos que intervengan se crean de forma rápida y fácil. Mejora la comunicación entre distintos nodos. Refleja una estructura organizativa donde las BBDD se almacenan en los departamentos donde tienen relación. Bajo coste a la hora de crear una red de pequeñas computadoras. Al presentar una red de BBDD se implementa de forma modular. Esto hace que las operaciones de modificar, insertar o eliminar alguna BD sean mucho más fáciles que en el ejemplo anterior.
<b>Inconvenientes</b>	Presenta una estructura de diseño más compleja. Aumenta el riesgo de violaciones de seguridad. Mecanismos de recuperación más complejos, debido a que existen muchos más datos.

### Componentes: hardware y software

A nivel de componentes **hardware**, las bases de datos Distribuidas suponen un mayor uso de infraestructura, ya que, a diferencia de las bases de datos Centralizadas, las primeras ubican sus datos en más de una máquina (denominados nodos o sitios).

En referencia al **software** necesario, las bases de datos Distribuidas deben interconectar los nodos que las componen, por lo que necesitan de una red a través de la cual transmitir la información entre los mismos.

## Niveles de procesamiento de consultas: procesadores locales y distribuidos

En los **procesadores locales** solamente se hace referencia tanto a tablas como a datos locales, es decir, a aquellos que pertenecen a una misma instancia en una única máquina. Las subconsultas que se ejecutan en un nodo (consulta local) se van a optimizar utilizando el esquema local del nodo. Los diferentes algoritmos se pueden elegir para ejecutar las operaciones relacionales.

En los **procesadores distribuidos**, en cambio, el objetivo principal va a ser pasar las transacciones de usuario a instrucciones para poder manipular los datos. El principal problema que presentan es de optimización, ya que se determina el orden en el que se realizan el número mínimo de operaciones.

### CONCEPTO

El **procesamiento de consultas** es bastante más complicado en los procesadores distribuidos que en los locales. Esto se debe a que en los distribuidos interviene un gran número de parámetros que pueden afectar al rendimiento de las consultas que se tengan que realizar.

La función más importante de un procesador de consultas relacionales va a ser transformar una consulta de una especificación de nivel alto (normalmente en cálculo relacional) en otra equivalente de bajo nivel (normalmente en álgebra relacional).

Esta transformación se debe llevar a cabo y, si se consigue, debe ser perfectamente correcta y eficiente.

## Bloqueo y concurrencia. Transacciones distribuidas

Cuando nos referimos a los SGBD debemos señalar que estamos ante sistemas concurrentes. Estos sistemas van a ejecutar sus consultas y estas se van a ir procesando al mismo tiempo.

Las transacciones distribuidas se pueden definir como transacciones planas o anidadas, que pueden acceder a objetos que han sido administrados por diferentes servidores. Cuando una transacción distribuida finaliza, esta necesita que todos los servidores que han formado parte del proceso verifiquen su buen funcionamiento. En caso de que no se pueda verificar, debe abortar la transacción. Este mecanismo se puede llevar a cabo gracias a su propiedad de atomicidad.

## Distribución de los datos

La distribución de los datos es una tarea que corresponde al diseñador. Este se va a encargar de elegir dónde se va a posicionar y qué esquema va a representar dicha BD. Podemos encontrar las siguientes distribuciones: centralizadas, replicadas, particionadas e híbridas.

- **Centralizada:** esta distribución es muy parecida al modelo cliente/servidor, en el que la base de datos está centralizada en un nodo central y se distribuye entre los distintos clientes. La desventaja que supone utilizar esta distribución es que la disponibilidad depende de un solo nodo.
- **Replicadas:** es un esquema muy costoso, ya que cada nodo va a tener información duplicada. Es más lento, porque tiene muchos datos a almacenar, pero merece la pena a largo plazo, ya que va a tener mucha disponibilidad a la hora de leer la información.
- **Particionadas:** en este caso solo tenemos una copia de cada nodo. De todas formas, cada nodo alojará algunos fragmentos de la BD. Esto hace que el coste sea más reducido aunque, también, va a tener menos disponibilidad que el anterior.
- **Híbridas:** aquí vamos a representar la partición y replicación del sistema.

## Seguridad y recuperación de la información en bases de datos distribuidas

Existen diferentes tipos de ataques a la seguridad entre los que podemos destacar: de privacidad y confidencialidad de los datos, los que están asociados a la autenticación y los que deniegan al servicio.

En cuanto a las herramientas de seguridad, debemos señalar los distintos protocolos de seguridad, el cifrado de las claves y los cortafuegos.

Para poder recuperar los datos, debemos tener activa la tolerancia a fallos que permite que, en caso de fallo de algún componente, el sistema siga funcionando de forma correcta.

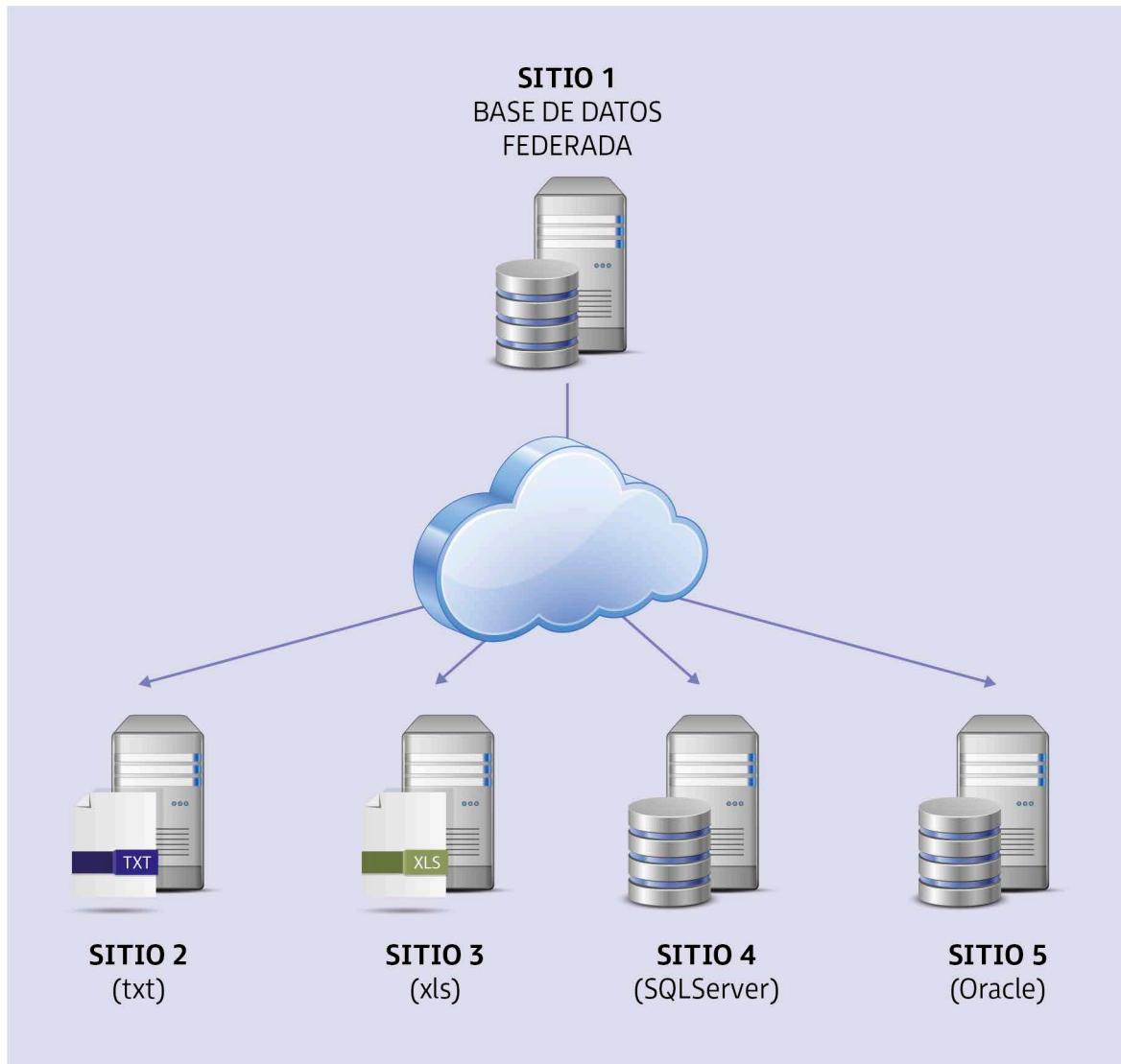
## Arquitectura-implementaciones: múltiples y federadas

Las **BBDD federadas** son un conjunto de sistemas de BBDD que trabajan de forma cooperativa y autónoma.

Los usuarios pueden acceder a los datos a través de una interfaz. Esta interfaz no tiene diseñado un esquema total

donde estén todos los datos, sino que contiene diferentes esquemas más pequeños que hay que unificar.

Las **BBDD federadas** parecen BBDD normales, pero no tienen existencia física por sí solas: son una vista lógica.



Arquitectura

Las **BBDD múltiples** actúan como una interfaz de varios componentes diferentes. Una BD múltiple cuenta con distintas operaciones que facilitan el acceso a la información, manteniendo la consistencia de esta y ofreciendo un acceso uniforme a los servicios.

### Diseño y gestión de BBDD distribuidas

Cuando necesitemos diseñar una BD distribuida, tendremos que dar una serie de pasos que nos permitan tener el diseño correcto para almacenar nuestros datos. Existen una serie de pasos que debemos seguir a la hora de diseñar una BD distribuida, veamos los más importantes:



1. **Diseñar el esquema conceptual:** para empezar, necesitaremos detallar el esquema conceptual de toda la BD.
2. **Diseñar la BD:** posteriormente, organizar el esquema conceptual y establecer sus métodos de acceso.
3. **Diseñar fragmentación:** necesitaremos fragmentar, es decir, realizar subdivisiones en fragmentos de las diferentes partes de la BD.
4. **Diseñar asignación de fragmentos:** por último, organizar y seleccionar cómo se van a unir los diferentes fragmentos previamente creados.

Para después gestionar correctamente nuestra BD, necesitaremos tener una BD estable, cuyas relaciones sean coherentes y su interconexión entre los nodos sea correcta. Para administrar las transacciones en BBDD distribuidas podemos utilizar lo que se conoce como **Administrador de Transacciones Distribuidas (DTM)**. Esto es un programa que procesa y coordina las consultas o transacciones de nuestra BD.

**ponte a prueba**

**Indica cuál de las siguientes opciones es una ventaja de trabajar con bases de datos distribuidas.**

- a) Supone un bajo coste a la hora de crear una red de computadoras pequeña.
- b) Aumenta el nivel de seguridad.
- c) Los mecanismos de recuperación de datos son óptimos gracias a que intervienen distintos nodos.
- d) Solamente tiene una entrada para cada dato que se almacena.

**Indica cuál de las siguientes opciones pertenece a una base de datos centralizada.**

- a) No depende ni necesita una red central para obtener servicio.
- b) No tiene demasiados elementos de procesamiento.
- c) No almacena todos los componentes en una única máquina.
- d) No puede aplicar restricciones de seguridad.

**El diseñador es el encargado de distribuir los datos en una base de datos. ¿Cuál de las siguientes opciones se corresponde con un esquema costoso, en el que cada uno de los nodos tendrá la información duplicada, que también dispone de mucha disponibilidad pero que resulta más lento al tener muchos datos?**

- a) Distribución centralizada.
- b) Distribución replicada.
- c) Distribución particionada.
- d) Distribución híbrida.

**Indica cuál de las siguientes opciones es una de las ventajas principales de las bases de datos centralizadas.**

- a) Ante un problema, la recuperación de datos es complicada.
- b) Si un sistema falla, perdemos la disponibilidad de la información.
- c) Rendimiento óptimo al procesar datos.
- d) Acceso rápido.



# 2

## MODELO ENTIDAD-RELACIÓN

El modelo entidad-relación es un modelo de datos abstracto, que nos permite representar de manera conceptual una base de datos.

Se compone de 2 elementos principales:

- El diagrama entidad-relación.
- Las restricciones y otras anotaciones que no se pueden representar en el diagrama (como las claves candidatas).

El diagrama es el elemento principal del modelo entidad-relación. El diagrama entidad-relación se compone principalmente de entidades, relaciones entre las entidades, atributos y cardinalidades.

Este modelo nos ayuda a la creación de una BD. Se realiza tras la recopilación de información sobre la futura BD que se desea construir y antes del modelo relacional.

Para contextualizar el modelo entidad-relación dentro de la creación de una BD, vamos a describir las fases que precisa una BD para su completa formación:

- 1. Recopilar información:** es el primer paso. Consiste en recabar toda la información posible de la BD que deseamos diseñar. Esta información puede ser dada por el cliente o por el mismo desarrollador, dependiendo del caso.
- 2. Realizar el modelo entidad-relación:** tras la recopilación de información se realiza el diseño del modelo entidad-relación. Para ello, se refleja toda la información recogida en el diagrama entidad-relación y en las restricciones asociadas al diagrama. Este modelo corresponde al modelo conceptual.
- 3. Paso del modelo entidad-relación al modelo relacional:** una vez ya tenemos listo el modelo **entidad-relación**, realizamos la conversión de este al modelo relacional. Para ello usaremos unas reglas determinadas. El modelo relacional es el modelo lógico.
- 4. Normalización:** en esta fase retocamos el modelo relacional para hacerlo más eficiente de cara a su codificación.
- 5. Codificación:** en esta fase trasladamos el modelo lógico que hemos diseñado de una BD a una máquina física, como un servidor. Para ello tendremos que elegir el hardware más pertinente y también el software más adecuado, es decir, el SGBD. Una vez hayamos elegido estos dos elementos, pasaremos a codificar en un Lenguaje de bases de datos (SQL) el modelo lógico anteriormente creado. Esta fase pertenece al modelo físico.

Por tanto, podemos definir tres tipos de modelos de una base de datos:

- **Modelo conceptual:** representación estructurada de la realidad en entidades y relaciones.
- **Modelo lógico:** especificación de todas las tablas.
- **Modelo físico:** archivo SQL que determina en el sistema informático la BD.

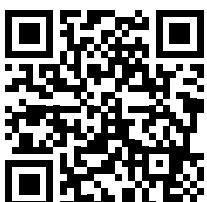
Veamos un ejemplo de BD en el que podemos utilizar el modelo entidad-relación:

### EJEMPLO

En unos grandes almacenes se tiene que publicar un folleto comercial. Gracias a la BD de la empresa, el departamento encargado sabrá qué productos consumen sus clientes y en qué fechas lo hacen. Con esta información, se ofertan los productos más consumidos y también pueden conseguir fidelizar a sus clientes y atraer a compradores potenciales.



**Modelo entidad-relación (e-r) y sus distintos componentes**  
[youtu.be/faDWd5niMOE](https://youtu.be/faDWd5niMOE)



## 2.1. CONCEPTO DE MODELO ENTIDAD-RELACIÓN

Como ya introdujimos anteriormente, el modelo entidad-relación es un arquetipo conceptual que representa un problema planteado a través de entidades y relaciones.

Debemos diferenciar entre modelo entidad-relación y el diagrama entidad-relación. El modelo entidad-relación es el modelado abstracto de una BD, el cual usa la herramienta del diagrama para representarse y las restricciones para matizar el diagrama y completar el modelado conceptual de la BD. Simplificando, se podría expresar, como se expresó anteriormente, que el modelo entidad-relación consiste principalmente en un diagrama y unas restricciones asociadas.

Ejemplo de una representación de dos entidades y una relación:



**ENTIDADES:** Cada uno de los objetos de los que almacenamos los datos.

**RELACIÓN:** Asociación entre entidades.

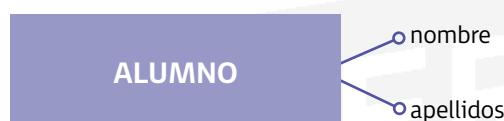
## 2.2. ENTIDAD: REPRESENTACIÓN GRÁFICA, ATRIBUTOS Y TIPOS DE CLAVES

Podemos definir las **entidades** como la representación de aquellos elementos (físicos o abstractos) de los que se desea almacenar la información.

Se pueden representar gráficamente mediante un **rectángulo** que contiene en su interior el nombre del elemento al que representan. Este nombre debe ser único, es decir, no puede aparecer repetido en nuestro diagrama.



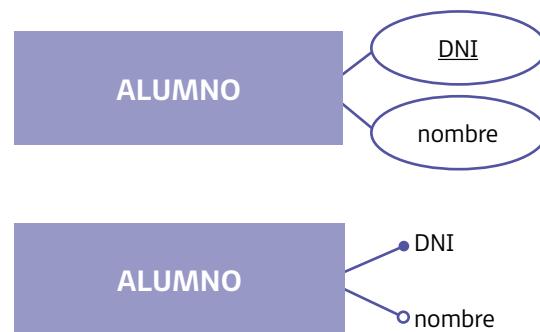
Cuando hablamos de **atributos de una entidad** hacemos referencia a las propiedades o características que tiene una instancia en particular de esa entidad. Por ejemplo: título, autor, nombre o fecha. Un ejemplo de un elemento en el que se representan sus atributos sería el siguiente:



O también puede representarse:



Respecto a los atributos, hemos de saber que un atributo o un conjunto de atributos pueden conformar una **clave primaria**, la cual permite identificar de manera única un registro de una tabla. Este concepto lo ampliaremos más adelante, y su representación en el diagrama entidad-relación es la siguiente:



Estas son dos formas de representar la clave primaria en el diagrama D. En este caso, la clave primaria de la entidad *Alumno* es *DNI*.

- **Atributo multivaluado:** un atributo es multivaluado cuando para la misma instancia de una entidad, el atributo posee varios valores posibles. En tal caso, se representa con una doble circunferencia:



Una instancia de una entidad es un valor concreto. Por ejemplo, de la entidad *Alumno* podríamos tener diversas instancias y cada una sería un alumno específico, por ejemplo el alumno “Juan Pérez”, la alumna “María Gómez” o el alumno “Marcos Pino”. Todos ellos serían posibles instancias de la entidad *Alumno*.

- **Atributo derivado:** un atributo es derivado o calculado cuando se puede deducir de otro u otros atributos. Se representa con una circunferencia discontinua. Por ejemplo, podemos conocer la edad de un alumno si conocemos su fecha de nacimiento y la fecha actual.





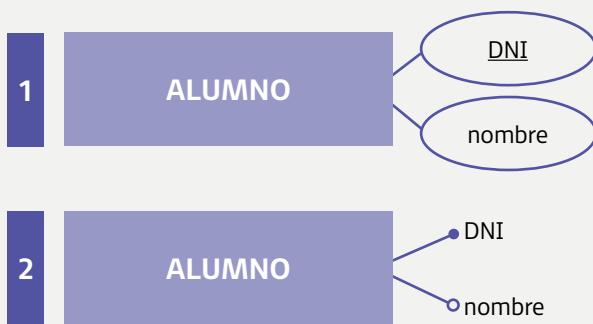
### ponte a prueba

¿Cuál es el orden correcto de los nombres de los elementos de la siguiente imagen (de arriba abajo)?

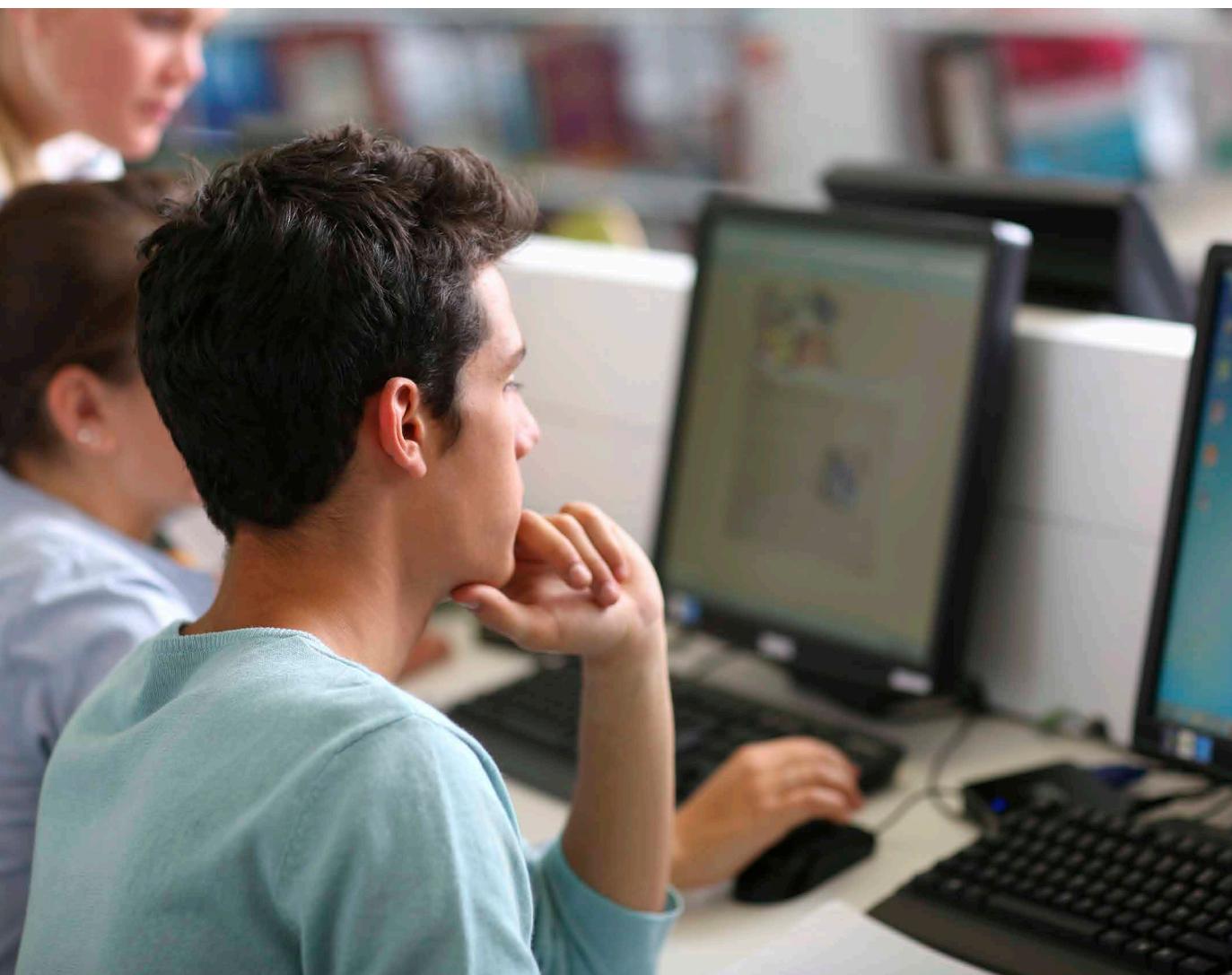


- a) Relación, atributo multivaluado, entidad, entidad débil atributo.
- b) Entidad débil, atributo, relación, entidad, atributo multivaluado.
- c) Atributo, Atributo multivaluado, relación, entidad, entidad débil.
- d) Atributo multivaluado, atributo, relación, entidad, entidad débil.

¿Cuál de las dos formas es correcta para representar un diagrama E-R?



- a) La opción 1.
- b) La opción 2.
- c) Ambas opciones son correctas.
- d) Ninguna opción es correcta.



## 2.3. RELACIÓN: REPRESENTACIÓN GRÁFICA, ATRIBUTOS, GRADO Y CARDINALIDAD

La relación sirve para **especificar las conexiones entre las diferentes entidades**, dándoles así un significado semántico más completo.

La palabra escogida para identificar la relación acostumbra a ser un verbo e indica la relación existente entre dichas entidades. Este puede estar en infinitivo o en una forma verbal, normalmente *Presente*.

Se pueden representar gráficamente mediante un **rombo**:



Por ejemplo, imaginemos que estamos diseñando la BD de un instituto. Tendríamos la entidad *Alumno*, la entidad *Asignatura* y la relación *Estudia*. Por tanto, representaríamos en el diagrama entidad-relación la realidad de que los alumnos de la BD estudian asignaturas de este modo:



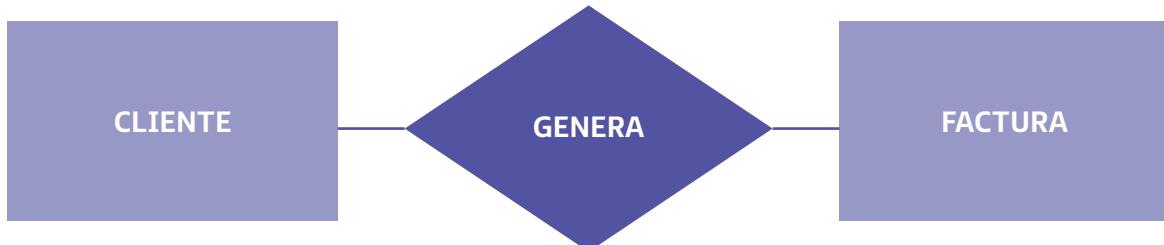
## 2.4. DIAGRAMA ENTIDAD-RELACIÓN

A continuación, nos adentramos en los distintos conceptos, características o funcionalidades que presentan los diagramas entidad-relación.

### Cardinalidad

La **cardinalidad** representa la participación que hay entre las entidades, es decir, el número de instancias de una entidad respecto a otra entidad con la que está relacionada y viceversa.

Por ejemplo, imaginemos las entidades *Cliente* y *Factura*, y la relación *Genera*:



Pues bien, la cardinalidad nos indicará si un cliente genera una factura o muchas facturas. Y al revés también. La cardinalidad nos indicará si una factura puede ser generada por un cliente o por muchos clientes. **Las cardinalidades posibles son: de uno a uno, de uno a muchos y de muchos a muchos.**

Veámoslo en la siguiente tabla:

CARDINALIDAD	EXPLICACIÓN
<b>1:1 (Uno a uno)</b>	Este caso ocurre cuando las instancias de dos entidades están relacionadas, como máximo, con solo una instancia de la otra entidad. Es decir, que no habrá una instancia de una entidad que pueda relacionarse con dos o más instancias de la otra entidad, sino solo con una, como mucho.
<b>1:N (Uno a muchos)</b>	Es un caso asimétrico. Ocurre cuando una entidad puede tener una instancia que se relacione con muchas instancias de la otra entidad, pero no al revés, pues cada instancia de esa otra entidad solo podrá relacionarse, como máximo, con una instancia de la primera entidad y no con varias.
<b>N:M (Muchos a muchos)</b>	Es una situación simétrica. Cada instancia de cualquier entidad puede relacionarse con dos o más instancias de la otra entidad.

A continuación, un ejemplo de los tres tipos de cardinalidades:

**1:1**



En este caso se indica que un entrenador, como máximo, dirigirá a un equipo. Y que un equipo será dirigido, como máximo, por un entrenador.

**1:N**



Este caso indica que un cliente puede generar muchas facturas, pero que cada factura pertenecerá un solo cliente.

**N:M**



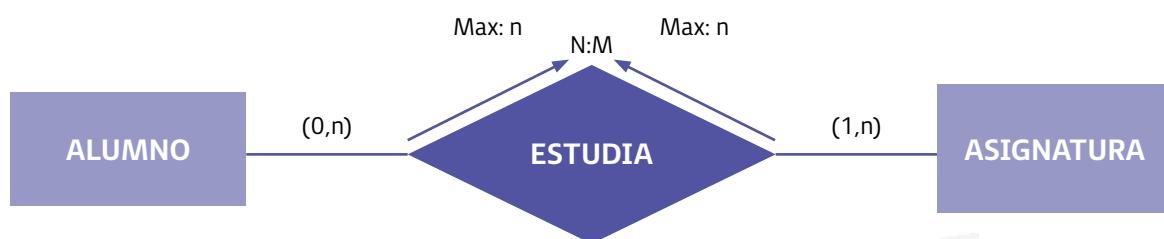
Y en este caso se indica que un alumno puede estudiar muchas asignaturas y que cada asignatura puede ser estudiada por muchos alumnos.

Si hay que poner una segunda N, la segunda N se transforma en una M (se hace así por convenio), pero su significado se mantiene invariable.

**AVISO**

## Participaciones

Las participaciones son las cardinalidades particulares de cada lado de la entidad. Representan el mínimo y máximo de relaciones que puede tener una entidad con la otra entidad. Por ejemplo:



En este caso se indica que un alumno debe estudiar, como mínimo, una asignatura y, como máximo, muchas. Esto se representa con: (1,n)

Y por otro lado, la expresión (0,n) nos indica que una asignatura puede ser estudiada por ningún alumno, como mínimo, o por muchos alumnos, como máximo.

Como se puede deducir del ejemplo anterior, la cardinalidad general de la relación vendrá dada por los máximos de las dos entidades que participan en ella.

A continuación mostramos todas las participaciones posibles:

PARTICIPACIÓN	DESCRIPCIÓN
(0,1)	Mínimo cero, máximo uno
(1,1)	Obliga a la participación
(0,n)	Mínimo cero, máximo n (indefinido)
(1,n)	Mínimo uno, máximo n (indefinido)

La notación que utilizamos para expresar la participación de un diagrama entidad-relación consiste en colocar la participación al lado de la entidad, sobre la línea de la relación.



Para facilitar su comprensión, podemos leerlo como si se tratase de una oración. De este modo, nuestra oración comienza con la primera entidad como sujeto, la relación como el verbo y la otra entidad, como el complemento directo.

### Entidades fuertes y débiles

Podemos hacer diferencia entre **dos tipos de entidades**:

- **Entidad fuerte**: tiene existencia por sí misma, es decir, está dotada de significado propio.

**ENTIDAD**

- **Entidad débil**: entidad cuyos atributos no la identifican completamente. Su participación va ligada a una relación fuerte para que esta le ayude a identificarla.

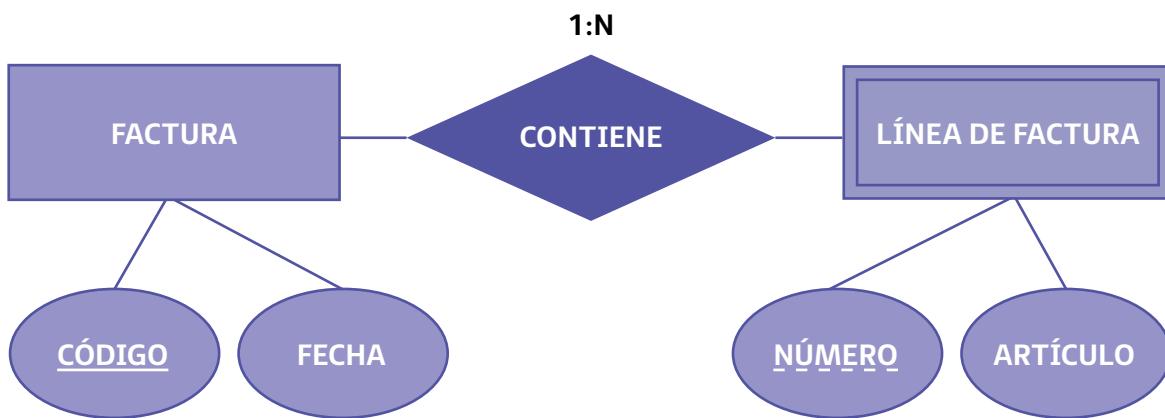
**ENTIDAD**

### EJEMPLO

Podemos poner como ejemplo de **entidad fuerte y débil** el siguiente caso práctico:

Imaginemos que tenemos una BD con la entidad *Factura* y la entidad *Línea de Factura*. Una factura tiene los atributos de *Código de factura* y *Fecha de factura*. Y la entidad *Línea de Factura* tiene como atributos *Número de línea* y *Artículo*.

Cada factura tendrá varias líneas de facturas y cada línea de factura pertenecerá solo a una factura. Es decir, por ejemplo, la factura X3466-L de 21 de mayo de 2022 puede tener cuatro líneas de factura: la 1, con leche; la 2, con manzanas; la 3, con jabón y la 4, con queso.



**Línea de Factura** es una entidad débil, dado que depende de **Factura**, que es su entidad **Fuerte**.

**Línea de factura** no nos puede asegurar que cada instancia que tenga no se repita, dado que podría ocurrir que dos facturas distintas tuviesen una línea de factura con el número “3” y con “leche” como **Artículo**.

Sin embargo, la entidad **Factura** es una entidad fuerte, dado que nunca habrá dos facturas con el mismo código.

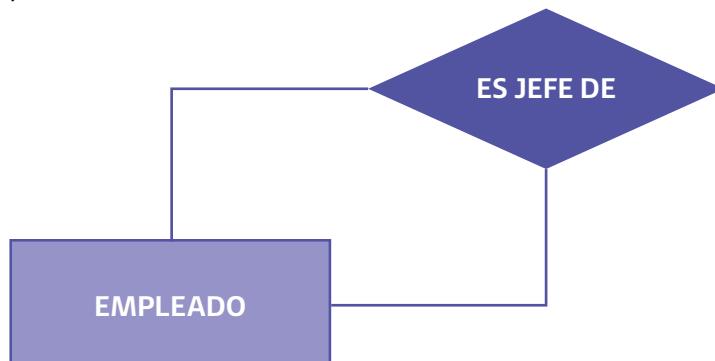
Las entidades débiles tienen identificador, que se representa con una línea subrayada discontinua. La entidad débil necesitará su identificador junto a la clave primaria de su entidad fuerte (en este ejemplo, **código de factura**) para poder distinguir de manera única cada instancia.

#### Tipos de correspondencias en las relaciones: binaria, reflexiva y otros

En los diagramas entidad-relación podemos diferenciar entre distintos tipos de relaciones:

- **Relación unaria o reflexiva:** tipo de relación en la que solo participa una entidad asumiendo diferentes roles, dependiendo del sentido de la relación. También llamadas de anillo o grado 1.

**En el ejemplo:** un empleado puede ser jefe de otros empleados.



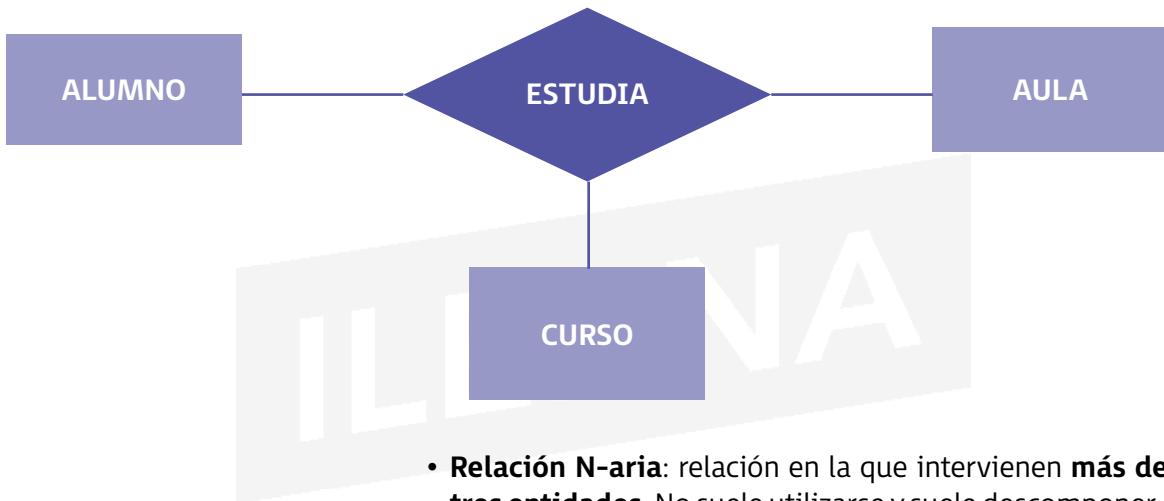
- Relación **binaria** o de grado 2: intervienen dos entidades

**En el ejemplo:** un alumno estudia una asignatura y una asignatura es estudiada por un alumno.



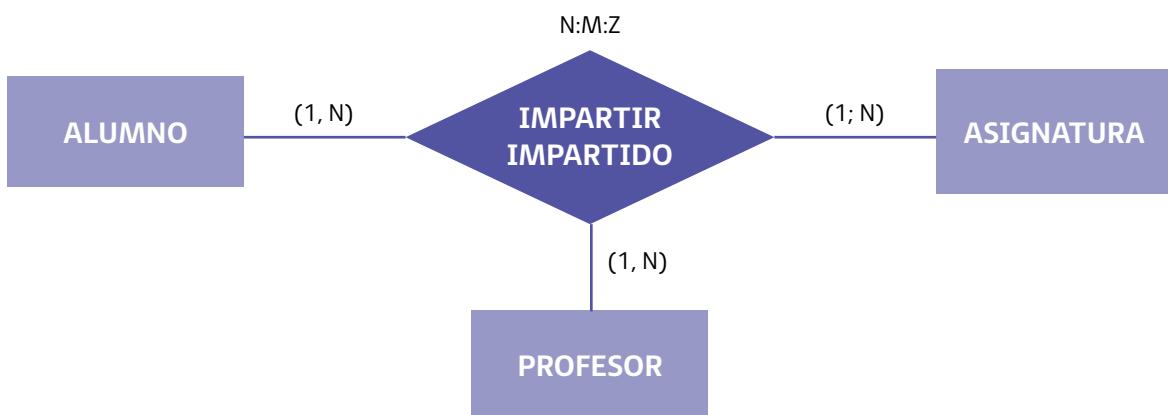
- Relaciones **ternarias** o de grado 3: intervienen tres entidades.

**En el ejemplo:** un curso puede ser impartido X veces. Un aula puede alojar X veces dichos cursos impartidos. Finalmente, una impartición (asignatura es impartida) de dicho curso en dicha aula puede tener X alumnos.



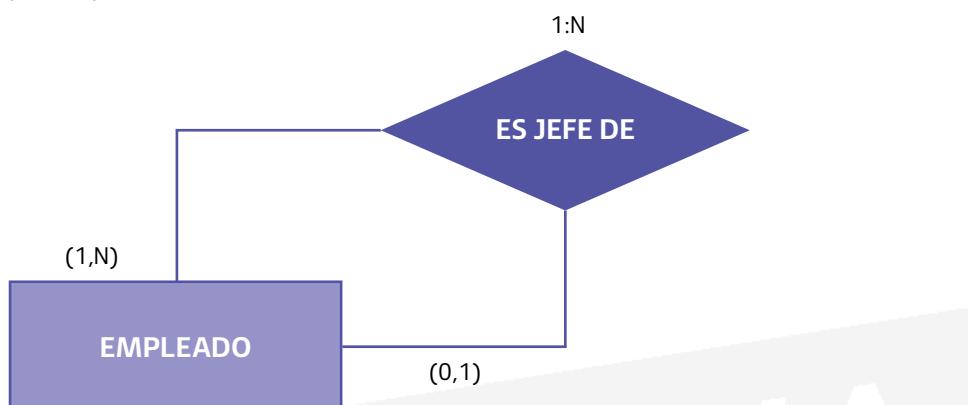
- **Relación N-aria:** relación en la que intervienen **más de tres entidades**. No suele utilizarse y suele descomponerse en relaciones de menor grado.

En la cardinalidad de **relaciones binarias** se debe coger el valor máximo de cada participación. La **cardinalidad en relaciones terciarias y N-arias** se hacen de forma similar a las binarias: debemos escoger el máximo valor de cada participación.



\*\*Establecemos (1,N) en *Alumno* porque entendemos que un *Profesor* debe impartir una *Asignatura* a al menos 1 *Alumno*, si no, dejaría de tener sentido. También podemos pensar que una asignatura puede no tener alumnos, pero entonces tampoco tendría sentido que existiera la figura de *Profesor* como (1,N), por ello en este ejemplo y supuesto, estimaremos de este modo las relaciones.

La **cardinalidad en relaciones reflexivas** se lleva a cabo cuando una entidad toma varios roles. Aunque, a la hora de calcular su cardinalidad, seguimos tomando como base el mismo principio. Es decir, tomamos los máximos de ambas participaciones.



## 2.5. MODELO ENTIDAD-RELACIÓN EXTENDIDO

Hasta ahora hemos visto el modelo entidad-relación simple, es decir, binario. Se denomina binario porque una relación conecta a dos entidades. Existe otro modelo de entidad-relación más complejo, denominado entidad-relación extendido. En este modelo no existe la limitación binaria que presenta el modelo simple.

Este modelo presentó bastantes limitaciones desde sus comienzos debido, sobre todo, a las tecnologías del momento.

Este problema se ha ido solventando con el paso de los años, hasta conseguir un nivel adecuado para los diseñadores de las BBDD. Se han incorporado todos los elementos del modelo entidad-relación con todos los conceptos de subclase y superclase, junto a los de especialización y generalización. Todos estos factores han dado lugar al modelo entidad-relación extendido.

Este modelo puede representar bastantes más restricciones en el mundo real, tales como: atributos derivados, generalización/especialización, agregación, exclusividad, exclusión, inclusividad, inclusión.

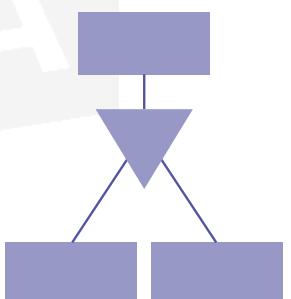
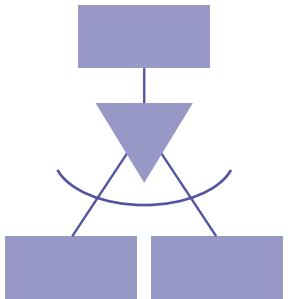
## Generalización/especialización

La **generalización** es un caso especial de relación entre diferentes tipos de entidad que llamamos *subtipos*, relacionado con otros más genéricos que definimos como *supertipos*. La relación que se establece entre ambos es del tipo: "es un" o "es un tipo de" o "is a". Esta jerarquía nos la podemos encontrar de dos formas distintas:

- **Generalización:** cuando dos o más tipos de entidades van a compartir diferentes atributos. Así, podemos deducir que va a existir una entidad supertipo que va a ser la que contenga aquellos atributos comunes a los subtipos.
- **Especialización:** cuando un tipo de entidad tiene algunos atributos que tienen sentido para algunos ejemplos, pero no para todos, es necesario definir subtipos que contengan estos atributos y dejar aquellos que sean comunes para todos en el supertipo.

En cualquier caso, podemos clasificar las entidades de generalización/especialización de dos maneras distintas: inclusiva vs exclusiva o total vs parcial.

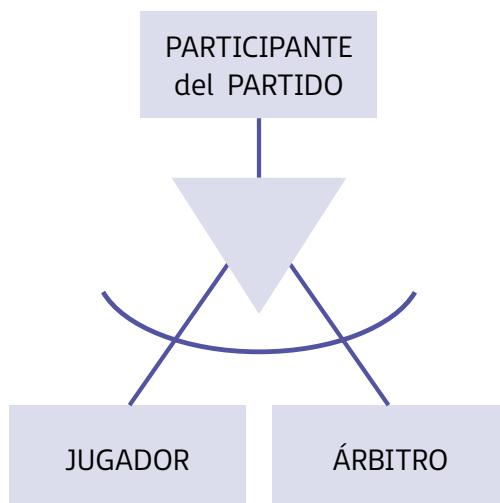
Veamos la diferencia entre inclusiva y exclusiva:

<b>INCLUSIVA (O SOLAPAMIENTO)</b> Ocurre cuando un ejemplar del supertipo puede que pertenezca a más de un subtipo.	
<b>EXCLUSIVA (O DISJUNTA)</b> Cuando un ejemplar del supertipo solo puede pertenecer a un subtipo.	

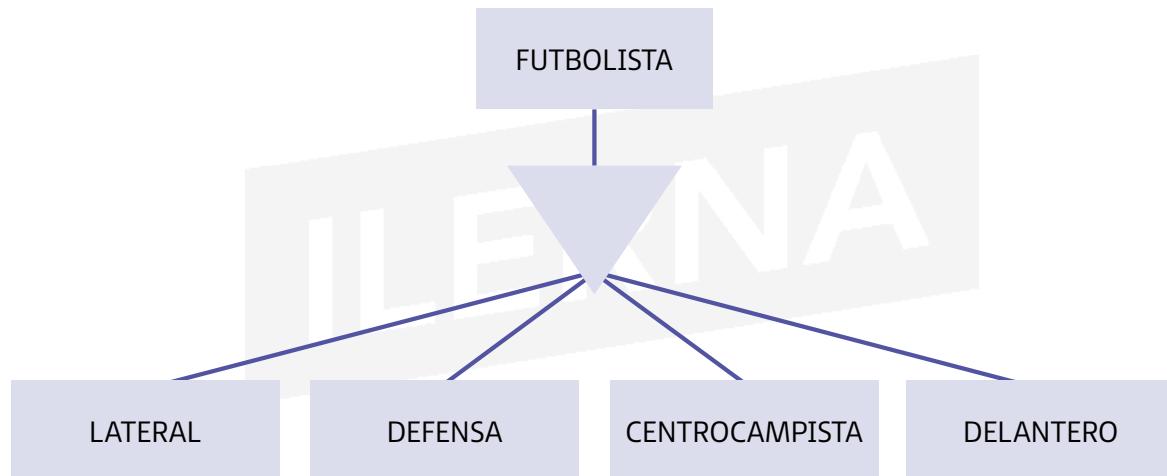
**La diferencia para representarlas en el diagrama es el arco.** La exclusiva lo lleva y la inclusiva no.

Por ejemplo, imaginemos el supertipo *Participante* de un partido, una instancia de este supertipo podría ser *Pedro Martínez*, el cual, o bien sería jugador o bien sería árbitro, pero no podría ser ambos subtipos a la vez.

Dado que no podría ser ambos subtipos a la vez, nos encontramos ante una relación exclusiva.



*Ejemplo de exclusividad.*



*Ejemplo de inclusividad.*



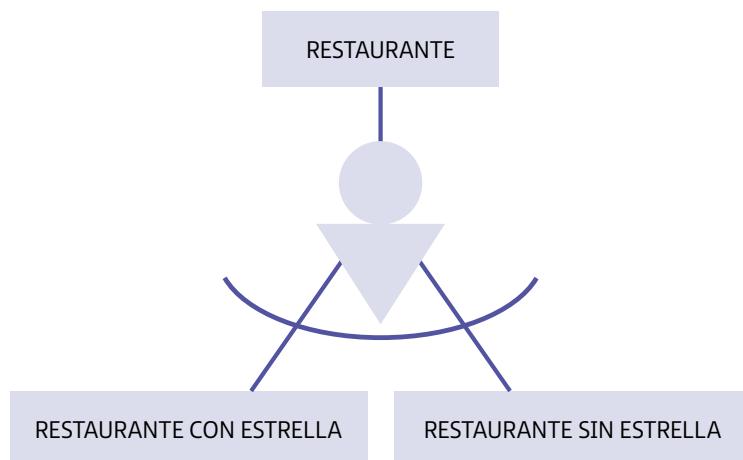
Un futbolista puede ser lateral, defensa central, centrocampista o delantero, pero también es posible que un futbolista tome un papel tanto de defensa central como de lateral durante el transcurso del mismo partido. Por tanto, el supertipo adoptaría más de un subtipo en esta instancia. Lo cual nos confirma que es un caso de inclusividad.

Tenemos la otra posible clasificación:

<b>TOTALIDAD</b> Cuando todo ejemplar del supertipo debe pertenecer a algún subtipo.	
<b>PARCIALIDAD</b> Cuando hay ejemplares del supertipo que no pertenecen a ningún subtipo.	

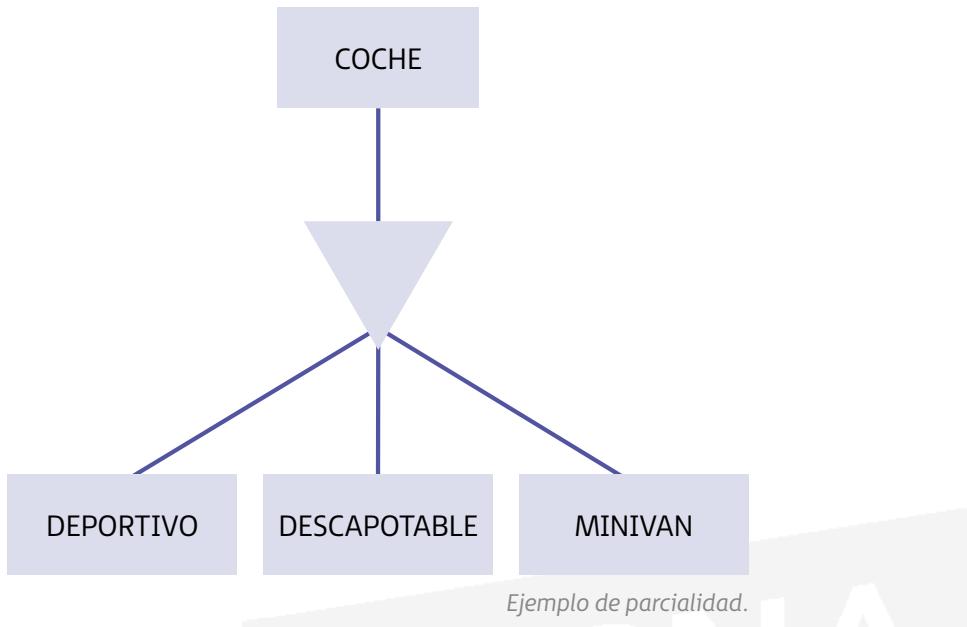
La diferencia para representarlas es el *Círculo*. En el caso de totalidad se añade dicho símbolo y en el caso de parcialidad no se añade.

Por ejemplo, imaginemos una entidad *Restaurante*. Las instancias de dicha entidad deberán tener sí o sí algún subtipo, en este caso, todos los restaurantes o bien pertenecen al subtipo *Con estrella* o bien al subtipo de *Sin estrella*, pero en ningún caso podrán no pertenecer a ninguno de los subtipos.

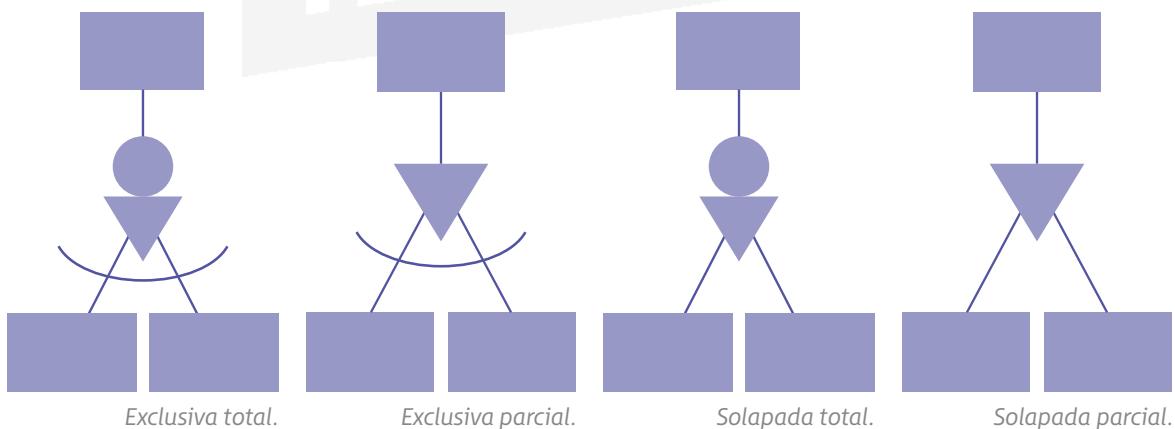


*Ejemplo de totalidad.*

Por otro lado, tenemos un ejemplo de parcialidad con el supertipo *Coche*. Una instancia de *Coche* puede pertenecer, además, al subtipo *Deportivo* o *Descapotable* o *Minivan*, pero también podría no pertenecer a ninguno de estos subtipos sin dejar de ser un coche. Por tanto es un claro ejemplo de parcialidad.



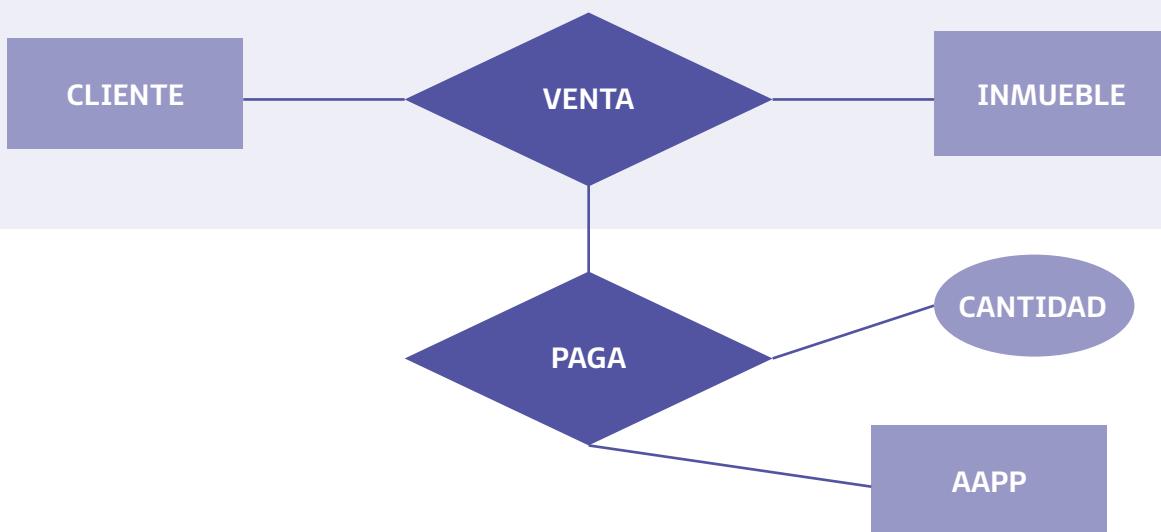
Por tanto, los diferentes tipos en los que podemos diferenciar una relación jerárquica van a ser los siguientes:



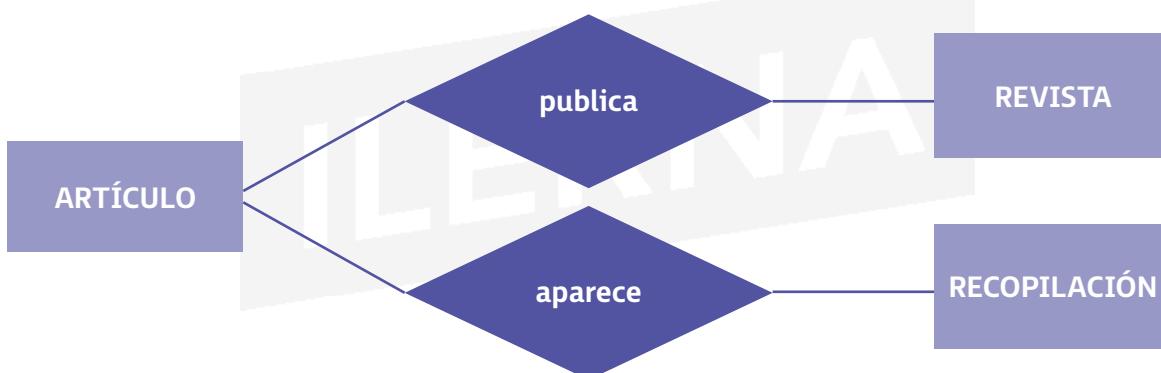
Debemos tener claro que una entidad jerárquica, sea exclusiva o inclusiva, es independiente a que sea total o parcial.

- **Agregación:** cuando deseamos que una relación formada por entidades se comporte como entidad para ser relacionada con otras entidades.

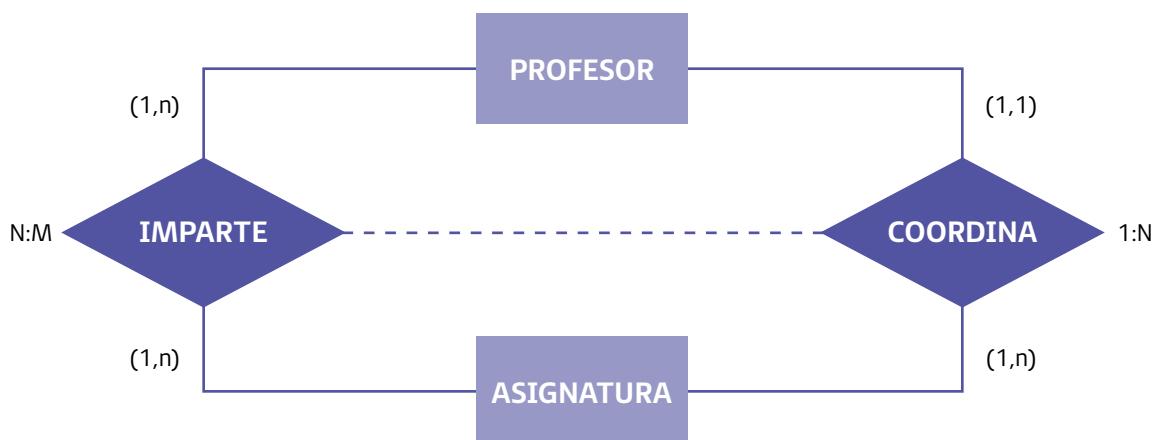
Por ejemplo, un cliente se compra un piso. Por esa venta, a la Administración Pública se le paga una cantidad. Si el cliente no se hubiera comprado ese piso, no se le pagaría dicha cantidad.



- **Exclusividad:** se da exclusividad de dos o más tipos de interrelación con respecto a una entidad cuando cada ocurrencia solo pertenece a uno de los dos tipos de interrelación.

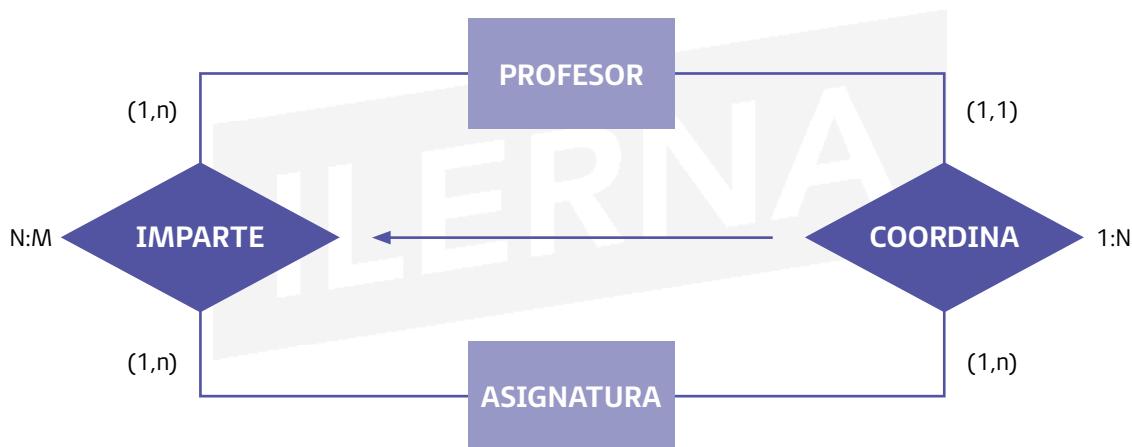


- **Exclusión:** cuando tenemos una ocurrencia de un determinado tipo de entidad, relacionada con otra ocurrencia de otro tipo de entidad diferente mediante una interrelación, no se puede relacionar con esa ocurrencia por ningún otro tipo de interrelación.

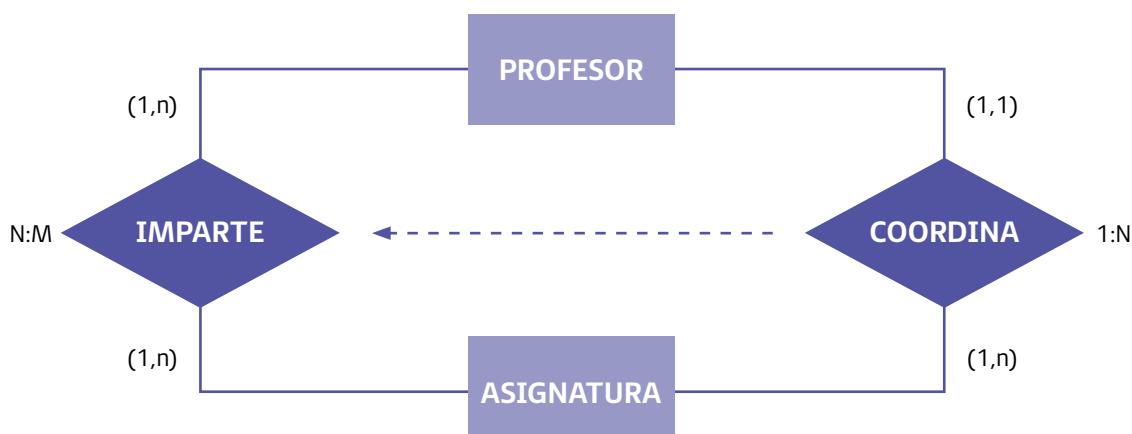




- **Inclusividad:** para que toda ocurrencia de un tipo de entidad tenga interrelación, debe formar parte, necesariamente, de otro tipo de interrelación.



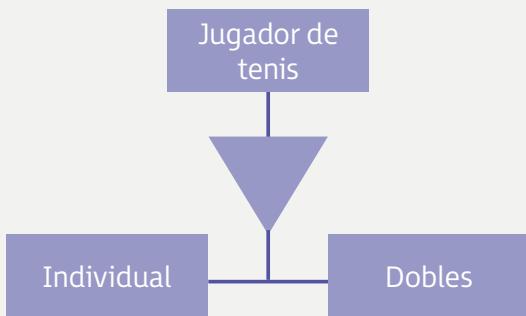
- **Inclusión:** se da el caso de inclusión cuando una ocurrencia de un tipo de entidad se relaciona con otra ocurrencia de otro tipo de entidad por una interrelación. Necesariamente se debe relacionar con esa misma ocurrencia por una interrelación diferente.





ponte a prueba

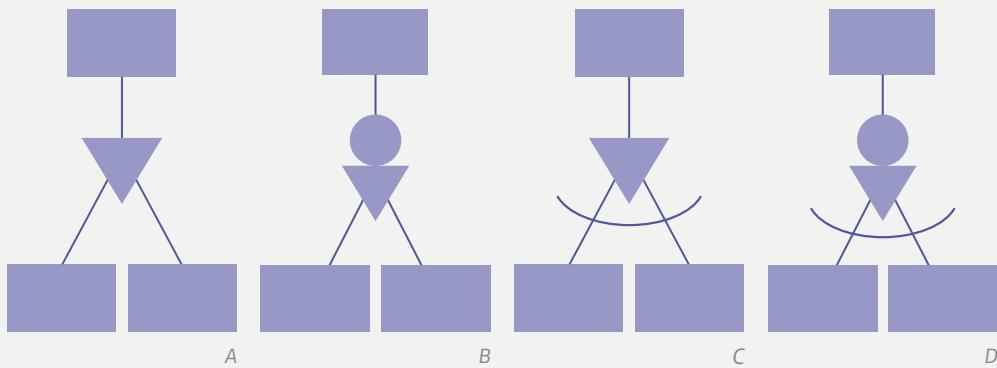
En el modelo E-R extendido, indica a qué restricción semántica pertenece el siguiente ejemplo.



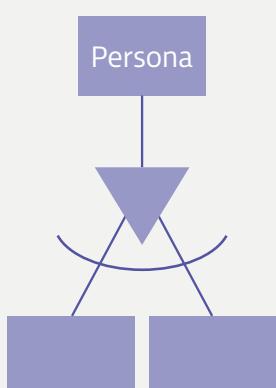
Un jugador de tenis se apunta a un campeonato donde puede participar en la modalidad de dobles, en individual o en ambas.

- a) Inclusiva.
- b) Exclusiva.
- c) Parcial.
- d) Ninguna opción es correcta.

Indica el orden correcto de cada diagrama en función del orden alfabético [A, B, C, D]



- a) Inclusiva total, inclusiva parcial, exclusiva total, exclusiva parcial.
- b) Inclusiva parcial, inclusiva total, exclusiva parcial, exclusiva total.
- c) Exclusiva parcial, inclusiva total, exclusiva total, inclusiva parcial.
- d) Inclusiva total, inclusiva parcial, exclusiva parcial, exclusiva total.



Dada la siguiente jerarquía, indica cuál de los subtipos pueden pertenecer a esta jerarquía exclusiva.

- a) Hombre, Mujer.
- b) Estudiante, Trabajador.
- c) Amigo, Padre.
- d) Anciano, Estudiante.



## 2.6. GUÍA PARA LA CONSTRUCCIÓN DE UN MODELO ENTIDAD-RELACIÓN

A la hora de realizar la construcción de un modelo entidad-relación es importante poner atención para obtener un modelo óptimo y evitar errores. Para ello, hacemos a continuación una recopilación de los siguientes pasos:

1. Leer el problema las veces que haga falta hasta entenderlo por completo.
2. Identificar los diferentes tipos de entidades.
3. Identificar los diferentes tipos de relaciones.
4. Buscar las cardinalidades adecuadas.
5. Identificar los atributos de cada tipo de entidad.
6. Identificar las claves de cada tipo de entidad.

Es importante distinguir los tipos de entidades e interrelaciones de los atributos. Los atributos deben ser atómicos y los característicos del tipo de entidad o interrelación que describan.



# 3

## MODELO RELACIONAL

## 3.1. TERMINOLOGÍA DEL MODELO RELACIONAL

La idea básica del modelo relacional consiste en representar las BBDD como un conjunto de relaciones interrelacionadas entre sí. Para entender mejor este concepto podemos visualizar cada relación como una *tabla*, donde cada *columna* es un *atributo* y cada *fila* de la tabla un *registro* o *tupla*.

Los términos principales del modelo relacional son los siguientes:

- **Tupla**: cada una de las filas de una tabla. También puede llamarse *registro*.
- **Atributo**: cada columna de las que consta una determinada tabla. Un atributo podría ser *nombre*, *fecha*, *código*...
- **Dominio**: el rango de posibles valores que se asigna inicialmente a un atributo determinado. Por ejemplo, en un atributo *código* se podría determinar como dominio los números enteros, o en un atributo *nombre* se podría establecer como dominio cualquier cadena de texto.
- **Grado**: el número de atributos de una relación.
- **Cardinalidad**: el número de tuplas de una relación, es decir, es el número de registros en la tabla.
- **BD relacional**: conjunto de relaciones normalizadas.



**ponte a prueba**

Mostrando el siguiente ejemplo de una tabla de vehículos, ¿a qué término del modelo relacional pertenecen los nombres Marca, Modelo, Año y Precio?

Marca	Modelo	Año	Precio
Ford	Focus	2019	21.000 €
Opel	Astra	2014	5.000 €
VW	Golf	2007	3.000 €

- Atributo.
- Tupla.
- Título.
- Dominio.



## 3.2. CONCEPTO DE RELACIÓN. PROPIEDADES Y RELACIONES

El elemento principal de este modelo es la **relación** que va a utilizarse para representar las diferentes entidades e incluso algunas relaciones de los diagramas entidad-relación. A las filas de una relación se las denomina **tuplas**. Cada una de las tuplas tiene una serie de atributos con un determinado valor.

NIF	Nombre	Apellidos	Teléfono	Fecha nacimiento	Fecha alta
74512593P	María	Rodríguez Moreno	635984754	13/07/1980	21/09/2016
48915264M	José	Martín Delgado	650221416	09/02/1975	14/10/2013
21733456Z	Judit	Casas Pérez	679548979	22/06/1963	10/05/2015
32564542F	Marta	Alonso Sánchez	643315642	07/12/1992	22/06/2016

Relación socio.

## 3.3. ATRIBUTOS Y DOMINIO DE LOS ATRIBUTOS

Como hemos definido anteriormente, el concepto de atributo hace referencia a las columnas de una relación (tabla). El atributo representa la propiedad de un registro o tupla. Por ejemplo, la edad de cada persona, el código de cada cliente o el precio de cada artículo en una tienda.

Cada atributo posee un dominio, es decir, cada atributo tiene un rango de valores posibles que podemos otorgar a cada fila de la tabla. Siempre asociamos la idea de dominio a la de tipo de datos, entre los que podemos diferenciar los siguientes: números enteros y decimales, cadenas de caracteres, fechas y horas, valores lógicos y objetos.

Los datos se organizan en relaciones, y una relación R está definida sobre un conjunto de dominios D<sub>1</sub>, D<sub>2</sub>... D<sub>n</sub> y consta de:

- **Cabecera:** es un conjunto finito de pares (atributo: dominio):  
{(A<sub>1</sub>:D<sub>1</sub>), (A<sub>2</sub>:D<sub>2</sub>), ... (A<sub>n</sub>:D<sub>n</sub>)}  
Cada atributo se corresponde con un único dominio (no al contrario) no habiendo dos atributos que se tengan el mismo nombre.
- **Cuerpo:** conjunto de variables de tuplas.

Cada tupla, es un conjunto de pares atributo: valor.

$\{(A_1:V_1), (A_2:V_2), \dots, (A_n:V_n)\}$  con  $i=1, 2, \dots, m$  donde  $m$  es la cardinalidad de  $R$ . Para cada par, se cumple que  $V_i$  pertenece a  $D_j$ .

Veamos el siguiente ejemplo:

Nombre	Función
Dpto1	Ventas
Dpto2	Estadística
Dpto3	Compras

Relación departamento.

Cabecera: { (Nombre: NOMBRE), (Función: FUNCIÓN) }

Cuerpo: una tupla { (Nombre: Dpto1), (Función: Ventas) }

Las **propiedades** que deben cumplir todas las relaciones son las siguientes:

- Cada relación debe tener asignado un nombre distinto al de las demás relaciones. Por ejemplo: no puede haber dos relaciones que se llamen *cliente*.
- Cada relación tiene un número fijo de atributos para todas las tuplas.
- Por ejemplo, si una relación tiene como atributos *DNI*, *Nombre* y *Teléfono*, no puede haber ninguna fila, es decir, ninguna tupla de esa tabla que posea cuatro atributos.
- Cada atributo tiene un único dominio.

Por ejemplo, el atributo *edad* tiene como dominio solo los números enteros positivos.

- No importa el orden de los atributos.

Por ejemplo, estas dos relaciones son equivalentes:

DNI	APELLIDOS	NOMBRE	DNI	NOMBRE	APELLIDOS
48312543	Belmonte García	José Luis	48312543	José Luis	Belmonte García
45211742	Ramírez Pardo	Sandra	45211742	Sandra	Ramírez Pardo
34224566	Josué Ramos	Andreu	34224566	Andreu	Josué Ramos

- En una relación no puede haber dos atributos con igual nombre.

Por ejemplo, no puede haber dos atributos *apellido* en una relación. En todo caso podríamos llamarlos *apellido 1* y *apellido 2*.

- Valores de atributos atómicos, es decir, en cada tupla, cada valor del atributo correspondiente toma un solo valor del dominio. Por ejemplo, en el atributo *código* no puede haber una tupla con dos códigos, es decir, dos valores, sino que siempre habrá solo uno.

- Cada tupla es única, distinta a las demás.

La clave primaria se encargará de que nunca se repitan todos los valores de dos tuplas o registros.

- No importa el orden de las tuplas.

Por ejemplo, estas relaciones son equivalentes:

CÓDIGO	NOMBRE ARTÍCULO	PRECIO
a4F13	Leche desnatada	1,14
b2s64	Mermelada	2,40
c3g32	Pack servilletas	0,95

CÓDIGO	NOMBRE ARTÍCULO	PRECIO
b2s64	Mermelada	2,40
a4F13	Leche desnatada	1,14
c3g32	Pack servilletas	0,95



### 3.4. CONCEPTO Y TIPOS DE CLAVE: CANDIDATAS, PRIMARIAS, AJENAS, ALTERNATIVAS

Una clave (también llamada *llave*) es un atributo o un conjunto de atributos que identifican de manera única una entidad.

Como hemos visto anteriormente, en una relación no puede haber tuplas repetidas, lo que significa que los registros o tuplas deben diferenciarse unos de otros en al menos un atributo o un conjunto de atributos.

Existen dos maneras de clasificar las claves o llaves, en función de su cardinalidad o en función del tipo de clave que sean.

Según su cardinalidad:

- **Claves simples** (o atómicas): están compuestas por un solo atributo.
- **Claves compuestas**: están compuestas por dos o más atributos.

Y según el tipo de clave:

- **Clave primaria (o principal)**: es el atributo o conjunto de atributos mínimo que identifica de manera única las tuplas de una relación, de modo que ninguna tupla repita el atributo o los atributos de su clave primaria. En caso de que existan varios atributos o conjuntos de atributos que cumplan el requisito de clave primaria, el diseñador de la BD deberá elegir el más adecuado.

Por ejemplo, en la siguiente relación *Personas*, la clave primaria es el atributo *DNI*, dado que este nunca se va a repetir. Sin embargo, el atributo *nombre* o *apellidos* podría repetirse, como ocurre en el ejemplo, pues existen dos registros con nombre *Andreu* y también dos registros con apellidos *Josué Ramos*.

DNI	NOMBRE	APELLIDOS
48312543	José Luis	Belmonte García
45211742	Sandra	Ramírez Pardo
34224566	Andreu	Josué Ramos
24576232	Andreu	García Ruiz
12867282	Begoña	Josué Ramos

- **Clave candidata:** son aquellos atributos o conjuntos de atributos que optan a ser la clave primaria de la relación.

Por ejemplo, en la siguiente relación *Cliente* existen dos claves candidatas, *Código* y *DNI*. Esto ocurre porque tanto el código como el DNI del cliente identificarán de manera única a cada una de las tuplas, es decir, a cada uno de los clientes.

CÓDIGO de CLIENTE	DNI de CLIENTE	Nombre	Apellidos
J3356	48202344	Pedro Jesús	Martínez Ferrer
K7494	23762922	Ana María	Zuriala Mínguez
P4933	33452892	Antonio	Correa Sempere

- **Claves alternativas:** son aquellas claves candidatas que no fueron elegidas para ser la clave primaria de la relación.
- **Claves ajena(s) (o foránea(s)):** atributo o conjunto de atributos de una relación que son clave primaria de otra relación distinta.

En el ejemplo 1, podemos observar las tablas *País* y *Ciudad*. El atributo *país* de la tabla *ciudad* es un ejemplo de clave ajena a *código país* de la tabla *País*.

En el ejemplo 2, podemos observar las tablas *País* y *Ciudad*. *Código país* es clave primaria en la tabla *País* y actúa como clave foránea en la tabla *Ciudad*.

1

CÓDIGO PAÍS	NOMBRE PAÍS	POBLACIÓN
1001	Estados Unidos	327 000 000
1057	Colombia	49 000 000
1058	Venezuela	31 000 000

2

CÓDIGO CIUDAD	NOMBRE CIUDAD	PAÍS
101	Caracas	1058
102	Bogotá	1057
35	Washington	1001

**ponte a prueba**

**¿Cómo se conoce al conjunto de atributos de una relación que son clave primaria de otra relación distinta y que por causas del diseño deben estar relacionadas?**

- a) Clave alternativa.
- b) Clave foránea.
- c) Clave candidata.
- d) Clave primaria.

**Dada la siguiente tabla, ¿cuál de los siguientes campos es la clave primaria?**

DNI	NOMBRE	APELLIDOS
48123456S	Andrés	García Pardo
47987654B	Andrés	Fuentes Ruiz
45652365M	María	Ramos Fernández

- a) DNI.
- b) NOMBRE.
- c) APELLIDOS.
- d) Cualquiera de las tres podría ser clave primaria.

**Dada la siguiente tabla, ¿cuáles de los siguientes campos optan para ser clave candidata?**

CÓDIGO CLIENTE	DNI	NOMBRE	APELLIDOS
A12S35	48123456S	Andrés	García Pardo
A25S32	47987654B	Andrés	Fuentes Ruiz
B56A25	45652365M	María	Ramos Fernández

- a) Código cliente y DNI.
- b) Nombre y apellidos.
- c) DNI y apellidos.
- d) Código cliente y apellidos.

**Dadas las siguientes tablas, ¿cuál de los siguientes atributos son clave foránea?**

Cliente		
Código cliente	Nombre	Apellidos
10256	Arturo	García Pomares
10257	Andrea	López Rosales

Pedido		
Código pedido	Código cliente	Fecha
17852147	10256	08/05/2020
15236547	10257	15/08/2020

- a) Código cliente de la tabla Pedido.
- b) Código cliente de la tabla Cliente.
- c) Fecha de la tabla Pedido.
- d) Apellidos de la tabla Cliente.

## 3.5. VALORES NULOS

El concepto de valor nulo hace referencia a la **ausencia de valor** en el atributo de una tupla. Los valores nulos nunca podrán aparecer en una clave primaria.

No se debe confundir valor nulo con el valor 0. Por ejemplo, imaginemos la tabla *Artículo*, la cual posee un atributo llamado *ventas*, que indica la cantidad de veces que un artículo ha sido vendido. No significa lo mismo que el valor sea 0, es decir, que ese artículo nunca haya sido vendido, que la ausencia de valor de ese dato. En realidad significa que desconocemos el número de veces que ese artículo ha sido vendido.

CÓDIGO	NOMBRE ARTÍCULO	VENTAS
b2s64	Mermelada	292
a4F13	Leche desnatada	0
c3g32	Pack servilletas	NULL

En este ejemplo podemos ver la diferencia de valor nulo (null) y 0.

Durante la normalización de una BD comprobaremos que los valores nulos no están permitidos. No obstante, en la práctica es habitual permitir la existencia de valores nulos en la programación típica de las BBDD.

## 3.6. REGLAS DE INTEGRIDAD: DE ENTIDAD Y REFERENCIAL

La **integridad** es uno de los objetivos que deben cumplir los SGBD. Gracias a la integridad de los datos se dificulta la pérdida de estos.

Parte de este objetivo puede cumplirse si se cumple la coherencia y veracidad de la información de la BD. Cabe tener en cuenta que las operaciones de inserción, borrado y modificación de tuplas pueden afectar a la integridad. Si el SGBD no asegura desde un principio la integridad, esta debe ser garantizada por las aplicaciones.

Para mantener la integridad de la BD hay que tener en cuenta algunas restricciones, como por ejemplo, cada vez que se inserte o se modifique algún valor del atributo.

Las principales **restricciones** que deben cumplirse en todas las BBDD son:



- **Reglas de la integridad de la entidad**

- Ningún atributo (o campo) que forme parte de la clave primaria puede ser nulo (Nulo= ausencia de valor, valor desconocido).
- Se deben realizar comprobaciones en cada inserción y modificación.

- **Reglas de la integridad referencial**

- Los valores de una clave ajena o coinciden con un valor de la clave primaria a la que referencian o son nulos.
- Un valor nulo en la clave ajena significa que la interrelación entre las entidades es opcional.
- Si la clave ajena nunca toma valores nulos, la interrelación es obligatoria.
- Para cada clave ajena se debe especificar si debe o no tomar valor nulo y determinar las consecuencias de las operaciones de inserción, borrado y modificación sobre las tuplas de la relación referenciada.
- Si a la BD llega una petición para realizar una operación ilegal, debemos rechazar la operación o aceptarla y conducirla hasta conseguir un estado legal.
- Debemos seguir una serie de reglas para mantener la integridad referencial.

### Nulos no permitidos

No se admiten valores nulos de la clave ajena. Tienen que coincidir siempre con un valor de la clave primaria.

### Restringido

Un valor de clave primaria no puede ser modificado ni borrado si existe alguna tupla en otra R que lo contenga como clave ajena.

### Transmisión en cascada

Si borramos o modificamos una tupla de R que contiene la clave primaria referenciada en otra R diferente por una clave ajena, se producirá el borrado o modificación en cascada de todas las tuplas de la otra R que contengan esa clave como ajena.

### Puestas a nulos

El borrado o modificación de una tupla de una R que contiene la clave primaria referenciada en otra R por una clave ajena producirá la puesta a nulo del valor de la clave ajena en todas las tuplas de la otra R que contengan esa clave como ajena.

### Puesta a un valor por defecto

El borrado o modificación de una tupla de una R que contiene la clave primaria referenciada en otra R por una clave ajena, producirá la puesta a un valor por defecto de la clave ajena en todas las tuplas de las R que contengan esa clave como ajena.

## 3.7. TRADUCCIÓN DEL MODELO ENTIDAD-RELACIÓN AL MODELO RELACIONAL

Antes de empezar con la traducción, debemos tener presente que durante el proceso de transformación del modelo entidad-relación al modelo relacional puede perderse parte de la semántica representada en el modelo entidad-relación. Por ejemplo, en el modelo entidad-relación existen entidades y relaciones que son elementos claramente distintos. Sin embargo, en el modelo relacional todo se traduce a tablas, es decir, a relaciones, que son el elemento principal del modelo relacional.

Para la transformación del modelo entidad-relación al modelo relacional hay que tener en cuenta algunos principios básicos:

- Las entidades del modelo entidad-relación se traducen en tablas en el modelo relacional.
- Las relaciones del modelo entidad-relación se convertirán a tablas, o no, en función del caso correspondiente.
- Los atributos de una entidad en el modelo entidad-relación suelen transformarse en los atributos de las tablas en el modelo relacional.

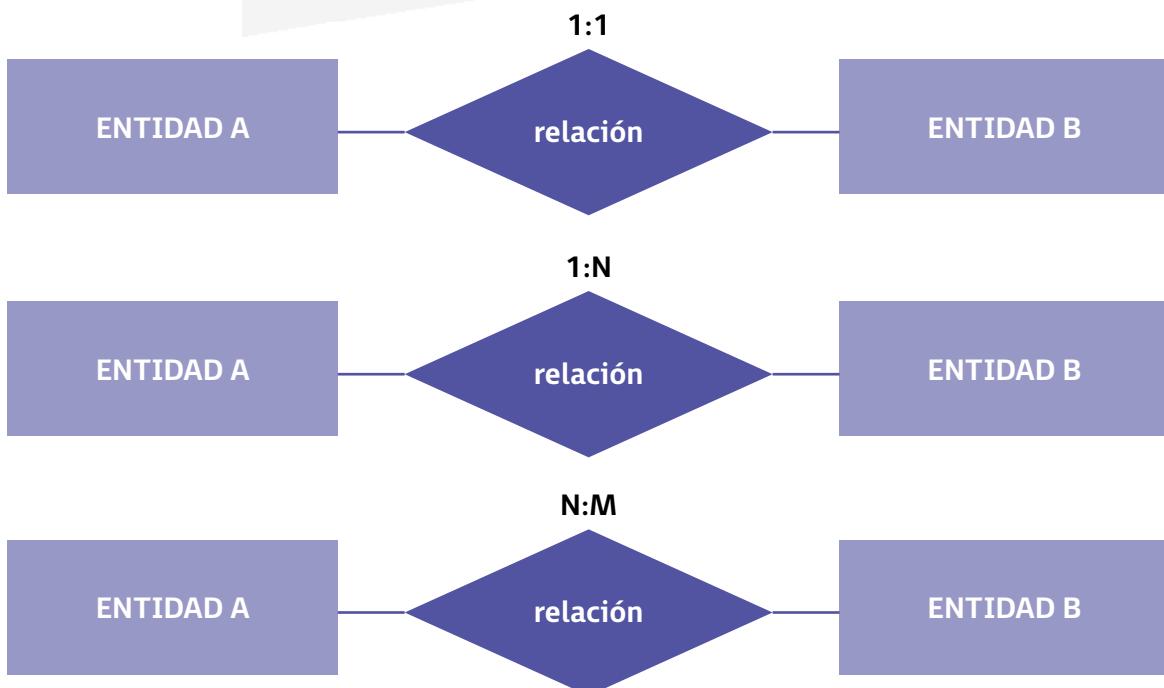
Las tablas del modelo relacional normalmente heredan las nomenclaturas usadas en el modelo entidad-relación.

- Convertir todas las relaciones del modelo entidad-relación en binarias nos facilitará la tarea. Es decir, convertir el modelo E/R extendido en E/R simple.
- Respecto a los atributos de las entidades:
  - Los atributos clave de las entidades en el modelo entidad-relación pasan a ser las claves primarias de las tablas/relaciones en el modelo relacional.
  - El resto de atributos pasan a ser las columnas de las tablas. Los atributos no opcionales, es decir, a los atributos que en cada instancia (fila) deben tener obligatoriamente un valor, se les debe aplicar la restricción *notnull*, y a las claves alternativas, la restricción *unique*.
  - Para mayor eficiencia en las futuras consultas a la BD, los atributos derivados en el modelo entidad-relación también pueden transformarse en columnas en el modelo relacional.

A continuación, vamos a especificar cada uno de los posibles casos que nos podemos encontrar al pasar del modelo entidad-relación al modelo relacional.

En general, en el modelo entidad-relación podemos encontrarnos con tres casos diferentes:

- Una relación con cardinalidad 1:1
- Una relación con cardinalidad 1:N
- Una relación con cardinalidad N:M



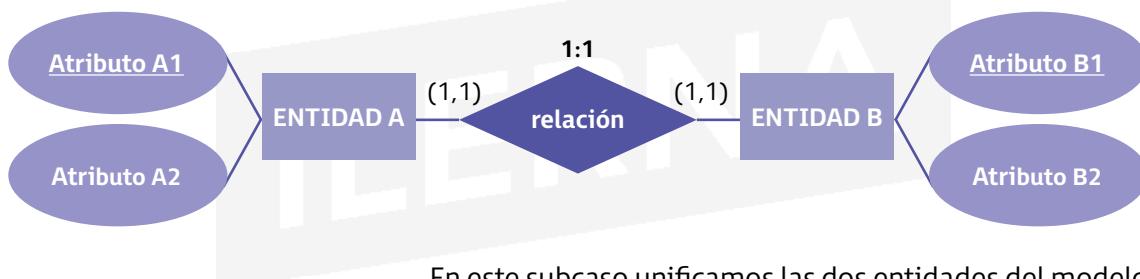
Dentro de cada uno de los tres casos, podemos diferenciar distintos subcasos que veremos a continuación.



### 3.7.1. CASO DE CARDINALIDAD TIPO 1:1

En este caso, podemos encontrar 3 subcasos:

- **Cuando las participaciones son (1,1) y (1,1):**

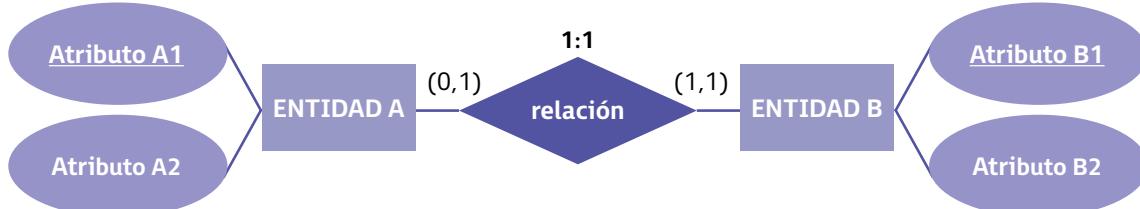


En este subcaso unificamos las dos entidades del modelo entidad-relación a una sola tabla en el modelo relacional, de modo que los atributos de ambas tablas (y los atributos de la relación, si los hubiese, pasan a ser atributos (columnas) de la nueva tabla:

ENTIDAD\_AB (AtributoA1, AtributoB1, Atributo A2, Atributo B2)

Como podemos observar, hemos elegido el "AtributoA1" como la clave primaria de la nueva tabla, pero debemos tener en cuenta que el "AtributoB1" también podría haber sido elegido la clave primaria. Por tanto, en este caso el "AtributoA1" pasa a ser la clave primaria y el "AtributoB1" pasa a ser clave alternativa.

- **Cuando las participaciones son (0,1) y (1,1):**

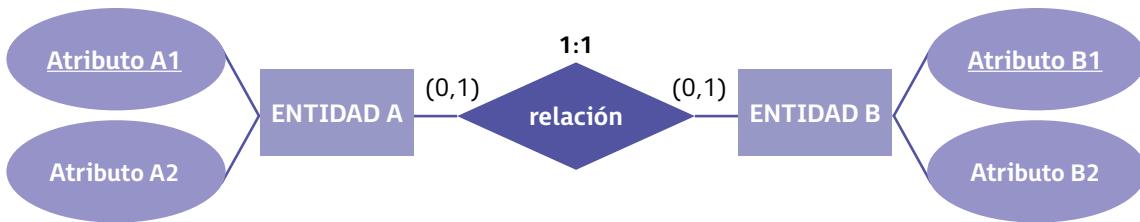


En este subcaso se crearán dos tablas, una por cada entidad, de tal modo que en la nueva tabla de la entidad que en el diagrama del modelo entidad-relación estaba en la parte del (0,1) heredará la clave primaria de la otra entidad, convirtiéndose esta en clave ajena tal que así:

Entidad\_A (AtributoA1, AtributoA2, **AtributoB1\_FK**)

Entidad\_B (AtributoB1, AtributoB2)

- **Y, por último, cuando las participaciones son (0,1) y (0,1):**



En este subcaso, cada entidad del modelo entidad-relación pasa a ser una tabla en el modelo relacional y, además, creamos otra nueva tabla para la relación entre ambas entidades. Esta nueva tabla se formará con las dos claves primarias de ambas entidades, teniendo que elegir a una de ellas como clave primaria de esta nueva tabla, quedando la otra como clave alternativa, tal que así:

Entidad\_A (AtributoA1, AtributoA2)

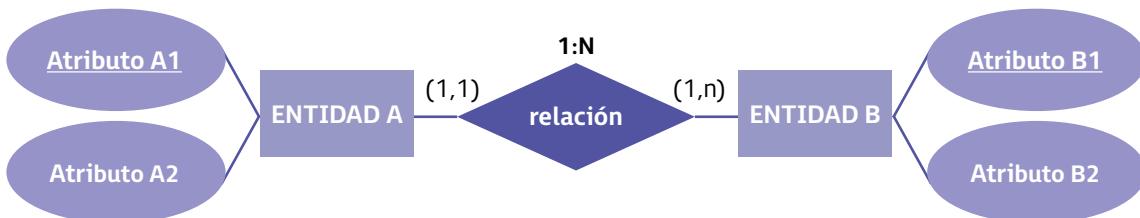
Entidad\_B (AtributoB1, AtributoB2)

Entidad\_AB (AtributoA1\_FK, AtributoB1\_FK)

### 3.7.2. CASO DE CARDINALIDAD TIPO 1:N

Aquí podemos encontrar dos subcasos:

- **Cuando las participaciones son (1,1) y (1,n):**



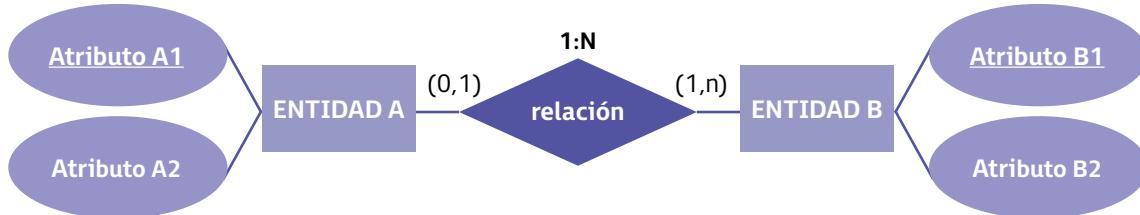
En este subcaso, se creará una tabla por cada entidad, y la tabla de la entidad que está en la parte del (1,n) heredará como clave ajena la clave primaria de la otra entidad. Los

atributos de la relación, si los hubiese, se propagarían a la entidad con cardinalidad (1,n).

Entidad\_A (AtributoA1, AtributoA2)

Entidad\_B (AtributoB1, AtributoB2, AtributoA1\_FK)

- **Cuando las participaciones son (0,1) y (1,n):**



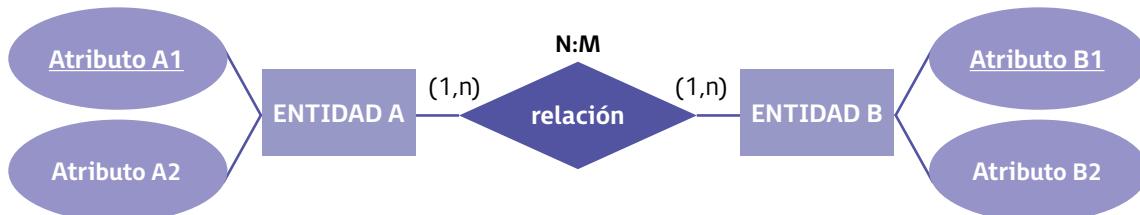
En este subcaso se creará una tabla por cada entidad y además una nueva tabla para la relación. La nueva tabla tendrá las dos claves primarias de ambas entidades como ajenas, siendo la clave primaria de la tabla relación la misma que estuviese en la entidad con la cardinalidad (1,n), es decir, en la parte del (1,n).

Entidad\_A (AtributoA1, AtributoA2)

Entidad\_B (AtributoB1, AtributoB2)

Entidad\_AB (AtributoA1\_FK, AtributoB1\_FK)

### 3.7.3. CASO DE CARDINALIDAD TIPO N:M



En este subcaso se creará una tabla por cada entidad y, además, una nueva tabla de la relación que estará formada por las dos claves primarias de las entidades, siendo ajenas en la nueva tabla. La clave primaria estará formada por ambas claves.

Entidad\_A (AtributoA1, AtributoA2)

Entidad\_B (AtributoB1, AtributoB2)

Entidad\_AB (AtributoA1\_FK, AtributoB1\_FK)



**ponte a prueba**

**¿Cuál de los siguientes casos podemos encontrarnos al pasar del modelo entidad-relación al modelo relacional?**

- a) Cuando la cardinalidad general es 1:1.
- b) Cuando la cardinalidad general es N:M.
- c) Cuando la cardinalidad general es 1:N.
- d) Todas las opciones son correctas.

**Indica la participación que falta en este ejemplo.**

*Un libro puede estar escrito por uno o varios autores, un autor puede escribir ninguno o varios libros.*



- a) (1,1)
- b) (1,n)
- c) 1:N
- d) (n,m)

**Indica la participación que falta en este ejemplo.**

*Un jugador de baloncesto únicamente puede jugar en un equipo y un equipo estará formado siempre por varios jugadores.*



- a) (1,n)
- b) (n,m)
- c) (0,n)
- d) Ninguna opción es correcta.



# 4

## NORMALIZACIÓN

## 4.1. CONCEPTO DE NORMALIZACIÓN, DEPENDENCIAS FUNCIONALES Y SUS TIPOS

La normalización es una técnica ideada por Edgar Frank Codd que consiste en aplicar unos requisitos y restricciones a las tablas que obtenemos en el modelo relacional tras haber ejecutado el proceso del paso del modelo entidad-relación al relacional.

Tras la normalización obtenemos una BD más eficiente, concretamente los objetivos de la normalización son los siguientes:

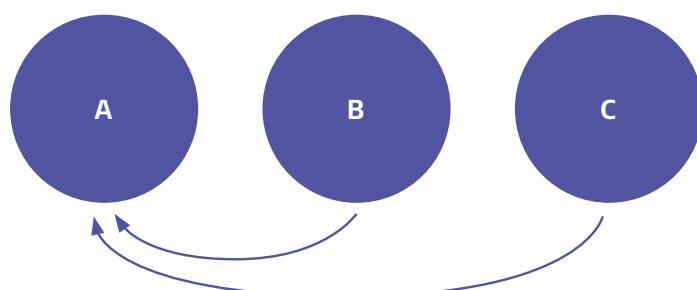
- Eliminar la redundancia en los atributos.
- Eliminar posibles problemas en las interrelaciones de los atributos.
- Eliminar la inconsistencia de datos.
- Garantizar la integridad referencial.
- Permitir acciones de inserción, modificado y borrado más seguras.
- Facilitar la inserción de nuevos tipos de datos.

Para entender la normalización se ha de tener claro el concepto de dependencia entre atributos y los tipos de dependencia existentes.

Si X e Y son dos subconjuntos de atributos de una relación, diremos que el atributo Y tiene dependencia funcional del atributo X, o dicho de otra manera, X determina a Y, si cada valor de X tiene asociado un único valor de Y. Esta dependencia se representa de la siguiente forma: **X → Y**.

Existen cuatro tipos de dependencias:

1. **Dependencia funcional:** ocurre cuando un atributo depende directamente de la clave primaria.



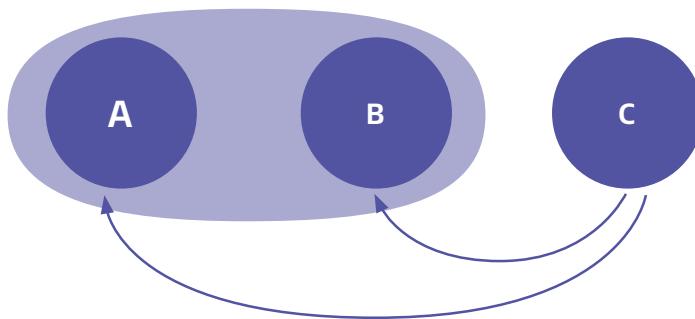
Ejemplo de dependencia funcional:

Imaginemos la relación **Alumno (Código, Nombre, Apellido, Curso)**. Podemos observar que el atributo

*Código* es la clave primaria y *Nombre, Apellido* y *Curso* son atributos que no forman parte de la clave primaria.

En este caso el atributo *Nombre* tiene una dependencia funcional de la clave primaria: **Código → Nombre**. De modo que para un valor concreto del código obtendremos un valor concreto del *Nombre*. Por ejemplo, para el código de Alumno X5432 podríamos tener el nombre de *Mario*.

2. **Dependencia funcional completa:** se da cuando un atributo depende de la totalidad de la clave (casos clave compuestos por más de un atributo).

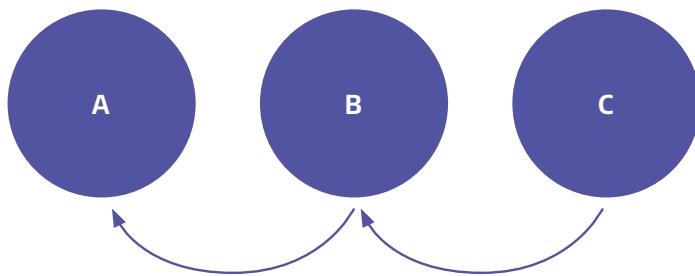


Ejemplo de dependencia funcional completa:

Imaginemos la relación **Calificación (Alumno, Asignatura, Nota, Comentarios)**. Observamos que la clave primaria es *Alumno* y *Asignatura* (un alumno puede tener varias calificaciones y una asignatura puede ser calificada para varios alumnos, por eso la clave es compuesta y no simple: son necesarios los dos atributos para identificar únicamente cada registro) y *Calificación* y *Comentarios* son atributos que no forman parte de la clave. Se representaría así: **Alumno, Asignatura → Nota**.

Por tanto, *Nota* tiene dependencia funcional completa de la clave primaria, ya que depende de la totalidad de la clave primaria y no de una parte de ella. Es decir, *Nota* depende del alumno y la asignatura. Por ejemplo, el alumno *Mario* en la asignatura de *Matemáticas* tiene la calificación de 7,5.

3. **Dependencia funcional transitiva:** un atributo depende de otro atributo no clave, que presenta una dependencia funcional con la clave.

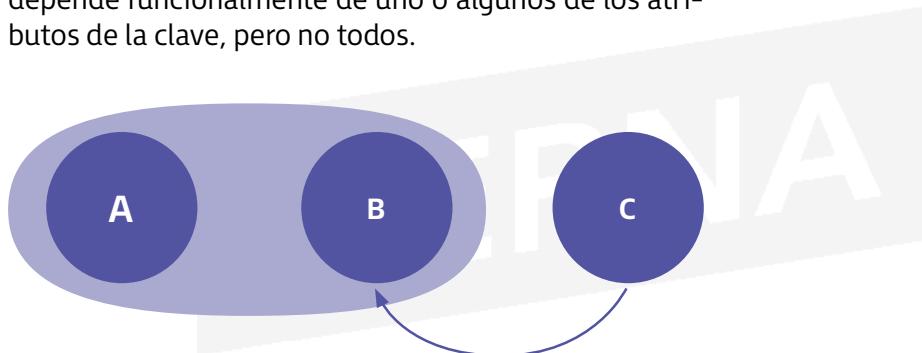


Ejemplo de dependencia funcional transitiva:

Imaginemos la relación **Libro (ISBN, Autor, año, Título, Nacionalidad del Autor)**. Observamos que la clave primaria es *ISBN* y el resto de atributos son no claves. Observamos también que la nacionalidad del autor depende del autor y que el autor depende del *ISBN* del libro (*ISBN* es un código único que se asigna a cada libro). Por tanto, existe una dependencia funcional transitiva entre *Nacionalidad del autor* e *ISBN*, dependencia transitiva que pasa por *Autor*. Se representa así: **ISBN → Autor y Autor→ Nacionalidad del Autor.**

De tal modo que, si por ejemplo, con el *ISBN* 10:8447306194 obtenemos el autor *Gabriel García Márquez* y con el autor *Gabriel García Márquez* obtenemos la nacionalidad *colombiana*, por transitividad podemos concluir que el *ISBN* 10:8447306194 nos indica la nacionalidad *colombiana*.

4. **Dependencia parcial:** un atributo no depende de la totalidad de atributos que forman la clave (en los casos de claves compuestas por más de un atributo), sino que depende funcionalmente de uno o algunos de los atributos de la clave, pero no todos.



Ejemplo de dependencia parcial:

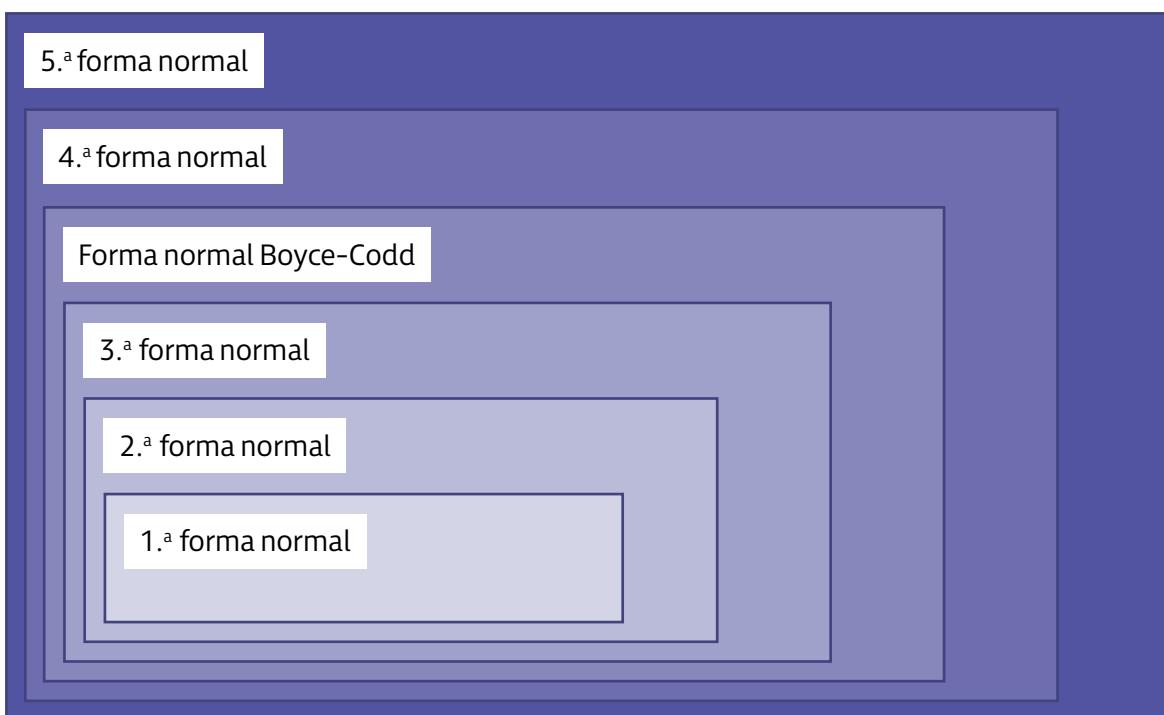
Imaginemos una relación sobre torneos anuales de tenis: **Torneo (Nombre del Torneo, Año, Ganador, Finalista, Resultado, Descripción del Torneo)**. Observamos que la clave primaria está formada por el nombre del torneo y el año, dado que un mismo torneo se puede jugar en diferentes años y en un año se juegan varios torneos. Pero un torneo concreto de un año concreto solo se jugará una vez y, por tanto, solo tendrá un ganador, un finalista, etcétera. De ese modo, como cada torneo se juega cada año, para identificar de manera única un torneo concreto necesitamos una clave primaria compuesta por el nombre del torneo y el año.

Por otro lado, tenemos el atributo no clave *Descripción del Torneo*, el cual tiene dependencia de *Nombre del Torneo*. **Nombre del Torneo → Descripción del Torneo.** Ahora bien, la descripción del torneo no tiene dependencia del otro atributo clave, *Año*. Por tanto, nos encontramos ante un caso de dependencia parcial.

## 4.2. LAS FORMAS NORMALES

Para realizar el proceso de normalización utilizamos las diferentes formas normales. Estas se pueden entender como diferentes etapas consecutivas y acumulativas durante el proceso. Por ejemplo, para normalizar un modelo relacional en segunda forma normal, antes hay que haberlo normalizado en primera forma normal. Para normalizar en tercera forma normal antes hay que haberlo normalizado en primera y segunda forma normal, y así consecutivamente.

De ese modo, se puede concluir que si un modelo relacional está en tercera forma normal es porque también lo está en primera y segunda forma normal.



*En esta imagen se puede observar esquemáticamente cómo las formas normales están englobadas unas en otras.*

## 4.3. PRIMERA FORMA NORMAL (1FN)

Para que un modelo relacional, es decir, un conjunto de tablas esté en primera forma normal (1FN) debe cumplir los siguientes requisitos:

1. El orden de las filas debe ser irrelevante.
2. El orden de las columnas debe ser irrelevante.
3. No debe haber filas repetidas.
4. En cada intersección de fila y columna debe haber solo un valor.
5. Todas las columnas deben ser regulares.

En la práctica, el requisito más crítico es el requisito 4. Es decir, en cada celda tiene que haber un único valor: no se permiten varios valores en el atributo de un registro.

Por ejemplo:

NIF	NOMBRE	APELLIDOS	TELÉFONO
12345678A	José	Pérez	555332211 666221133
12345678B	María	Martínez	555444555 666555777

En este caso, observamos que en la columna *Teléfono* tenemos campos multivaluados, es decir, con varios valores. Para adaptarlo a la primera forma normal debemos eliminar los campos multivaluados.

Para ello, creamos una nueva tabla donde almacenamos los teléfonos y arrastramos el campo clave de la tabla original, en este caso *NIF*.

Por lo que la solución final sería la siguiente:

ALUMNOS		
NIF	NOMBRE	APELLIDOS
12345678A	José	Pérez
12345678B	María	Martínez

ALUMNOS - TELÉFONO	
NIF	TELÉFONO
12345678A	555332211
12345678A	666221133
12345678B	555444555
12345678B	666555777

Como podemos observar, almacenamos la misma información que al principio, pero ahora no tenemos ningún campo multivaluado. Es decir, ninguna intersección de fila y columna con más de un valor.

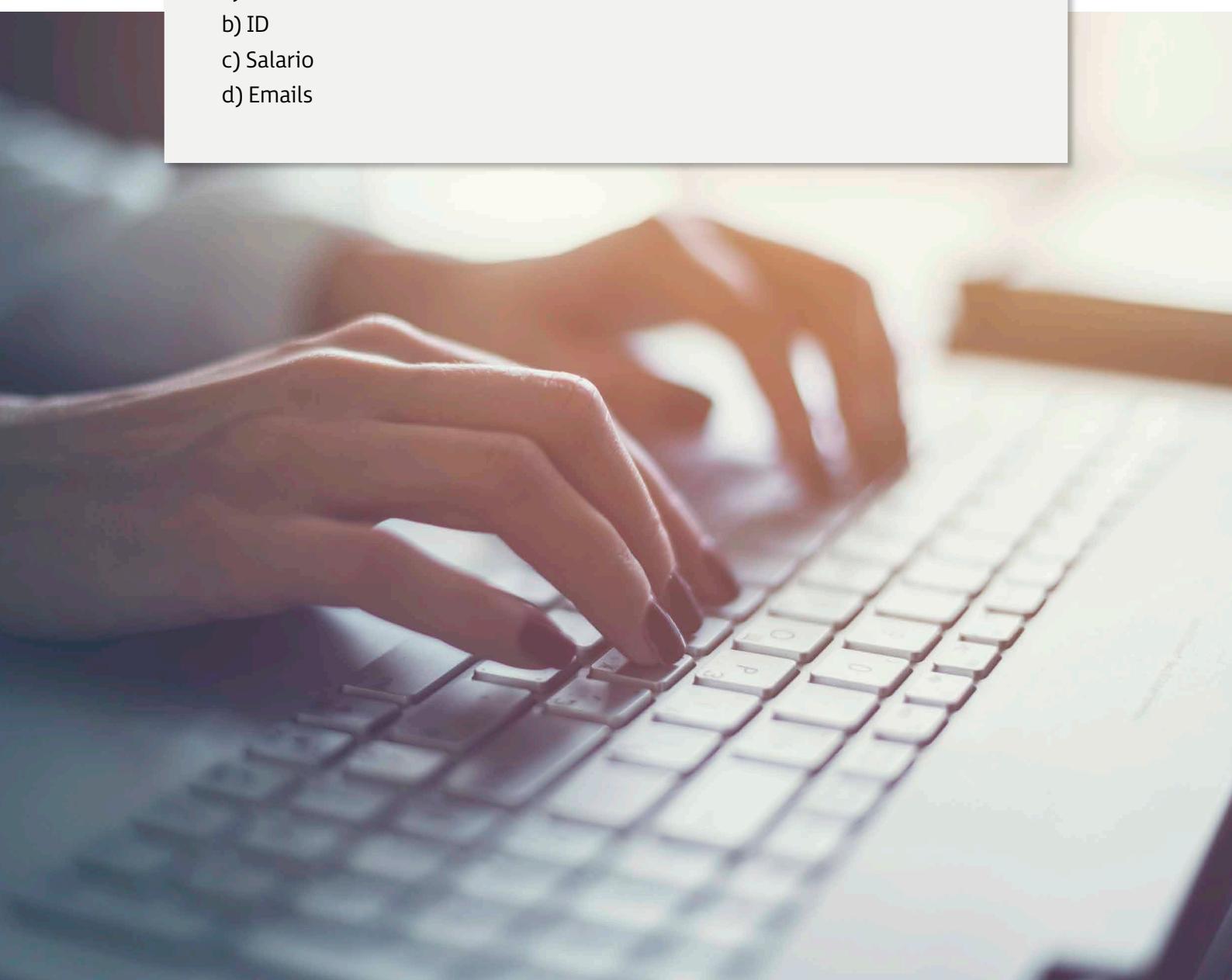


ponte a prueba

Para que una tabla se encuentre en primera forma normal, los atributos deben contener valores atómicos, es decir, que no se puedan dividir. Dada la siguiente tabla, ¿cuál de los siguientes atributos no cumple con esta primera forma normal?

ID	Nombre	Puesto	Salario	Emails
152	Juan Díaz	Jefe de sección	2500	jdiaz@correo.com; jefe@correo.com
153	Verónica Sánchez	Administrativo	2000	vsanchez@correo.com
154	Ana Flores	Administrativo	2000	aflores@correo.com

- a) Puesto
- b) ID
- c) Salario
- d) Emails



## 4.4. SEGUNDA FORMA NORMAL (2FN)

Para que una tabla esté en segunda forma normal (2FN) debe cumplir los siguientes requisitos:

1. Estar en primera forma normal.
2. Todos los atributos no claves deben tener una dependencia funcional completa de los atributos que sí forman la clave (no debe haber dependencia parcial).

Veamos el siguiente ejemplo, donde la clave primaria está compuesta por *CodLibro* y *CodTienda*:

<b>CodLibro</b>	<b>CodTienda</b>	<b>Stock</b>	<b>Dirección</b>
12	22	456	C/ Falsa, 123
22	45	567	C/ Elm, 13
44	22	4	C/ Falsa, 123

Como podemos ver, el campo *Dirección* almacena la dirección de la tienda, es decir, *Dirección* depende de *CodTienda*, pero no de *CodLibro*. Por tanto, esta tabla no se encuentra en 2FN.

Esta situación puede provocar una amenaza a la integridad de los datos. Por ejemplo, podría ocurrir que actualizáramos la dirección de la tienda de código 22 en la fila cuyo código de libro es 12, pero no actualizáramos la misma dirección para el registro donde el código del libro es 44. De tal modo, la información de la tabla presentaría una contradicción.

Para pasar esta tabla a 2FN debemos crear una nueva tabla a la que nos llevaremos el campo no clave que no cumplía la dependencia formal completa, junto con su atributo o atributos con los que sí tiene dependencia completa.

Por tanto la tabla anterior en 2FN se resolvería de la siguiente manera:

<b>STOCK</b>			<b>DIRECCIONES</b>	
<b>CodLibro</b>	<b>CodTienda</b>	<b>Stock</b>	<b>CodTienda</b>	<b>Dirección</b>
12	22	456	22	C/ Falsa, 123
22	45	567	45	C/ Elm, 13
44	22	4		

## 4.5. TERCERA FORMA NORMAL (3FN)

Para que una tabla esté en tercera forma normal (3FN) debe cumplir los siguientes requisitos:

1. Estar en 2FNL.
2. No existir dependencia funcional transitiva en los atributos no clave.

Pongamos el siguiente ejemplo, donde la clave es *Cod. Emp* (código de empleado):

Cod. Emp	Nombre	Apellidos	Fecha Nac	Cod. Dpto	Nombre Dpto
001	Pepe	Pérez	22/12/1958	12	Cuentas
002	María	Pascuález	12/11/1987	11	RRHH

Podemos ver que el campo *Nombre Dpto* depende transitivamente de la clave a través del campo *Cod. Dpto*, es decir, el nombre del departamento depende del código del departamento, el cual depende de la clave de la tabla, código del empleado, por tanto, esta tabla no está en 3FN.

Para llegar a una solución óptima podemos pasar a crear dos tablas, eliminando la información que tenía dependencia transitiva y creando una nueva tabla para ella.

EMPLEADOS					EMPLEADOS	
Cod. Emp	Nombre	Apellidos	Fecha Nac	Cod. Dpto	Cod. Dpto	Nombre Dpto
001	Pepe	Pérez	22/12/1958	12	12	Cuentas
002	María	Pascuález	12/11/1987	11	11	RRHH

De este modo eliminamos la transitividad en la tabla y obtenemos la normalización en 3FN. Por lo general, normalizar las tablas de un modelo relacional hasta la 3FN es una normalización buena y suficiente. No obstante, existen otras normalizaciones más avanzadas que describiremos a continuación.



ponte a prueba

Para que una tabla esté en tercera forma normal (3FN), es necesario que esté en 2FN y no disponga de dependencias funcionales en los atributos no clave, pero no es necesario que esté en 1FN.

- a) Verdadero.
- b) Falso.

## 4.6. FORMA NORMAL BOYCE-CODD (FNBC)

La **forma normal Boyce-Codd (FNBC)** es una versión de la 3FN un poco más estricta. Decimos que una relación está en FNBC cuando está en 3FN y todos sus atributos no clave son clave candidata.

The diagram illustrates the decomposition of a 3NF table into two 4NF tables. The original table has columns DNI ALUMNO, ASIGNATURA, and TUTOR. It decomposes into two tables: one with columns ASIGNATURA and TUTOR, and another with columns DNI ALUMNO and ASIGNATURA.

DNI ALUMNO	ASIGNATURA	TUTOR
12345678A	Programación	Paco Ibañez
12345678B	Base de datos	Manolo García

ASIGNATURA	TUTOR
Programación	Paco Ibañez
Base de datos	Manolo García

DNI ALUMNO	ASIGNATURA
12345678A	Programación
12345678B	Base de datos

## 4.7. OTRAS FORMAS NORMALES (4FN, 5FN)

Estas formas normales se van a ocupar de las dependencias entre atributos multivaluados:

- En la cuarta forma normal (4FN) se requiere que una tabla esté en 3FN y no haya dependencias multivaluadas no triviales.
- En la quinta forma normal (5FN) se requiere que una tabla esté en 4FN y cada dependencia de unión se implique por las claves candidatas.

## 4.8. DESNORMALIZACIÓN

Podemos definir la desnormalización como aquel proceso mediante el cual pretendemos optimizar el desarrollo de una BD mediante la agregación de aquellos datos redundantes.



# SQL

5

LENGUAJES DE LAS BBDD. SQL

El lenguaje de programación **SQL es el lenguaje fundamental de los SGBD** relacionales. Una de las características principales de este lenguaje es su aspecto declarativo en vez de imperativo, es decir, el programador debe indicar qué hacer y no cómo hacerlo.

Este lenguaje pretende ser lo más natural posible y, por esta razón, es considerado un lenguaje de cuarta generación. Los elementos que componen el lenguaje SQL son los siguientes:

- **DML (Data Manipulation Language)**: es el lenguaje que manipula los datos ya creados. Modifica los registros o tuplas de la BD.

#### DML

[youtu.be/ONtPL5W\\_TNU](https://youtu.be/ONtPL5W_TNU)



- **DDL (Data Definition Language)**: permite la creación y diseño de la estructura de la BD y las tablas que la componen.

#### DDL

[youtu.be/EfpDY9yN5ik](https://youtu.be/EfpDY9yN5ik)



- **DCL (Data Control Language)**: administra a los usuarios de las BBDD, concediendo o denegando los permisos oportunos.
- **TCL (Transaction Control Language)**: lenguaje que controla el procesamiento de las transacciones de las BBDD.

A continuación, para terminar este apartado de introducción al SQL, vamos a nombrar las **normas básicas** que tener en cuenta a la hora diseñar instrucciones en SQL:

- No se distingue entre mayúsculas y minúsculas.
- La instrucción en SQL debe terminar con el carácter ; (punto y coma). Esto se debe a que el compilador está diseñado para que vaya decodificando la instrucción hasta que se encuentre con este carácter que delimita el fin del comando.
- Antes de finalizar la instrucción, cualquier comando puede ir seguido por un espacio en blanco o un salto de línea.
- Se puede tabular la instrucción para clarificar la orden deseada.



### ponte a prueba

**¿Cuál de las siguientes opciones no es una norma básica para tener en cuenta cuando diseñamos instrucciones SQL?**

- a) Es posible tabular las instrucciones.
- b) No distingue entre mayúsculas y minúsculas.
- c) No es necesario delimitar el fin de un comando.
- d) Cualquier comando puede ir seguido de un espacio antes de finalizar la instrucción.

## 5.1. TIPOS DE LENGUAJES PARA GESTIONAR LOS DATOS EN UN SGBDR CORPORATIVO

En este tema vamos a explicar algunos de los tipos de lenguajes existentes que se pueden usar para la gestión y desarrollo de una BD en un SGBDR corporativo, por tanto, en esta categoría no incluiremos los lenguajes para SGBD tipo *freeware*, como podría ser SQLite.

### Para Oracle

PL/SQL es un lenguaje procedural usado en Oracle RDBMS (Relational Data Base Management System). PL/SQL es el acrónimo de Procedural Language / Structured Query Language. Al igual que la mayoría de lenguajes de BD, PL/SQL está basado en SQL y además soporta algunas características extra sobre el control de excepciones, el manejo de las variables o en las estructuras de control de flujo.

Los lenguajes procedurales son propios de la programación estructurada y usan llamadas a rutinas, las cuales tienen un conjunto de sentencias. PL/SQL es un lenguaje procedural orientado a SQL.

### Para Microsoft

SQL server es un SGBDR de la empresa Microsoft. El lenguaje usado en SQL server es **T-SQL**, acrónimo de Transact-SQL.

Dicho lenguaje deriva del estándar SQL, pero al igual que PL/SQL añade la opción de trabajar con procedimientos. Es un lenguaje muy potente que nos da muchas posibilidades para manejar una BD.

## 5.2. HERRAMIENTAS GRÁFICAS PROPORCIONADAS POR EL SGBD PARA LA EDICIÓN DE LA BD

Existen múltiples herramientas a la hora de crear y manipular una determinada BD si disponemos de alguna interfaz gráfica que ayude al Administrador de la Base de Datos (DBA). Esta también debe ser capaz de enviar diferentes comandos de administración de forma automática, sin necesidad de profundizar con más detalle sobre su sintaxis.

Aquí tenemos algunas herramientas gráficas para la gestión de un SGBD:

<p><b>MySQL Workbench</b></p> <p>Es una herramienta gráfica para la creación, desarrollo y mantenimiento de BBDD para el SGBD de MySQL server. Posee una versión freeware mantenida por la comunidad y otra versión privada, perteneciente a la empresa Oracle.</p>	
<p><b>MariaDB</b></p> <p>Es otra herramienta gráfica para la gestión de una BD de MySQL, pero está desarrollada por la comunidad y su desarrollo está liderado por algunos de los primeros desarrolladores de MySQL. Tiene una licencia de software libre.</p>	
<p><b>PhpMyAdmin de MySQL</b></p> <p>Es una herramienta que nos ayuda a manejar un SGBD como MySQL a través de la web. Esta herramienta tiene licencia de software libre. Tiene una interfaz muy intuitiva y soporta todas las características de MySQL. Dispone de diferentes opciones que nos permiten visualizar cualquier gestión que realicemos en la BD como crear, modificar o borrar tablas. Y también permite importar y exportar información, estadísticas, copias de seguridad, etc.</p>	

### Oracle Enterprise Manager y Grid Control

Dispone de dos herramientas gráficas con su correspondiente interfaz web. Estas están montadas en un servidor web de Oracle:

- **Enterprise Manager**: permite manipular cualquier función básica correspondiente a una BD. Esta herramienta ya viene incorporada en el software de Oracle.
- **Grid Control**: gestiona diferentes BBDD en distintos servidores y ofrece la posibilidad de poder consultar el estado y rendimiento de cualquiera de ellas. Se debe instalar aparte del software de Oracle.

### IBM Data Studio

Ofrece un entorno modular e integrado para manipular y modificar los objetos de los permisos de una BD, entre otras muchas acciones. Facilita la construcción de consultas SQL. Permite crear servicios web para poder distribuir datos de consulta.

## 5.2.1. DESCARGA E INSTALACIÓN DE MySQL WORKBENCH Y MARIA DB

Durante la edición de este libro vamos a usar como referencia el SGBD **MySQL**. Debemos tener en cuenta la diferencia entre el servidor y el cliente. MySQL dispone de una instalación completa que incluye tanto el servidor como el cliente.

Podemos trabajar por terminal de comandos o usar la herramienta gráfica **MySQL Workbench** para gestionar desde el cliente. Si el usuario deseara diferenciar de manera mucho más clara la instalación del servidor respecto a la instalación del cliente, podría usar HeidiSQL como software cliente. Por ello, mostraremos a continuación las dos opciones de instalación de MySQL server, con Workbench o HeidiSQL como cliente.

Sigue este pequeño tutorial sobre cómo descargar e instalar el SGBD de MySQL en sus dos posibilidades:

1. Usar MySQL como servidor y MySQL Workbench como cliente.
2. Usar MySQL como servidor y HeidiSQL como cliente.

Para descargar MySQL hay que seguir estos pasos:

1. Primero, debemos acceder al siguiente enlace para comenzar el proceso de descarga de MySQL:



Nos aparecerá una página web como la que se muestra abajo. Debemos hacer clic en *Go to Download Page* para continuar:

**MySQL Community Server 8.0.22**

Select Operating System: Microsoft Windows

Recommended Download:

**MySQL Installer** for Windows

All MySQL Products. For All Windows Platforms. In One Package.

Starting with MySQL 5.6 the MySQL installer package replaces the standalone MSI packages.

**Windows (x86, 32 & 64-bit), MySQL Installer MSI** Go to Download Page >

Other Downloads:

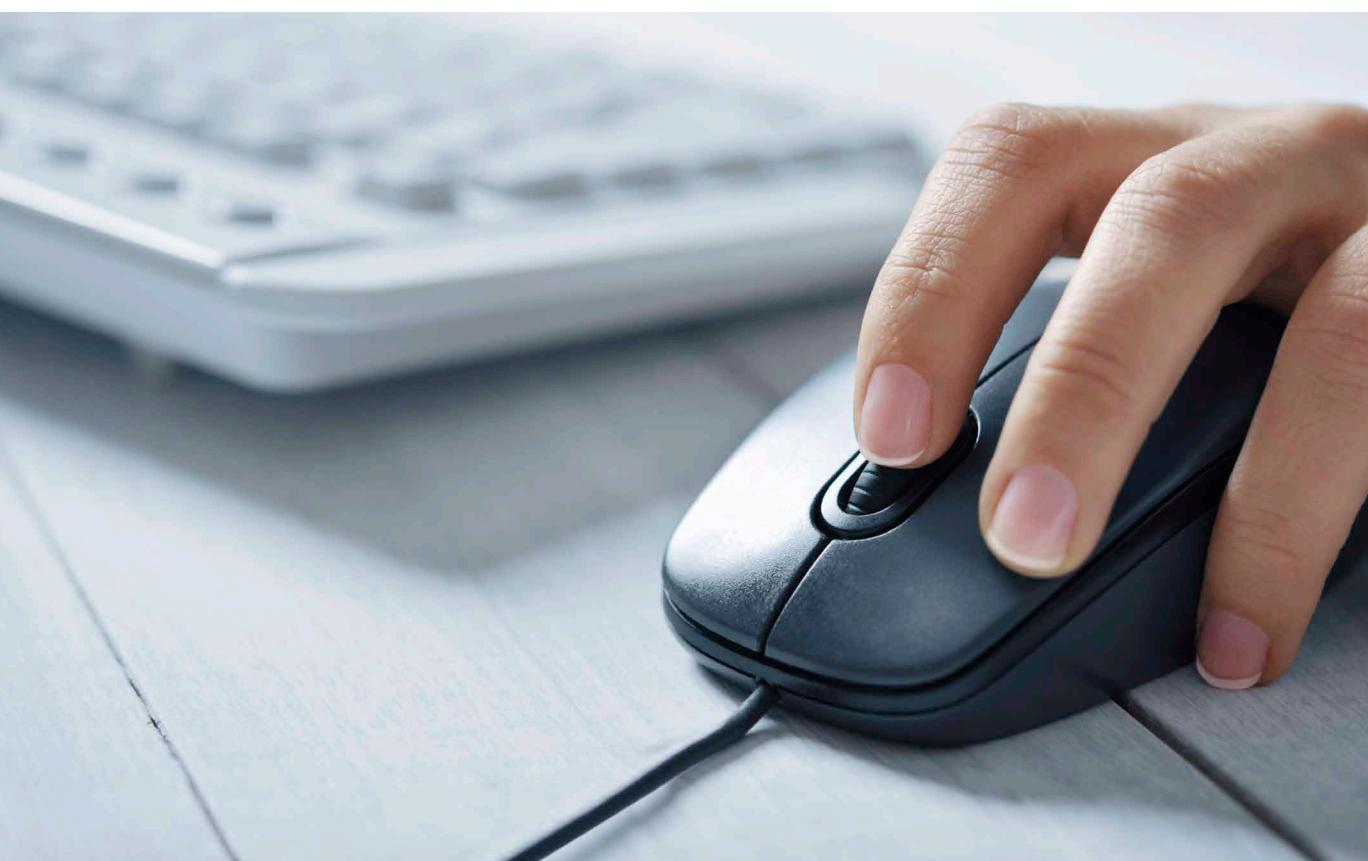
Windows (x86, 64-bit), ZIP Archive (mysql-8.0.22-winx64.zip)	8.0.22	191.4M	<b>Download</b>
Windows (x86, 64-bit), ZIP Archive Debug Binaries & Test Suite (mysql-8.0.22-winx64-debug-test.zip)	8.0.22	434.4M	<b>Download</b>

We suggest that you use the MD5 checksums and GnuPG signatures to verify the integrity of the packages you download.

2. Tendremos que elegir sistema operativo y las opciones de descarga. En nuestro caso elegimos *Microsoft Windows* y hacemos clic en la segunda opción de *Download* para descargar al completo el archivo en la máquina local.

Other Downloads:

Windows (x86, 64-bit), ZIP Archive (mysql-8.0.22-winx64.zip)	8.0.22	191.4M	<b>Download</b>
Windows (x86, 64-bit), ZIP Archive Debug Binaries & Test Suite (mysql-8.0.22-winx64-debug-test.zip)	8.0.22	434.4M	<b>Download</b>



3. A continuación, nos pedirá si deseamos registrarnos para comenzar la descarga. En nuestro caso, elegiremos la opción de *No thanks, just start my download* para descargarnos el fichero.

## ④ MySQL Community Downloads

[Login Now](#) or [Sign Up](#) for a free account.

An Oracle Web Account provides you with the following advantages:

- Fast access to MySQL software downloads
- Download technical White Papers and Presentations
- Post messages in the MySQL Discussion Forums
- Report and track bugs in the MySQL bug system

[Login »](#)

using my Oracle Web account

[Sign Up »](#)

for an Oracle Web account

MySQL.com is using Oracle SSO for authentication. If you already have an Oracle Web account, click the Login link. Otherwise, you can signup for a free account by clicking the Sign Up link and following the instructions.

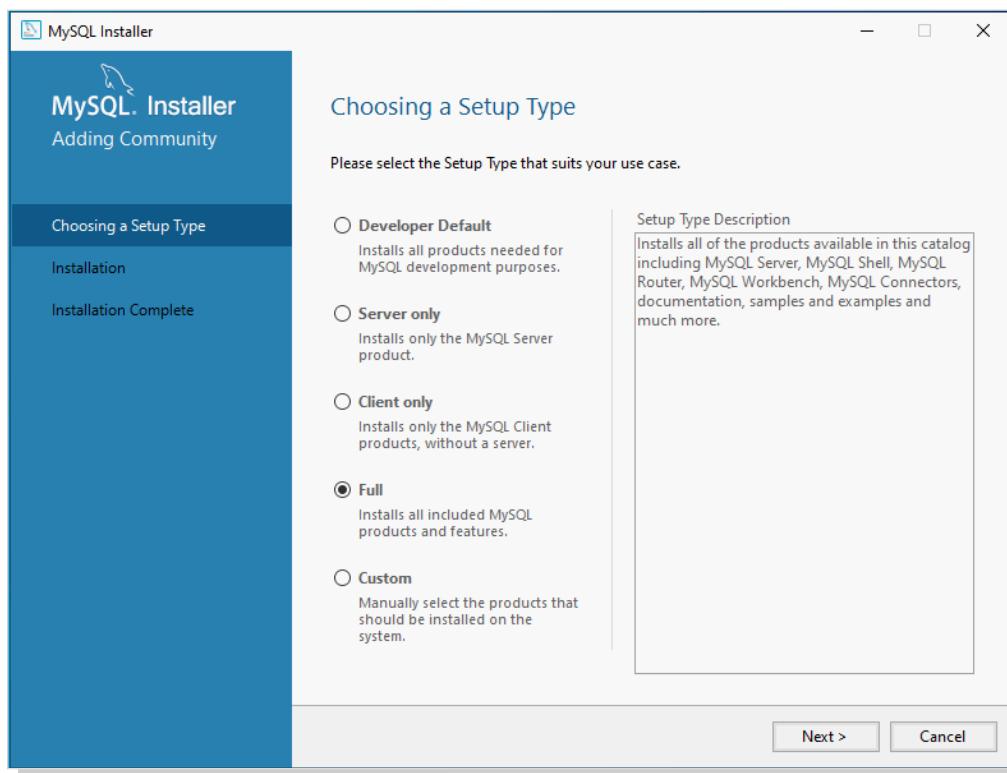
[No thanks, just start my download.](#)

Una vez realizados estos pasos, **el archivo que contiene MySQL Server se descargará en nuestra máquina local**. El próximo paso será instalarlo.

4. Una vez descargado el archivo, lo ejecutamos y nos aparecerán varias opciones. Dependiendo de qué tipo de instalación queramos hacer, tendremos que elegir una opción u otra:

- En caso de que deseemos trabajar con MySQL Workbench, seleccionaremos la opción *Full*.
- En caso de que deseemos trabajar con HeidiSQL, seleccionaremos la opción *Server only*.

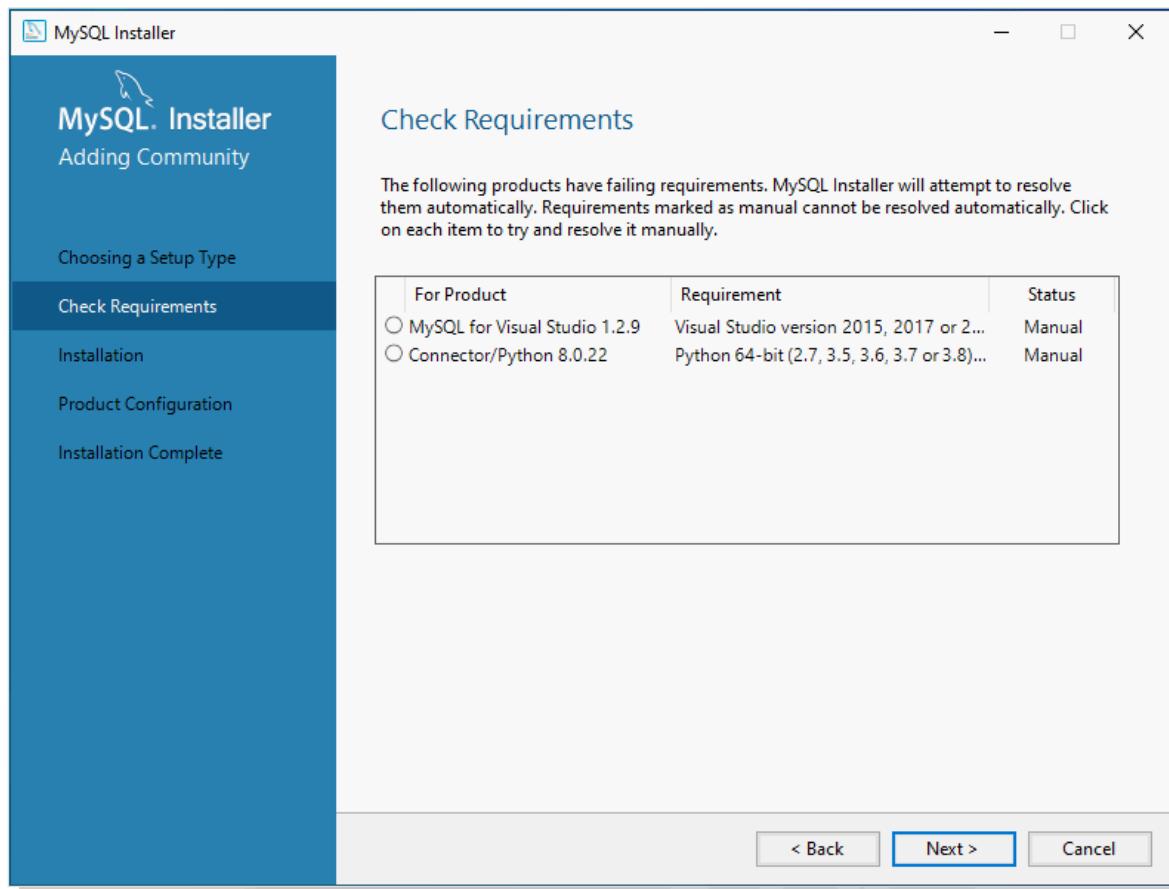
En este caso, vamos a optar por la opción *Full* para instalar las todas las herramientas disponibles en MySQL y, después, haremos clic en *Next*.



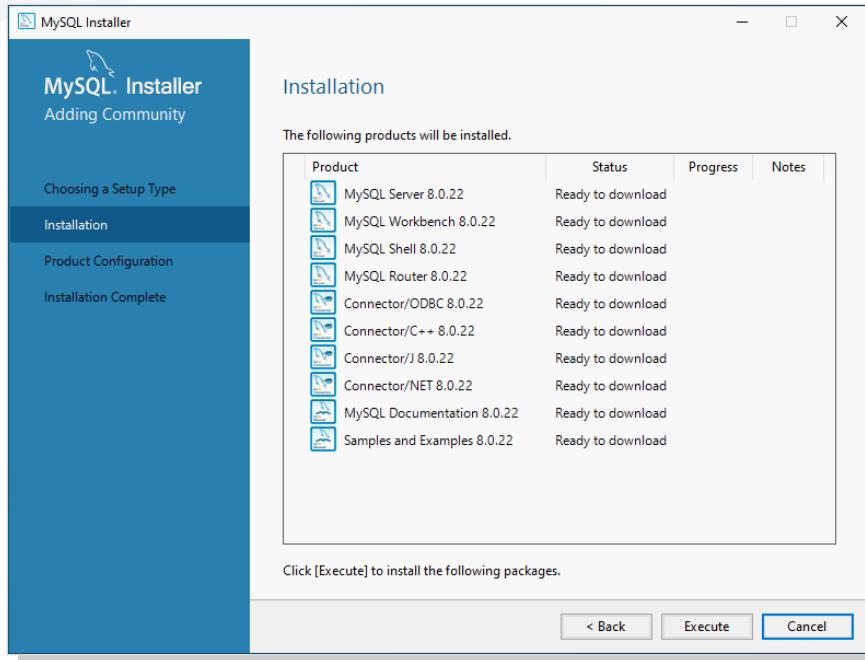
5. En este punto, la instalación nos informará de todos los programas que hay que tener instalados en la máquina para que MySQL Server funcione correctamente. Es posible que algunos programas ya los tuviéramos previamente instalados, por tanto, MySQL instalará algunos programas de manera automática, pero es posible que haya otros programas requeridos que tengamos que instalar a mano.

Es decir, aquellos programas requeridos que MySQL no instale automáticamente, tendremos que descargarlos e instalarlos por nuestra cuenta.

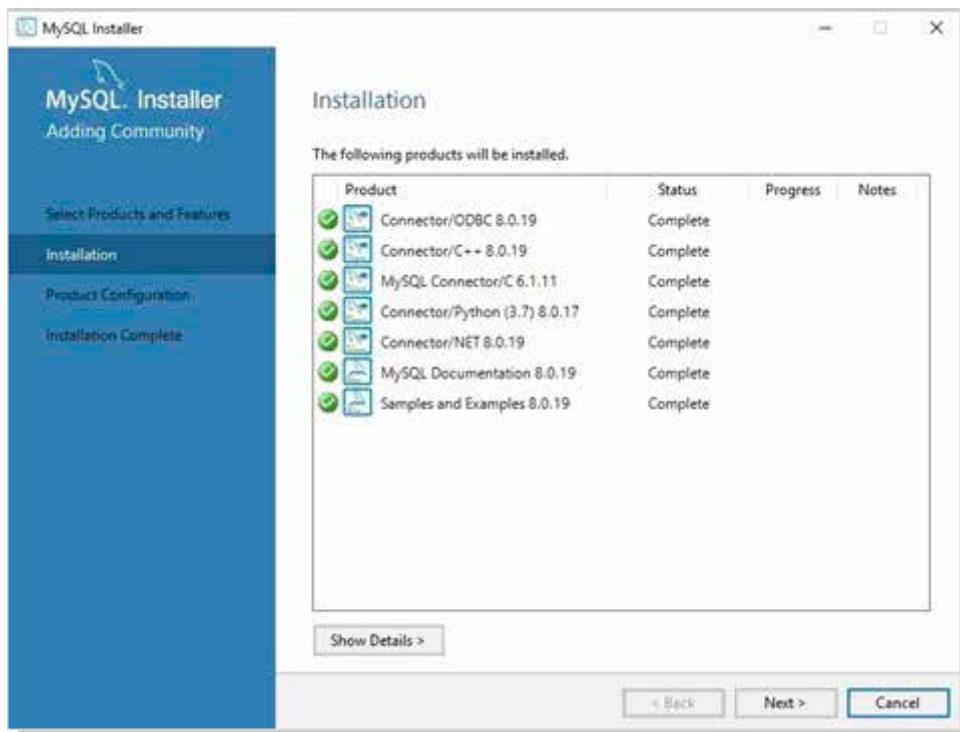
Como podemos observar en la imagen, algunos programas tienen un *status* de *Manual*, lo que significa que debemos instalarlos por nuestra cuenta. Hacemos clic en *Next*.



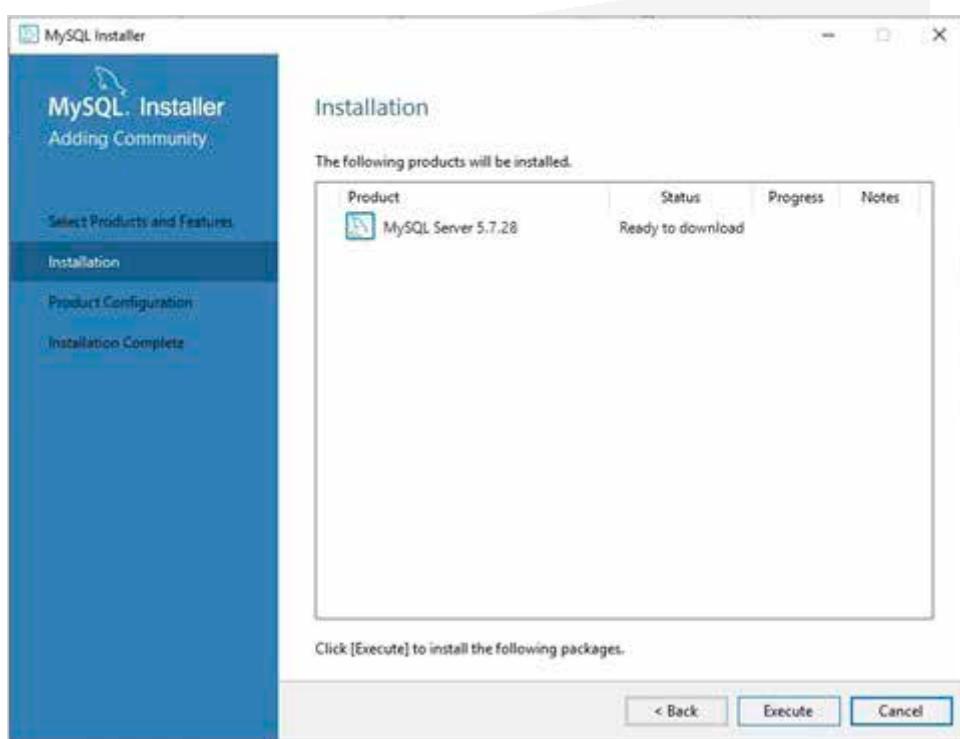
En esta imagen podemos observar cómo *MySQL Installer* automáticamente va instalando aquellos programas necesarios para el SGBD:



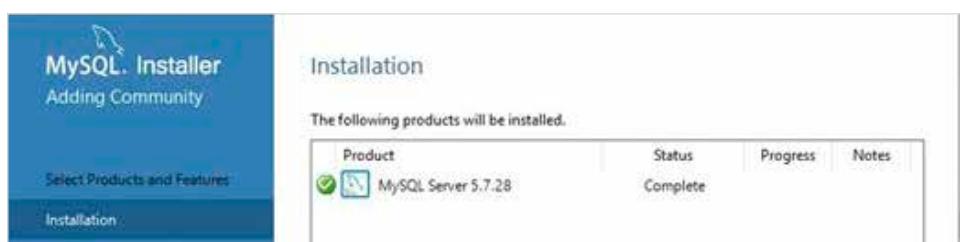
Al final del proceso, nos aparecerá un listado de los programas instalados. Después debemos clic en Next.



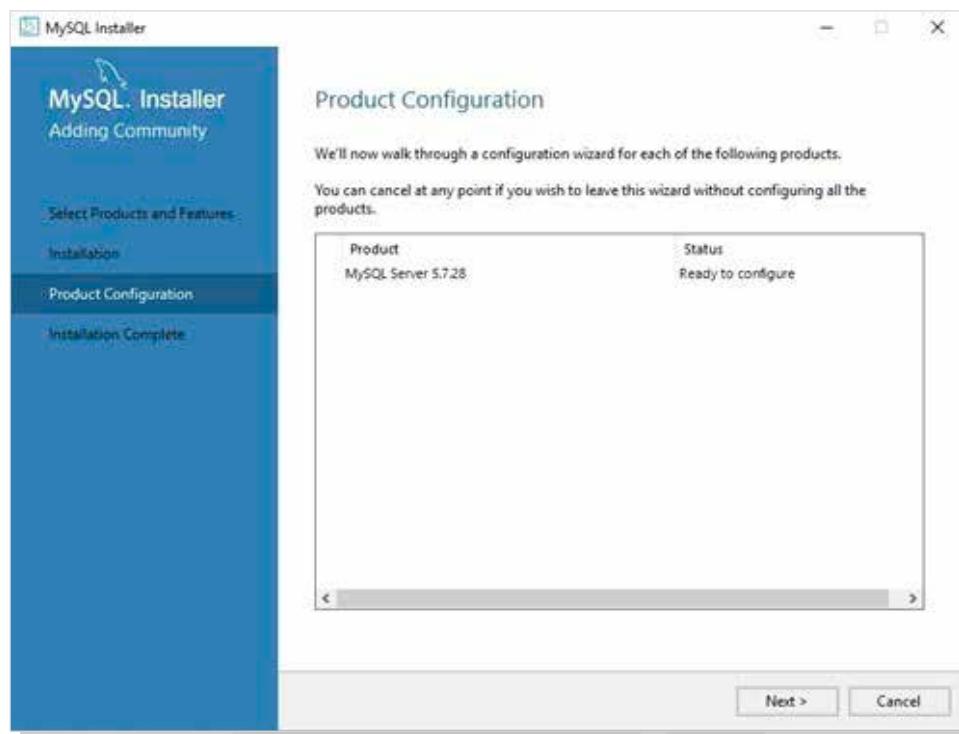
6. En el siguiente paso instalamos el propio MySQL Server. Debemos hacer clic en la opción *Execute*.



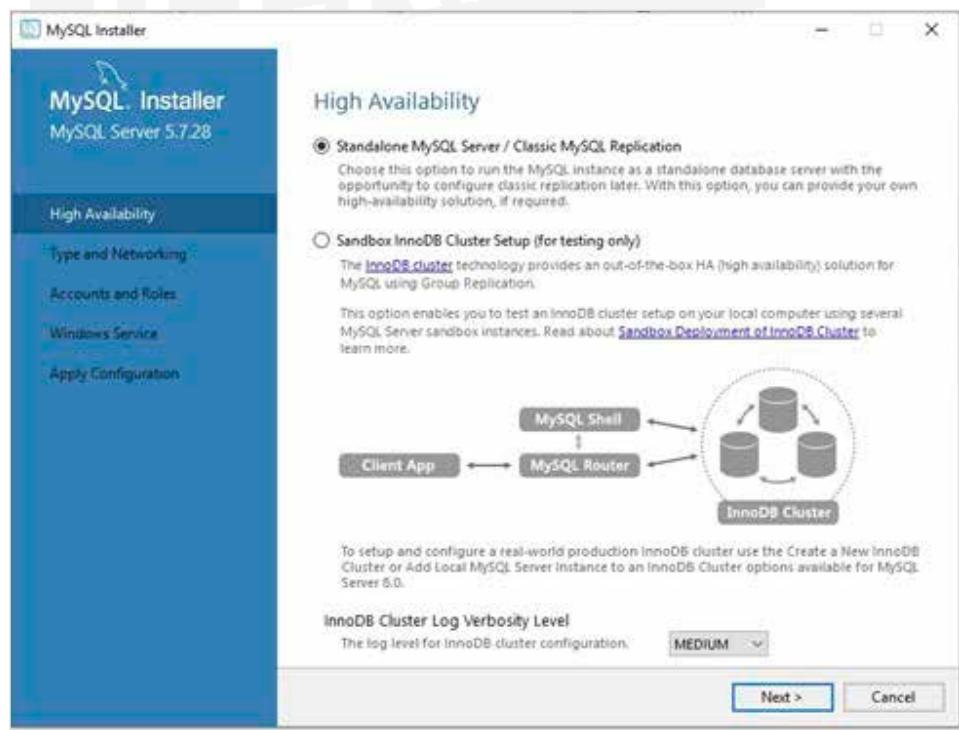
Una vez instalado, debemos hacer *clic* en *Next*:



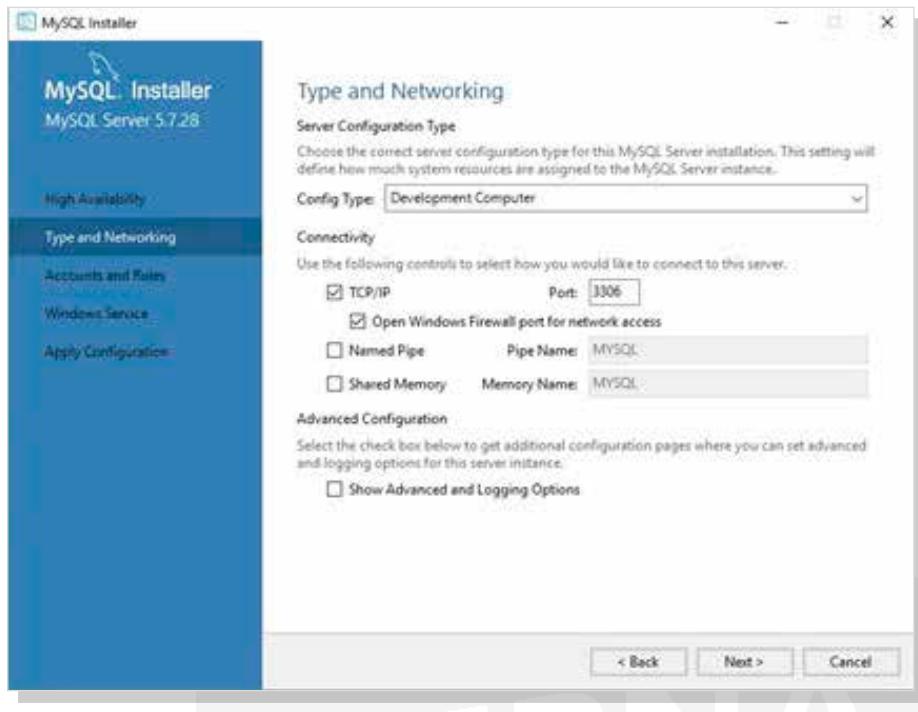
Ya tenemos MySQL Server instalado en nuestro sistema, pero ahora debemos configurarlo. Para ello, haremos clic en Next.



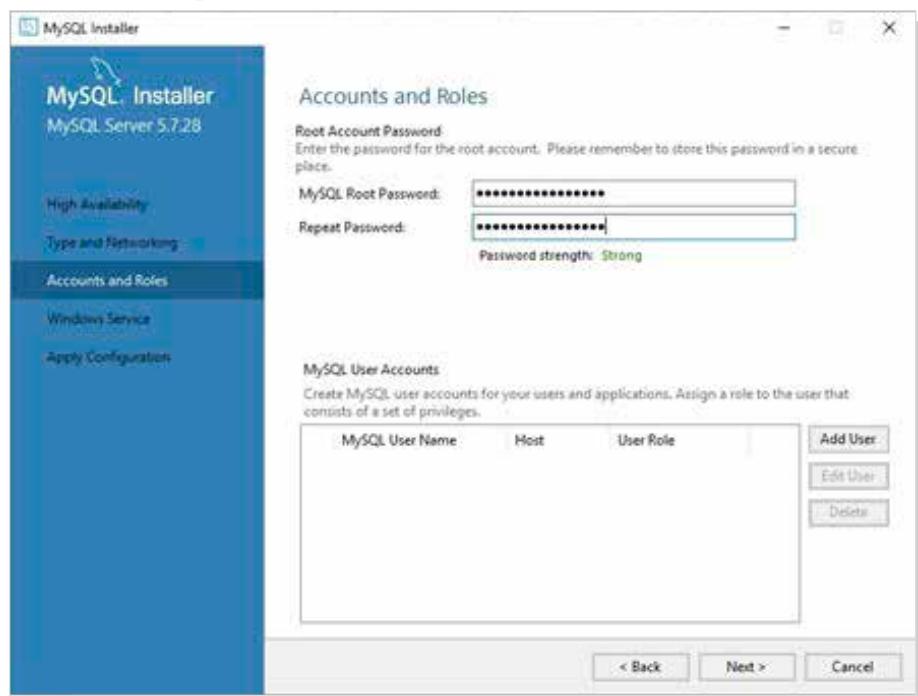
7. Para conseguir nuestro propósito, elegiremos la opción *Standalone MySQL Server* y haremos clic en *Next*.



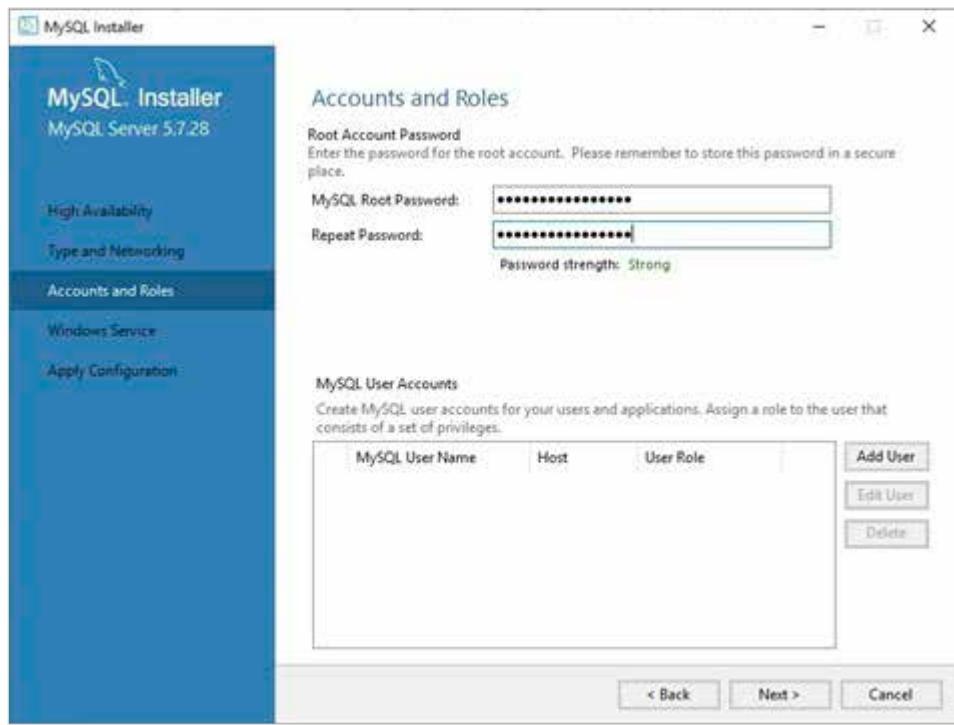
8. En el siguiente paso, el sistema nos pregunta en qué número de puerto queremos que trabaje el servidor. El puerto adecuado puede depender de la máquina en la que se instale MySQL Server. En nuestro caso, dejamos el puerto que viene por defecto y clicamos en Next:



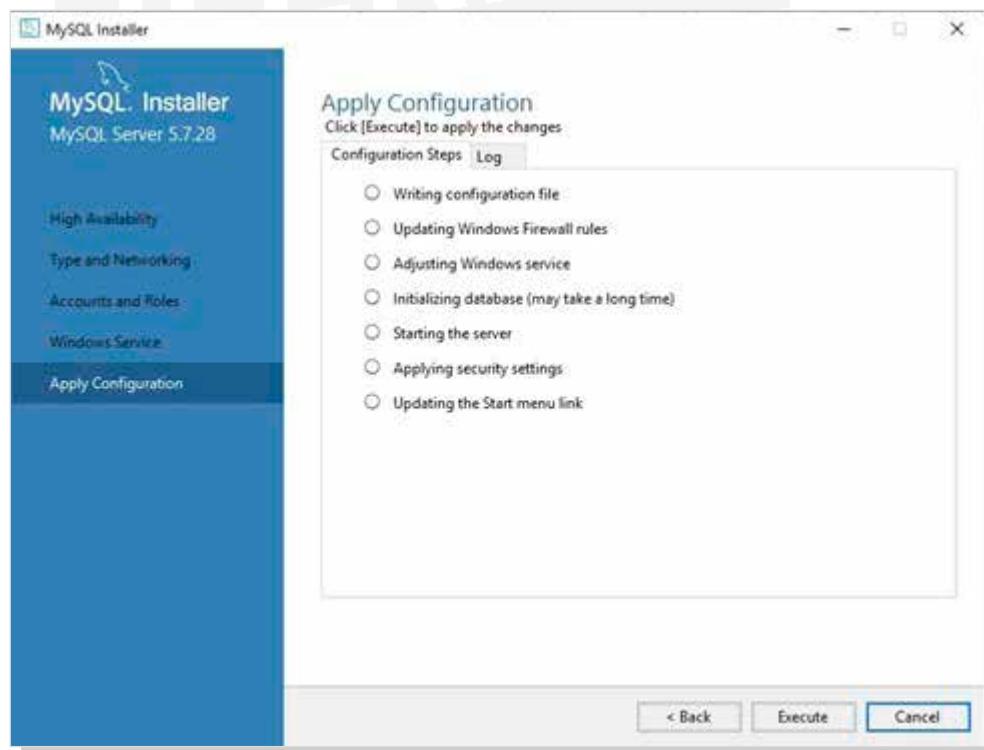
9. A continuación, indicamos la contraseña que asignamos a MySQL Server. Para este ámbito académico, se recomienda una contraseña fácil de recordar. Una vez creada la contraseña, hacemos clic en Next.



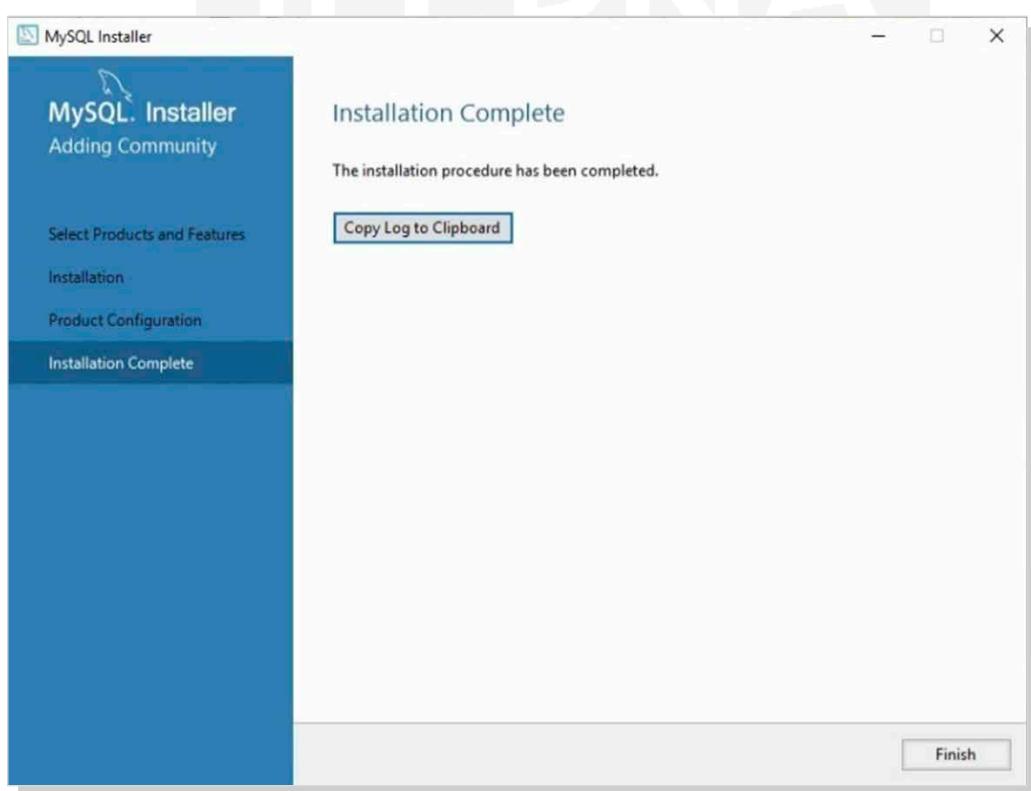
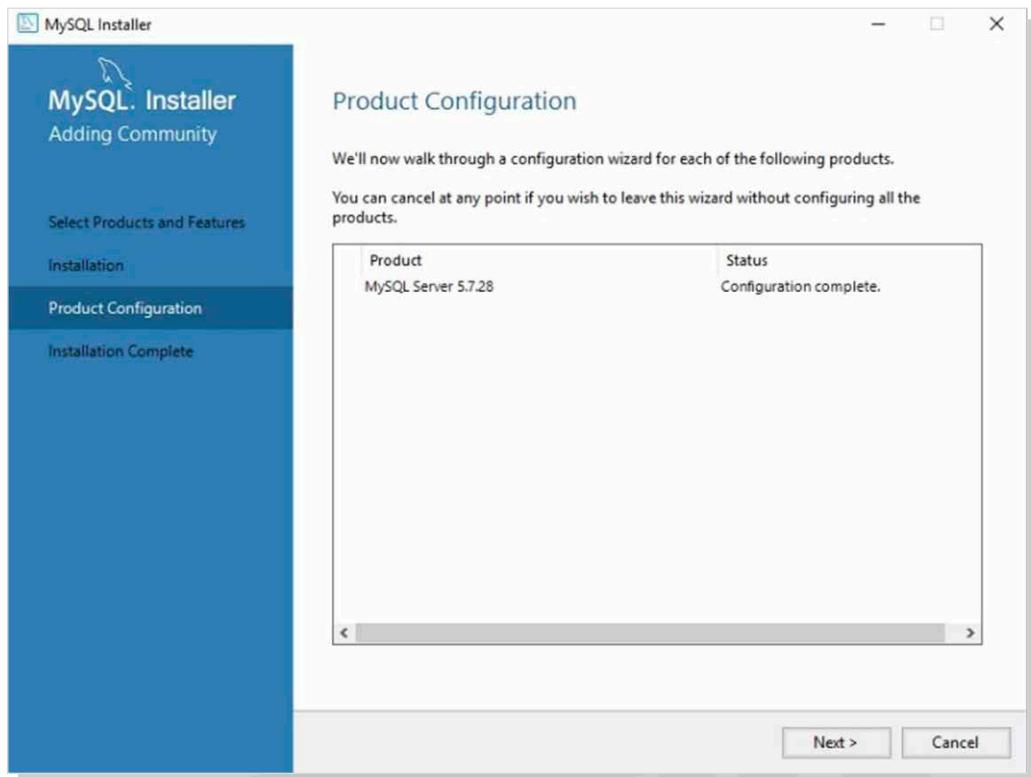
10. Veamos ahora el servicio que MySQL Server habilita en Windows. En nuestro caso, lo dejamos como está por defecto y clicamos en Next.

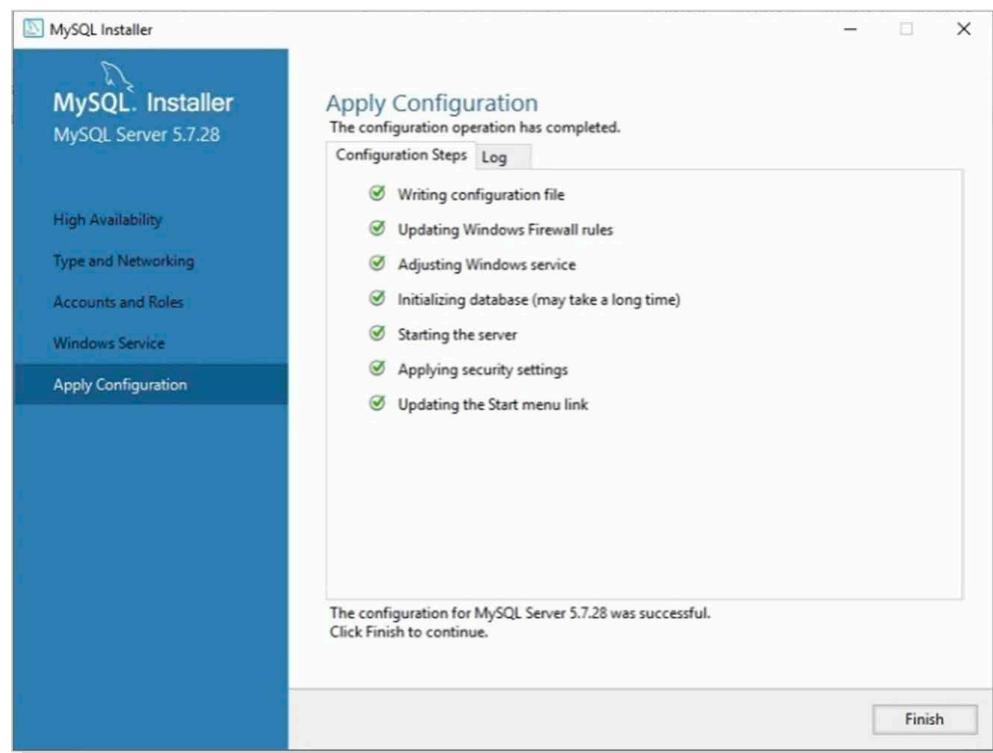


11. A continuación, nos dirigimos a Execute para que se apliquen ciertas configuraciones.



12. Y, finalmente, hacemos clic en *Finish*, en *Next* y ya habremos instalado MySQL Server en la máquina.

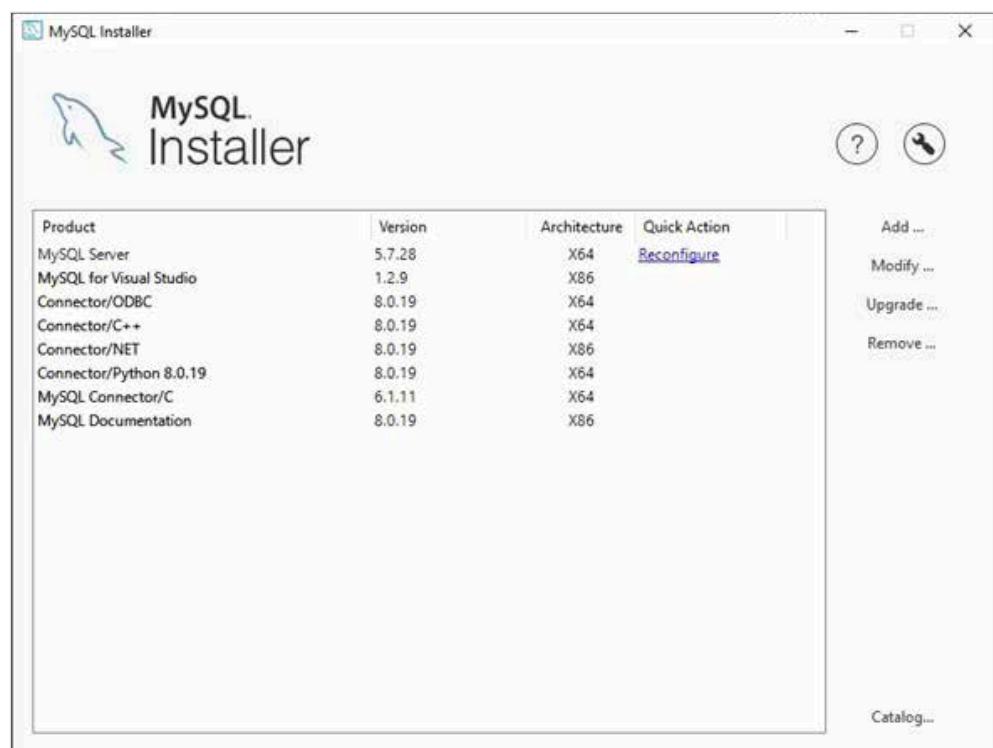




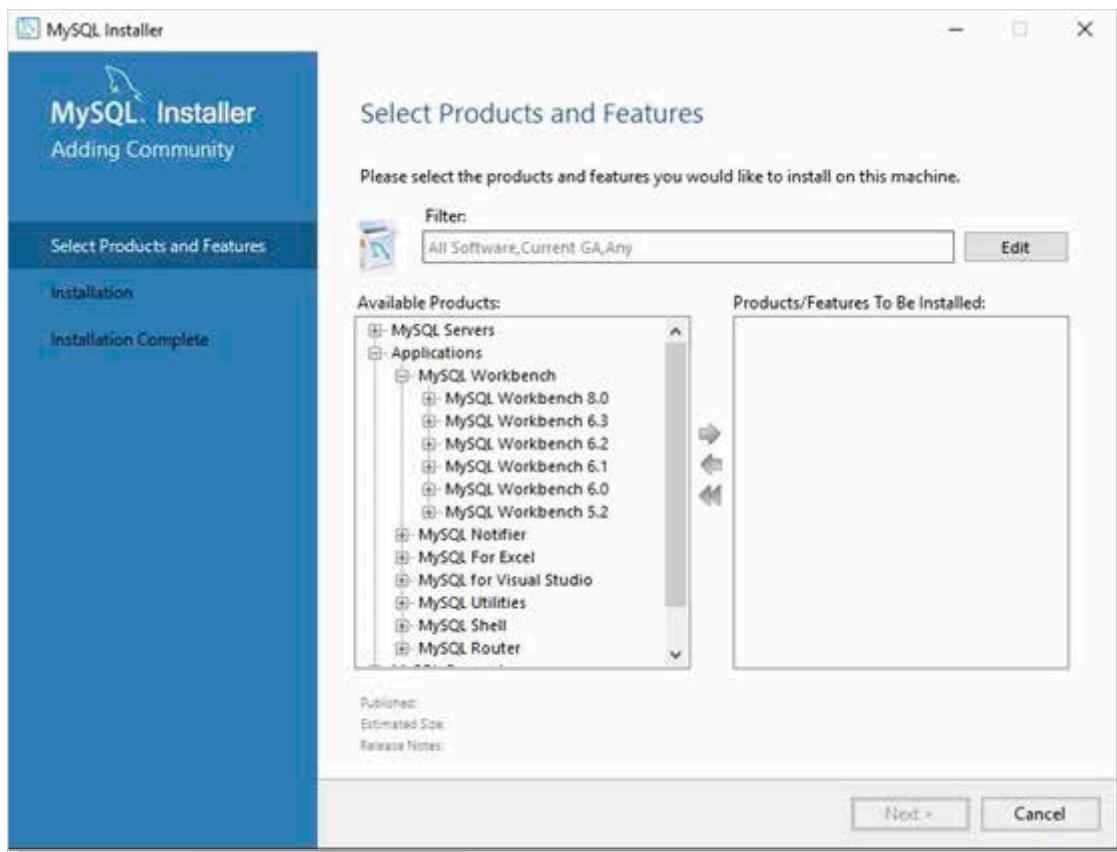
**Una vez instalado el servidor, debemos instalar el cliente.** Este tendrá la misión de funcionar como interfaz gráfica y de actuar como puente entre el servidor y el administrador de la BD.

En caso de que hayamos elegido instalar **MySQL Workbench**, procederemos de la siguiente manera:

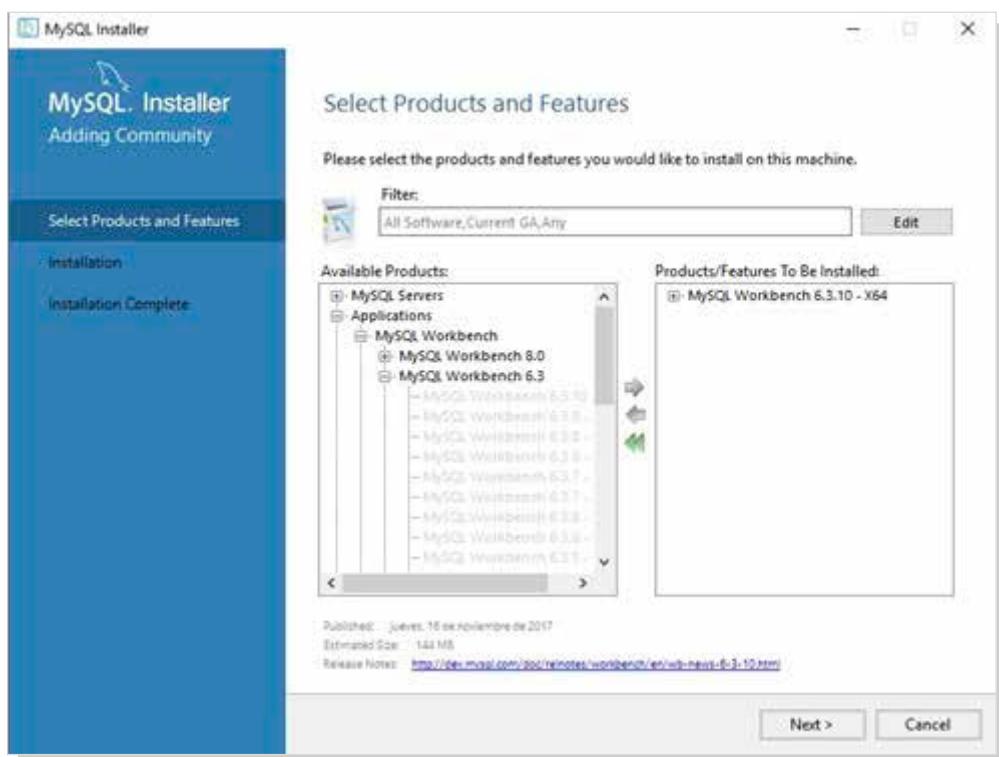
1. Abrimos MySQL Installer y pulsamos en *Add*:



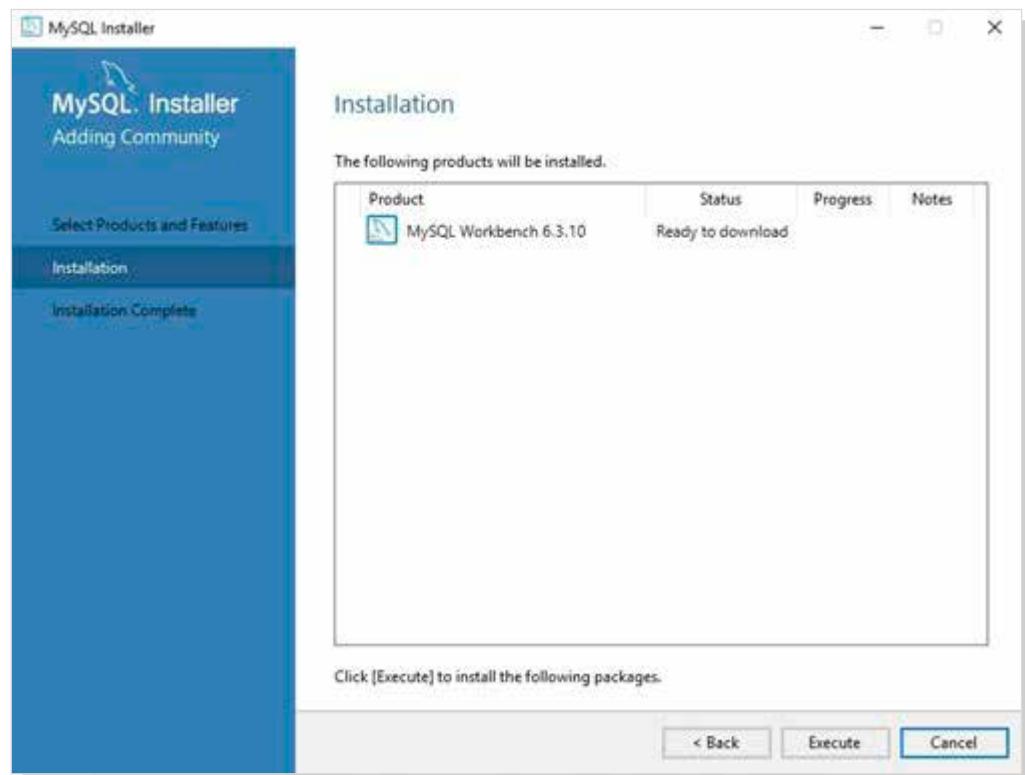
2. Observamos una cantidad de elementos que nos ofrece MySQL que podemos incorporar: ejemplos, documentación, consola (*shell*). En nuestro caso, nos interesa incorporar MySQL Workbench. Así que elegimos una versión de Workbench y la volcamos a la parte de la derecha para que sea incluida.



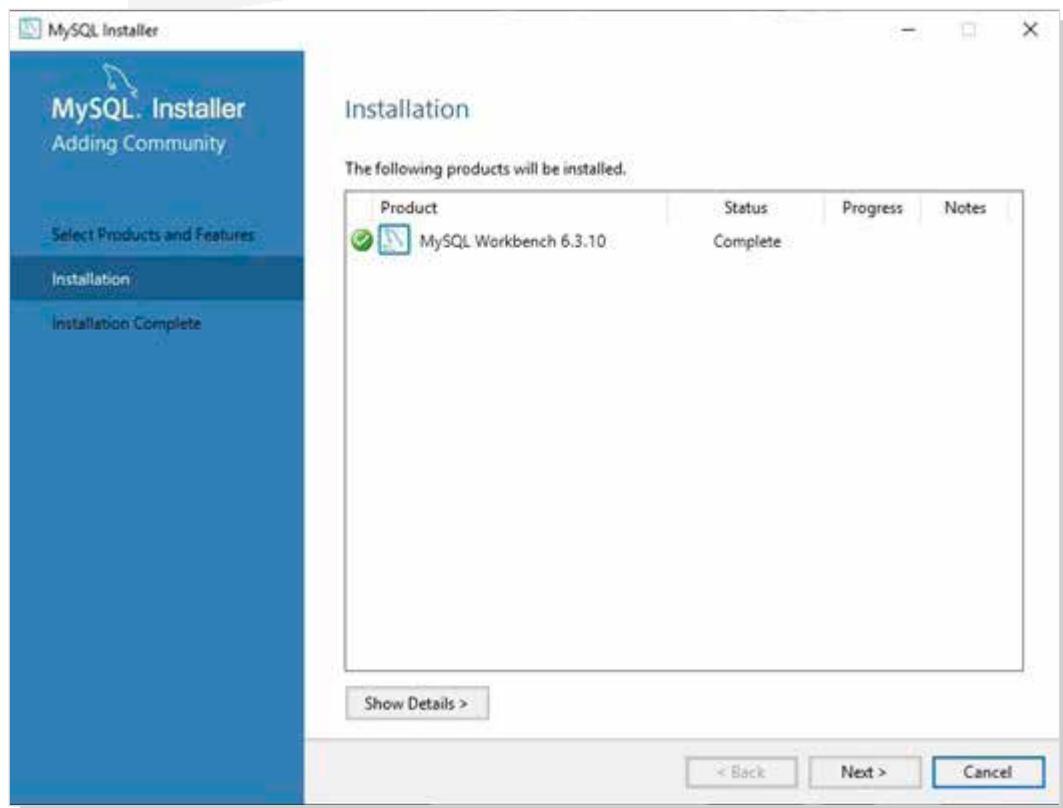
Una vez lo hayamos añadido, pulsamos *Next*:

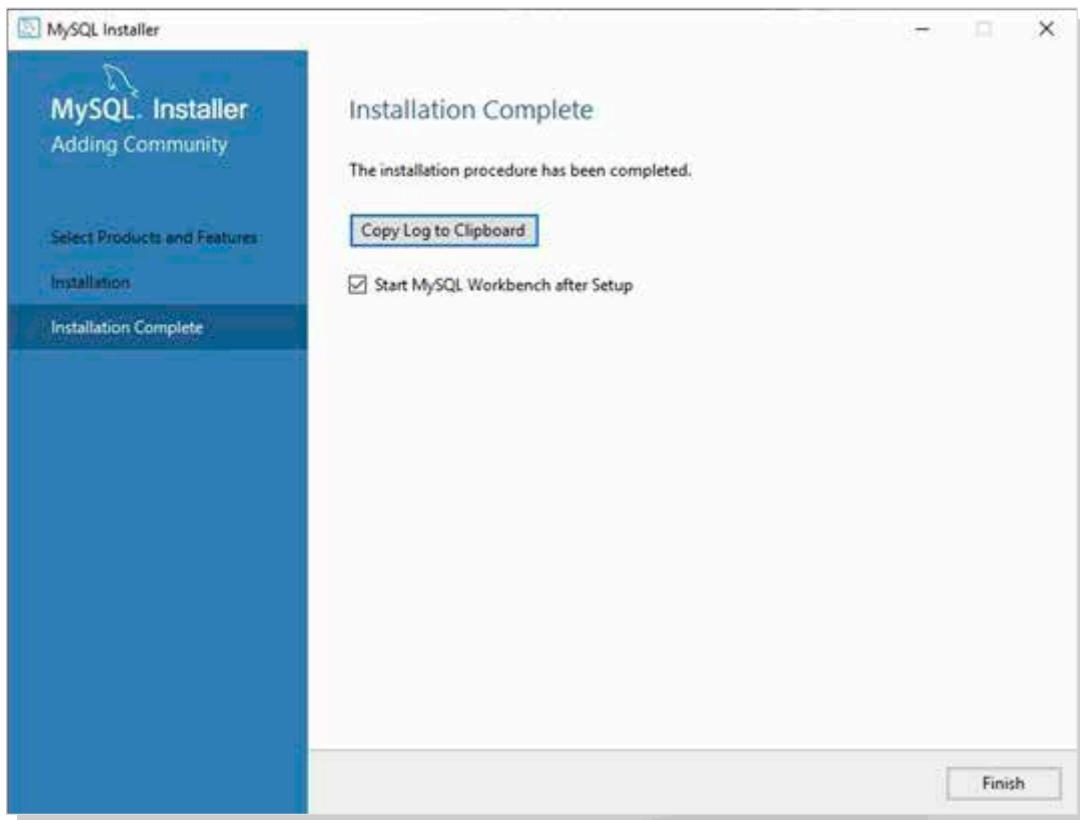


3. Llegados a este punto, hacemos clic en *Execute* para que el programa instalador descargue e instale MySQL Workbench.

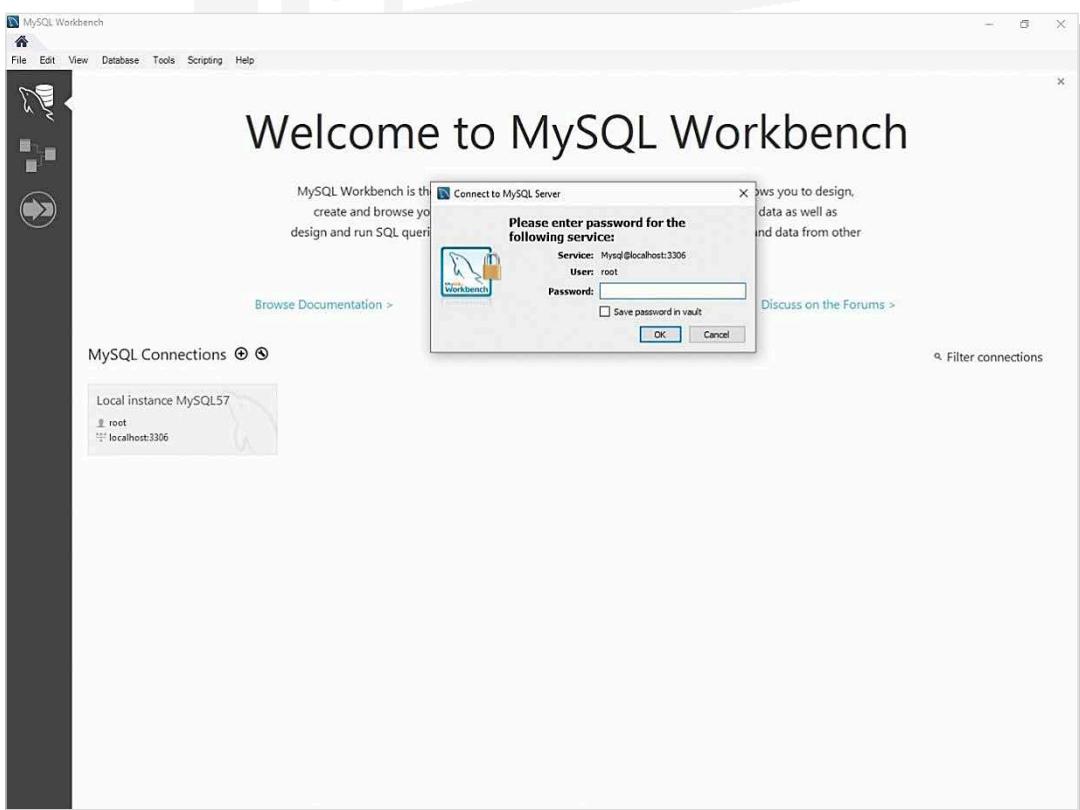


4. Una vez finalizado con éxito el paso anterior, ya tendremos disponible MySQL Workbench en nuestro sistema.





5. Ya podemos abrir nuestra herramienta de cliente MySQL Workbench para conectarnos al servidor que gestiona la BD. Workbench nos solicitará la contraseña que pusimos anteriormente durante el proceso de instalación de MySQL Server:



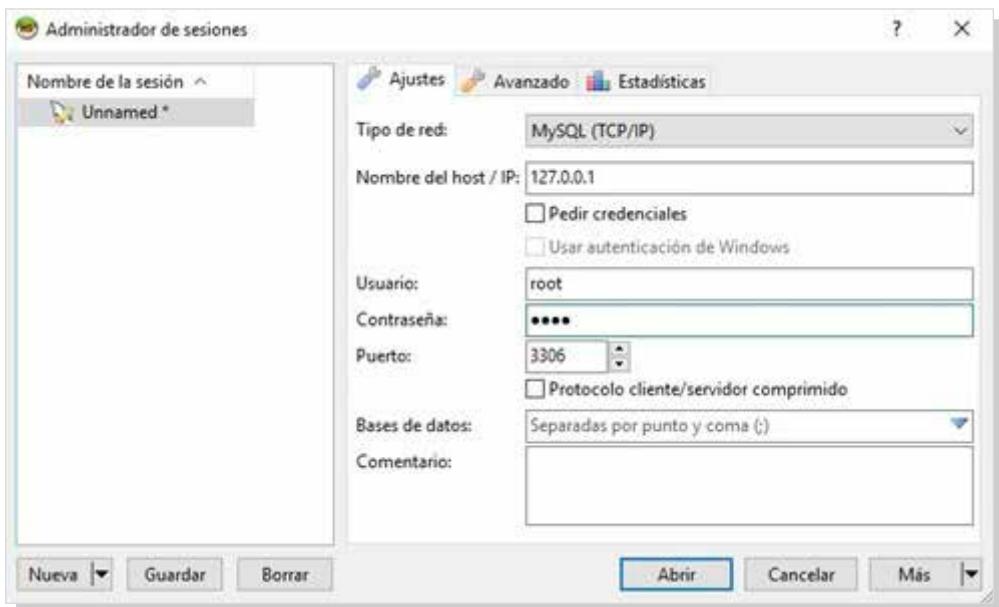
En caso de que queramos usar el software cliente HeidiSQL, lo primero que debemos hacer es descargarlo desde su página oficial: <https://www.heidisql.com/>

The screenshot shows the HeidiSQL website homepage. At the top, there's a green header with the HeidiSQL logo and navigation links: Home, Downloads, Images, Forum, Donate, Bugtracker, Help. A prominent "Start Download" button with a Microsoft logo and the text "Microsoft™ 64/32 Bit Driver Download for Windows." is displayed. To the right of the download button is a blue "OPEN" button. Below the header, there's a section titled "What's this?" which includes a brief description of HeidiSQL as free software for managing MariaDB, MySQL, Microsoft SQL, PostgreSQL, and SQLite databases. It also features two screenshots of the HeidiSQL interface: one showing a complex database structure and another showing a connection configuration dialog. To the right of these screenshots are three horizontal bar charts showing the number of HeidiSQL users by server type and version. The first chart is for MySQL, the second for MS SQL, and the third for PostgreSQL. The MySQL chart shows versions from 5.1 to 10.4. The MS SQL chart shows versions from 8.0 to 20.19. The PostgreSQL chart shows versions from 9.4 to 12.4.

1. Cuando accedamos a la página oficial del cliente pulsaremos en la sección de **Descargas**.

The screenshot shows the "Downloads" section of the HeidiSQL website. The header is identical to the main page. Below the header, there's a large heading "Download HeidiSQL 11.0" and a note "Release date: 17 Mar 2020.". On the left, there's a list of download options: "Installer, 32/64 bit combined", "Portable version (zipped): 32 bit, 64 bit", "Microsoft Store App, 32 bit", "Sourcecode", and "Previous releases". On the right, there's a green "Donate" button with the text "Support making HeidiSQL better and better". Below the download options, there's a section titled "Compatibility notes" with a bulleted list of requirements. At the bottom, there's a section titled "Nightly Builds of heidisql.exe" with a note about automatically compiled main executable and installers. There's also a footer note about context menu items and a status bar click event.

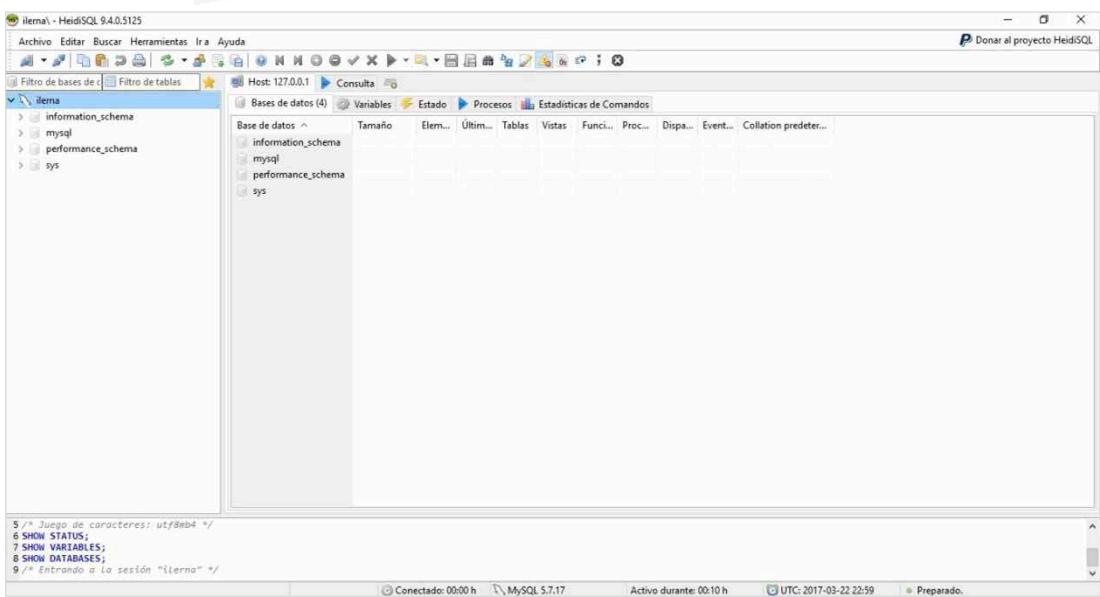
2. Una vez descargado y ejecutado el instalable, debemos aceptar las condiciones y los pasos por defecto. Pulsemos **Siguiente** hasta que ejecutemos el programa cliente y aparezca la captura de pantalla siguiente.



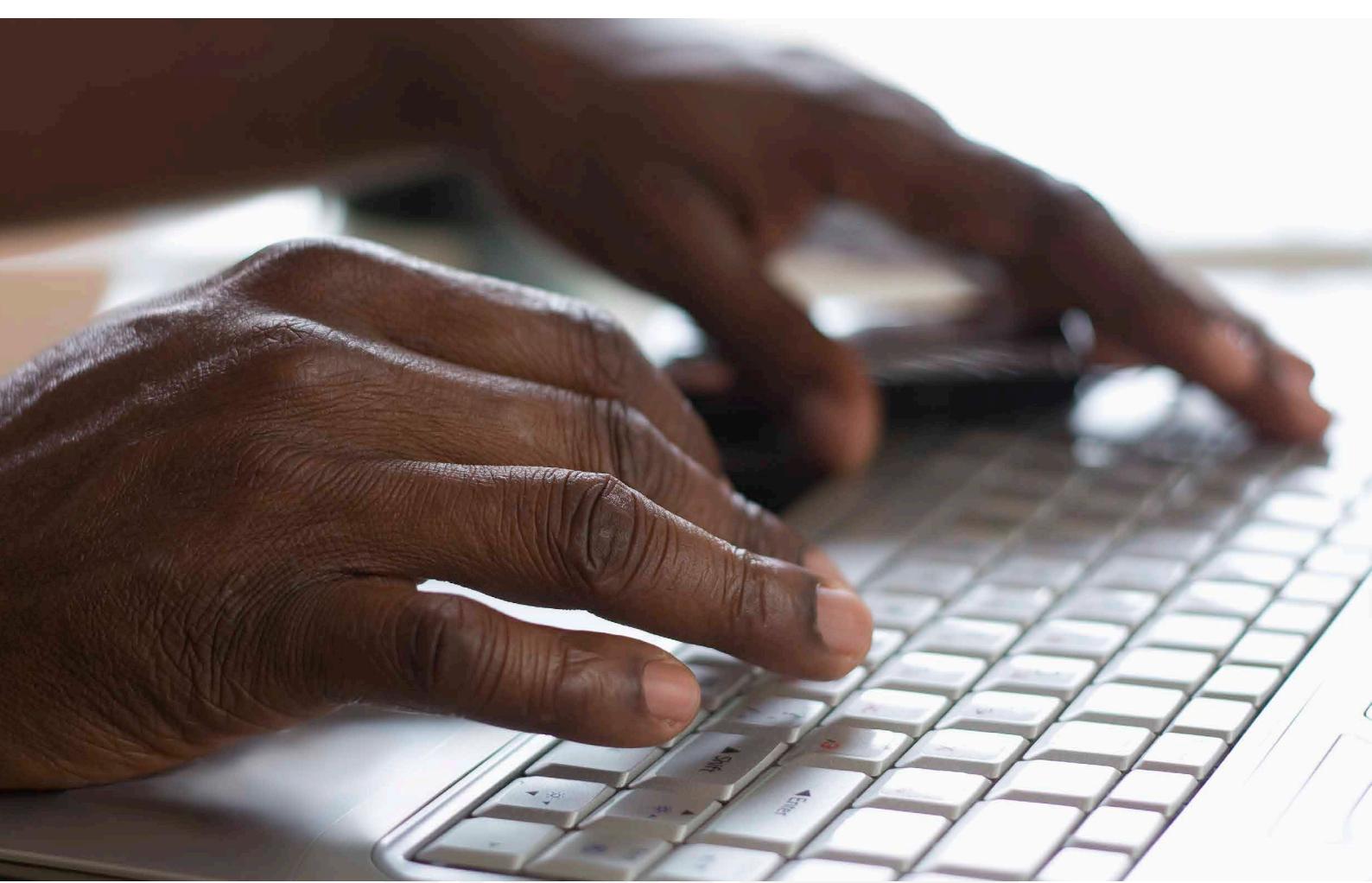
En esta ocasión pulsaremos el botón de **Nueva** para crear una nueva conexión al servidor. Dejaremos la IP del servidor (que nos aparecerá por defecto 127.0.0.1). A continuación, introduciremos la contraseña que definimos para el servidor. En nuestro caso **root**.

Una vez configurada la conexión, podremos cambiarle el nombre. Cuando esté todo listo, pulsaremos en **Abrir**.

El entorno gráfico del cliente de SQL es el siguiente:



Una vez finalicemos este paso, daremos por concluida la instalación.



## 5.3. LENGUAJE DE DEFINICIÓN DE DATOS (DDL)

De manera genérica, el **lenguaje de definición de datos (DDL)** es un lenguaje de programación para definir estructuras de datos. Aplicado a SQL, por DDL entendemos el lenguaje que el SGBD nos proporciona para definir las estructuras de las BBDD. Es decir, definir el continente (no el contenido). Esta parte del lenguaje se encarga de **definir, modificar y eliminar las estructuras básicas de la BD**, como tablas, vistas, índices y otros objetos relacionados con la definición de la BD.

Nota: debemos tener en cuenta que a partir del lenguaje estándar SQL, los SGBD suelen usar lenguajes derivados como los que usamos en varios ejemplos de este libro, como MySQL u ORACLE. El fin de dichos sublenguajes es el mismo: son muy similares entre sí, pero algunas de sus sintaxis pueden variar.

En este apartado sobre DDL analizaremos varias sentencias SQL, que podríamos agrupar en tres tipos: CREATE, ALTER y DROP.

## Crear BD

Para crear una BD usamos el comando “CREATE DATABASE”. Hay que diferenciar lo que es crear una BD y crear una tabla. **En una BD pueden existir muchas tablas.**

La sintaxis SQL para crear una BD es:

```
CREATE DATABASE nombreBaseDatos;
```

### Ejemplo en MySQL:

```
CREATE DATABASE territorio_info;
```

En este ejemplo estaríamos creando una BD llamada “territorio\_info”, y en el momento de crearla estaría vacía, es decir, de momento no tendría ninguna tabla.

## Creación de tablas dentro de una BD

Una vez ya hemos creado una BD, entramos en ella y creamos las tablas correspondientes. Para la creación de una tabla tendremos en cuenta el siguiente formato:

```
CREATE TABLE nombre_tabla (
    columna1 tipodato,
    columna2 tipodato,
    columna3 tipodato,
    {RESTRICCIONES} ) ;
```

Para la elección del nombre de tabla en SQL debemos cumplir las siguientes condiciones:

- Debe identificar el contenido.
- Máximo de 30 caracteres.
- No pueden ser palabras reservadas por el SGBD.
- Debe comenzar por una letra.
- Solo se permiten las letras, los números, el signo de subrayado “\_” y los caracteres “\$” y “#” (aunque no son los más recomendados).
- El nombre de la tabla debe de ser único.

### Ejemplo en MySQL:

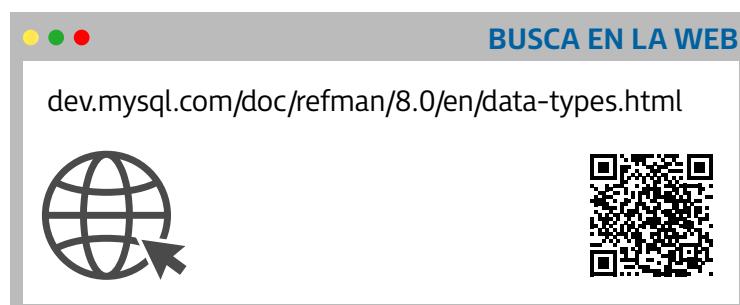
```
CREATE TABLE municipios (
    id INT PRIMARY KEY,
    nombre CHAR(20) NOT NULL,
    poblacion INT DEFAULT 1000
);
```

- Con PRIMARY KEY indicamos al SGBD que ese atributo es la clave primaria de la tabla.
- Con NOT NULL indicamos al SGBD que ese atributo siempre debe tener algún valor, es decir, no puede crearse un registro en esa tabla sin asignar un valor a ese atributo.
- Con DEFAULT indicamos al SGBD el valor que queremos que ese atributo tenga por defecto.

El nombre de los tipos de variables puede cambiar según qué variante de SQL usemos. Algunos de los existentes en MySQL son:

TIPO de dato MySQL	Descripción
<b>INT</b>	Valor numérico entero
<b>FLOAT(m,d)</b>	Valor numérico en coma flotante
<b>SMALLINT</b>	Valor numérico de 2 bytes de tamaño
<b>DATE</b>	Dato tipo fecha
<b>DATETIME</b>	Dato que almacena fecha y hora
<b>CHAR</b>	Cadena de caracteres de tamaño fijo
<b>VARCHAR()</b>	Cadena de caracteres de tamaño variable
<b>BOOLEAN</b>	Valor booleano

Se puede consultar la totalidad de tipos de datos en MySQL en la documentación que ofrece la página oficial de MySQL:



En el estándar SQL o en variantes como Oracle SQL, los tipos de datos pueden codificarse de manera distinta:

DESCRIPCIÓN	TIPOS ESTÁNDAR SQL	ORACLE SQL
TEXTO		
Texto de anchura fija	<ul style="list-style-type: none"> <li>• CHARACTER (<a href="#">n</a>)</li> <li>• CHAR (<a href="#">n</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• CHAR (<a href="#">n</a>)</li> </ul>
Texto de anchura variable	<ul style="list-style-type: none"> <li>• CHARACTER VARYING (<a href="#">n</a>)</li> <li>• VARCHAR (<a href="#">n</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• VARCHAR2 (<a href="#">n</a>)</li> </ul>
Texto de anchura fija para caracteres nacionales	<ul style="list-style-type: none"> <li>• NATIONAL CHARACTER (<a href="#">n</a>)</li> <li>• NATIONAL CHAR (<a href="#">n</a>)</li> <li>• NCHAR (<a href="#">n</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• NCHAR (<a href="#">n</a>)</li> </ul>
Texto de anchura variable para caracteres nacionales	<ul style="list-style-type: none"> <li>• NATIONAL CHARACTER VARYING (<a href="#">n</a>)</li> <li>• NATIONAL CHAR VARYING (<a href="#">n</a>)</li> <li>• NCHAR VARYING (<a href="#">n</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• NVARCHAR2 (<a href="#">n</a>)</li> </ul>

DESCRIPCIÓN	TIPOS ESTÁNDAR SQL	ORACLE SQL
NÚMEROS		
Enteros pequeños (2 bytes)	<ul style="list-style-type: none"> <li>• SMALLINT</li> </ul>	
Enteros normales (4 bytes)	<ul style="list-style-type: none"> <li>• INTEGER</li> <li>• INT</li> </ul>	
Enteros largos (8 bytes)	<ul style="list-style-type: none"> <li>• BIGINT</li> </ul>	
Enteros precisión decimal		<ul style="list-style-type: none"> <li>• NUMBER (<a href="#">n</a>)</li> </ul>
Decimal de coma variable	<ul style="list-style-type: none"> <li>• FLOAT</li> <li>• DOUBLE</li> <li>• DOUBLE PRECISION</li> <li>• REAL</li> </ul>	
Decimal de coma fija	<ul style="list-style-type: none"> <li>• NUMERIC (<a href="#">m,d</a>)</li> <li>• DECIMAL (<a href="#">m,d</a>)</li> </ul>	<ul style="list-style-type: none"> <li>• NUMBER (<a href="#">m,d</a>)</li> </ul>

DESCRIPCIÓN	TIPOS ESTÁNDAR SQL	ORACLE SQL
FECHAS		
Fechas	<ul style="list-style-type: none"> <li>• DATE</li> </ul>	<ul style="list-style-type: none"> <li>• DATE</li> </ul>
Fecha y hora	<ul style="list-style-type: none"> <li>• TIMESTAMP</li> </ul>	<ul style="list-style-type: none"> <li>• TIMESTAMP</li> </ul>

DESCRIPCIÓN	TIPOS ESTÁNDAR SQL	ORACLE SQL
BOOLEANOS Y BINARIOS		
Lógico	<ul style="list-style-type: none"> <li>• BOOLEAN</li> <li>• BOOL</li> </ul>	
Binarios	<ul style="list-style-type: none"> <li>• BIT</li> <li>• BIT VARYING (<a href="#">n</a>)</li> <li>• VARBIT (<a href="#">n</a>)</li> </ul>	
DATOS DE GRAN TAMAÑO		
Texto de gran longitud	<ul style="list-style-type: none"> <li>• CHARACTER LARGE OBJECT</li> <li>• CLOB</li> </ul>	<ul style="list-style-type: none"> <li>• LONG (en desuso)</li> <li>• CLOB</li> </ul>
Binario de gran longitud	<ul style="list-style-type: none"> <li>• BINARY LARGE OBJECT</li> <li>• BLOB</li> </ul>	<ul style="list-style-type: none"> <li>• RAW (en desuso)</li> <li>• LONG RAW (en desuso)</li> <li>• BLOB</li> </ul>

### Borrado de las tablas

Para borrar una tabla concreta debemos usar la siguiente sintaxis:

```
DROP TABLE nombre_tabla;
```

#### Ejemplo en MySQL:

```
DROP TABLE territorio_info;
```

Al eliminar una tabla, también desaparece su contenido. Si a la sentencia le añadimos la cláusula de eliminación en cascada también se eliminarán las restricciones de integridad referencial que afecten a la clave principal de la tabla que se va a borrar.

Una operación de borrado es **irreversible**. Por ese motivo debemos estar seguros antes de llevarla a cabo.

### Modificación de la tabla

Para modificar una tabla usamos el comando ALTER TABLE. Existen varias opciones para hacerlo. Podemos añadir, modificar o eliminar columnas de una tabla, así como activar o desactivar restricciones.

En el siguiente formato podemos ver todas las opciones de esta cláusula:

```
ALTER TABLE nombre_tabla {  
    [ADD (columna)]  
    [MODIFY/CHANGE (colum [...])]  
    [DROP COLUMN (colum....)]  
    [ADD CONSTRAINT restricción]  
    [DROP CONSTRAINT restricción]  
    [DISABLE CONSTRAINT restricción]  
    [ENABLE CONSTRAINT restricción]}
```

A continuación, veremos en detalle algunas de las opciones de esta sentencia:

### Añadir, modificar o eliminar columnas

- **ADD:**

- Si la columna no tiene restricción NOT NULL se añade sin más.
- Si tiene restricción NOT NULL se añade, se le asigna un valor y, por último, se le asigna la restricción.

- **MODIFY/CHANGE:**

- Se puede aumentar o disminuir la longitud de la columna.
- Se puede aumentar o disminuir el valor de los decimales del campo.

- **DROP:**

- No se pueden eliminar todos los campos de una tabla.
- No se pueden borrar claves primarias referenciadas por una clave ajena de otra tabla.



### Ejemplos en MySQL:

Para modificar el nombre de una tabla usamos `ALTER TABLE nombre_tabla` y `RENAME nombre_nueva_tabla`, donde indicamos el antiguo y nuevo nombre que deseamos para una tabla. A continuación, mostramos un ejemplo del uso de este comando:

```
ALTER TABLE municipios RENAME municipio;
```

Para eliminar una columna de una tabla usamos el comando `ALTER TABLE... DROP COLUMN ...`, en el que indicamos la tabla a la que nos referimos y la columna que se queremos borrar:

```
ALTER TABLE municipios DROP COLUMN poblacion;
```

Para eliminar la clave primaria de una columna usamos el comando `ALTER TABLE... DROP PRIMARY KEY`. Ojo, se borra la condición de clave primaria de un atributo, no el atributo. Tan solo debemos indicar en el comando el nombre de la tabla:

```
ALTER TABLE municipios DROP PRIMARY KEY;
```

Para lo contrario, es decir, para dotar como clave primaria a un atributo de una tabla usamos el comando `ALTER TABLE ... ADD PRIMARY KEY( )`, indicando entre los paréntesis el atributo que se desea hacer clave primaria:

```
ALTER TABLE municipios ADD PRIMARY KEY(ID);
```

Para insertar un atributo nuevo, es decir, una columna nueva en una tabla usamos el comando `ALTER TABLE ... ADD ...`, indicando el nombre de la tabla, el nombre del atributo, su tipo de dato y las restricciones que pudiese tener, tal y como se realiza en la creación de la tabla:

```
ALTER TABLE municipios ADD extension INT;
```

Para modificar una columna podemos utilizar el comando `ALTER TABLE... MODIFY...`, donde se indica el nombre de la tabla y las nuevas propiedades de ese atributo. En este ejemplo dotamos de la propiedad de autoincrementarse a la columna del id:

```
ALTER TABLE municipios MODIFY id INT auto_increment;
```

**Nota:** en caso de que ya hubiese registros en la tabla, la modificación se realizará si las nuevas propiedades de la columna son compatibles con los datos de los registros anteriormente insertados.



## Restricciones

Las restricciones en las BBDD hacen que las estructuras sean más eficaces, lo cual facilita la manipulación de los datos por parte de los usuarios.

Hay distintos tipos de restricciones:

- Clave primaria (PRIMARY KEY)
- Clave ajena (FOREIGN KEY)
- Columnas no nulas (NOT NULL)
- Valores por defecto (DEFAULT)
- Verificador de condiciones (CHECK)
- Valor único (UNIQUE)

A continuación, vamos a ver cómo podemos definir, manipular o eliminar estas restricciones. Cuando creamos una tabla podemos crear restricciones para sus campos de dos formas distintas:

- **En la propia definición del campo**

```
CREATE TABLE nombre_tabla (
    nomcolumn_1 TIPO_DATO() TIPO_RESTRICCIÓN,
    .... ) ;
```

- **Al final de la creación de la tabla**

```
CREATE TABLE nombre_tabla(column1 tipo_dato(),
.....,
[CONSTRAINT nombre_restriccion] UNIQUE (columna [columna])
,.....);
```

Para terminar, vamos a ver unos ejemplos de **cómo serían las distintas restricciones** que podemos definir en MySQL:

1. 

```
CREATE TABLE empleados (
    dni INT PRIMARY KEY,
    nombre TEXT(15),
    apellidos TEXT(30),
    cod_postal INT CONSTRAINT cod REFERENCES
    codigos_postales);
```

En esta tabla la restricción de clave primaria se aplica al campo *DNI*, y la restricción de clave foránea al campo *COD\_POSTAL*, referido a una supuesta tabla que se llama *CODIGOS\_POSTALES*.

2. 

```
CREATE TABLE empleados (
    dni INT PRIMARY KEY,
    nombre TEXT(15) NOT NULL,
    apellidos TEXT(30),
    cod_postal INT CONSTRAINT cod REFERENCES
    codigos_postales,
    salario CURRENCY NOT NULL,
    CONSTRAINT cf_cp FOREIGN KEY(cod)
    REFERENCES codigos_postales);
```

En este caso hemos creado la misma tabla que en el ejemplo anterior, pero hemos añadido restricciones de forma diferente. Tanto la clave primaria como los campos obligatorios (NOT NULL) se implementan a nivel de campo, pero la restricción de la clave foránea se está creando a nivel de tabla, donde iniciamos la restricción con la palabra reservada (CONSTRAINT) y, a continuación, un nombre que le damos a la restricción (cf\_cp). Seguidamente, nos encontramos con el tipo de restricción (FOREIGN KEY) y como es una clave ajena, entonces le tenemos que indicar a qué clave de otra tabla se refiere (REFERENCES).

También podemos hacer uso de las restricciones en las sentencias ALTER para añadir, modificar o eliminar alguna restricción:

3. 

```
ALTER TABLE empleados
ADD COLUMN cod_oficina INT;
```

Solo estamos añadiendo el campo *Código de oficina* a la tabla *empleados*.

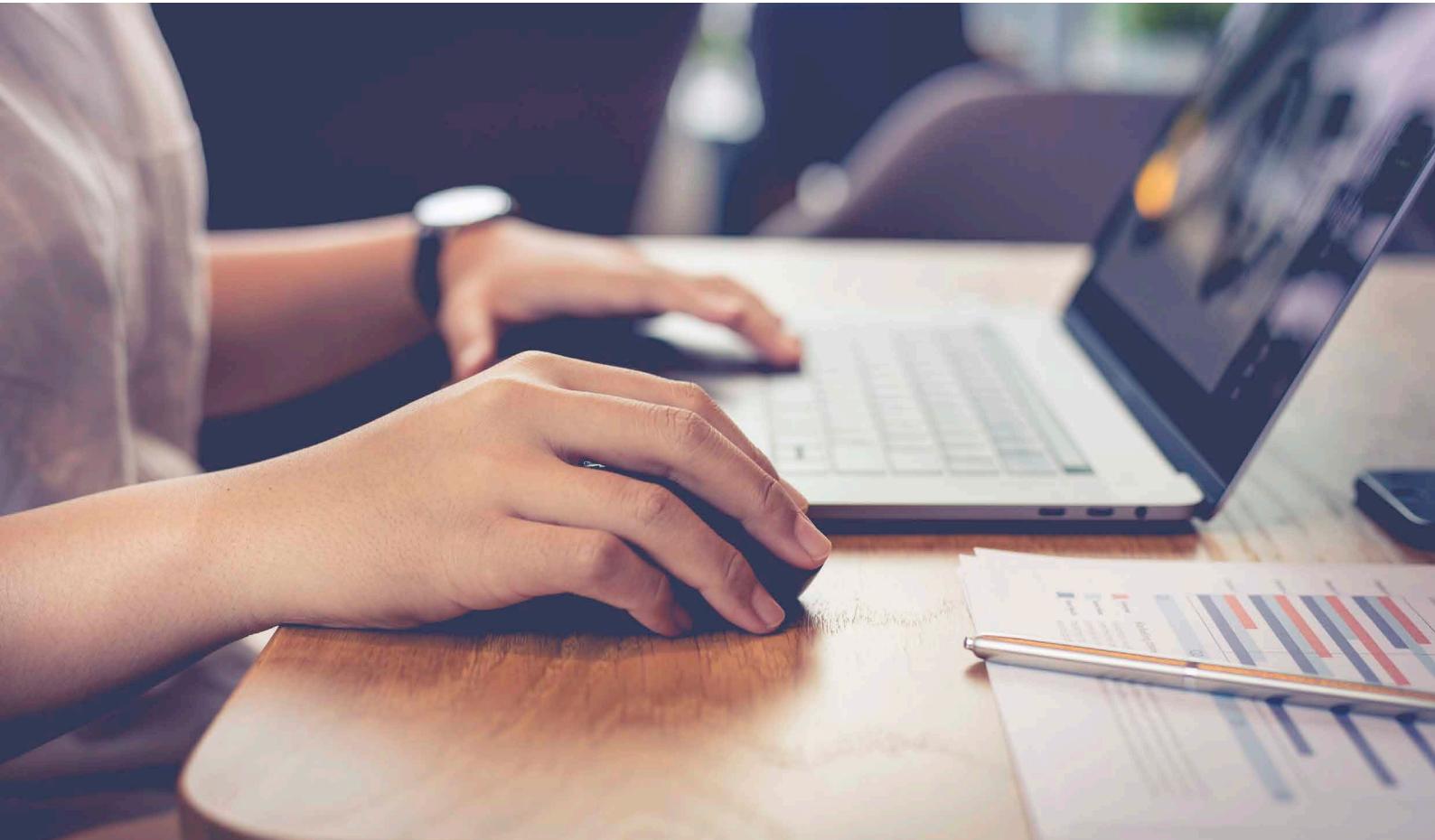
```
4. ALTER TABLE empleados  
    ADD CONSTRAINT cod_oficina  
    FOREIGN KEY (código)  
    REFERENCES oficinas;
```

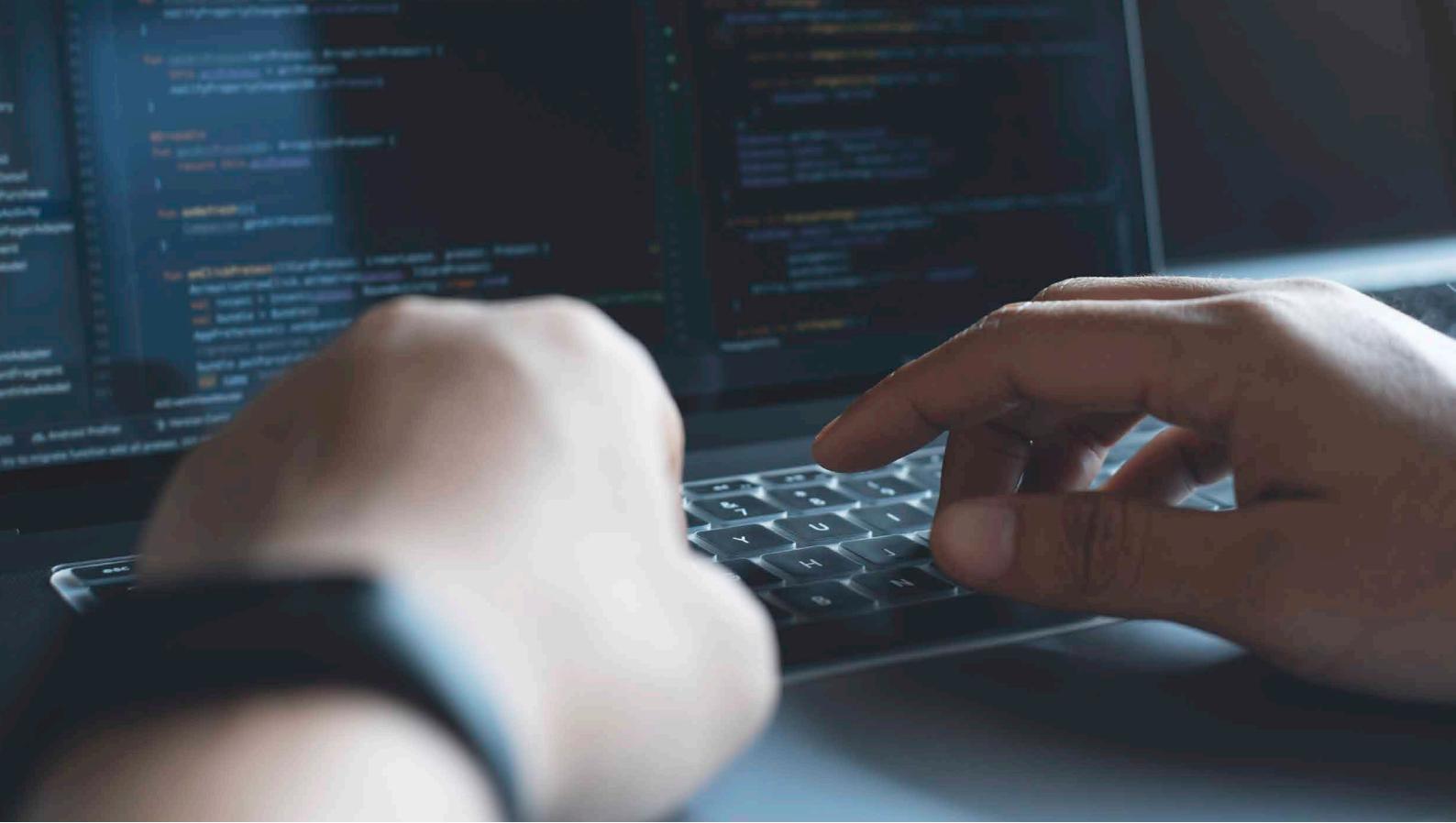
En este caso, podemos ver cómo se modifica una tabla añadiéndole una restricción a un campo existente, ya que ahora es una clave foránea de una clave principal (*Código*) de la tabla *Oficinas*.

```
5. ALTER TABLE empleados  
    ADD CONSTRAINT u_oficina UNIQUE (cod_oficina);
```

En este caso, además de que el código de la oficina es clave ajena, le estamos obligando a que su valor sea único, es decir, que en cada oficina solo puede trabajar un empleado.

Cuando trabajamos con sentencias en SQL, para diferencias es mejor utilizar las cláusulas SQL de los nombres que el administrador de campos y restricciones. Normalmente se emplean las mayúsculas para las sentencias y las palabras reservadas y las minúsculas, para los nombres definidos por el usuario.





## 5.4. LENGUAJE DE MANIPULACIÓN DE DATOS (DML)

El **DML** (Data Manipulation Language) es un lenguaje de manipulación de datos que nos ofrece la posibilidad de consultar, insertar, eliminar y modificar la información contenida en una BD.

Gracias a este lenguaje tenemos la posibilidad de introducir nuevos registros en las tablas, modificar la información de dichos registros, consultarlos o incluso borrar los registros de las tablas.

En este apartado analizaremos varias sentencias SQL de DML, que podríamos agrupar en cuatro tipos:

- SELECT
- INSERT
- UPDATE
- DELETE

### 5.4.1. CONSTRUCCIÓN DE SENTENCIAS DE INSERCIÓN

La inserción de datos es una operación obligatoria en las BBDD. Una vez ya hemos creado la BD y las tablas, el siguiente paso es insertar información en las tablas, es decir, insertar registros (tuplas) en las tablas. Como es lógico, si tuviéramos las tablas creadas pero sin información en su

interior, no podríamos ni modificar, ni consultar, ni borrar información alguna. Es por ello que la primera sentencia que vamos a ver es la de *insertar*.

La sintaxis se desarrollaría de la siguiente forma:

```
INSERT INTO nombre_tabla (columna1, columna2,...)
VALUES (valor1, valor2,...);
```

### Ejemplo en MySQL:

Pongamos que partimos de la siguiente tabla que tiene información relativa a los pueblos y ciudades de un país:

```
CREATE TABLE municipios (
    id INT PRIMARY KEY,
    nombre CHAR(20) NOT NULL,
    poblacion INT DEFAULT 0
) Engine=innodb;
```

**Nota:** "Engine=innodb" se añade opcionalmente para indicar a MySQL el motor de almacenamiento que deseamos. *innodb* es adecuado para trabajar con claves foráneas, transacciones o bloqueos, entre otros.

La inserción de un registro en dicha tabla podría ser la siguiente:

```
INSERT INTO municipios (id, nombre, poblacion) VALUES (11, "Valencia", 794000);
```

Existe otra posibilidad a la hora de introducir valores a todas las columnas de una tabla. No hace falta especificar el nombre de las columnas, como en el ejemplo anterior. El orden de los valores debe coincidir con el orden de los campos creados en la tabla.

```
INSERT INTO nombre_tabla
VALUES (valor1, valor2, valor3, ...);
```

### Ejemplo en MySQL:

Podríamos realizar una consulta como la del ejemplo anterior, pero sin especificar el nombre de cada atributo, siempre y cuando pongamos los valores del nuevo registro en el mismo orden que se estableció en los atributos de la tabla.

```
INSERT INTO municipios VALUES (12, "Málaga",
574000);
```

## 5.4.2. CONSTRUCCIÓN DE SENTENCIAS DE MODIFICACIÓN

Podemos realizar una modificación de uno o varios datos de un registro concreto de una tabla. Para ello usamos el comando UPDATE.

```
UPDATE nombre_tabla
SET columnal = valor1, columnas = valor2, ...
WHERE condicion;
```

Debemos tener cuidado a la hora de utilizar esta sentencia, ya que en la cláusula “WHERE” tenemos que introducir la condición que deben cumplir los datos para actualizarse. Si omitimos esta cláusula, entonces se van a modificar todos los registros de la tabla.

### Ejemplo en MySQL:

Imaginemos que tenemos la siguiente información en una tabla:

ID	NOMBRE	POBLACIÓN
11	Valencia	794000
12	Málaga	574000

Al realizar la siguiente sentencia:

```
UPDATE municipios SET poblacion = 800000
WHERE id = 11;
```

Obtendríamos una actualización del campo población del registro, cuyo id tiene el valor de 11, de modo que el contenido de la tabla quedaría así:

ID	NOMBRE	POBLACIÓN
11	Valencia	800000
12	Málaga	574000

Sin embargo, si hacemos una actualización sin especificar la cláusula del WHERE, actualizaremos el valor de todos los registros de la tabla. Por ejemplo, imaginemos que sobre la tabla anterior queremos cambiar el valor del atributo *nombre* de “Málaga” a “Sevilla”, pero no especificamos nada en la parte del WHERE, de modo que indicáramos esta sentencia:

```
UPDATE municipios SET nombre = "Sevilla";
```

La tabla resultante sería la siguiente:

ID	NOMBRE	POBLACIÓN
11	Sevilla	800000
12	Sevilla	574000

Habríamos cambiado el valor del atributo *nombre* de todos los registros de la tabla y no solo del registro que nos interesaba.

La manera correcta de hacerlo sería la siguiente:

```
UPDATE municipios SET nombre = "Sevilla"
WHERE nombre = "Málaga";
```

### 5.4.3. CONSTRUCCIÓN DE SENTENCIAS DE ELIMINACIÓN

En SQL utilizamos el comando DELETE FROM para eliminar registros de una tabla que cumplan una determinada condición. Como en el anterior apartado, si omitimos la cláusula WHERE, se borrarán todas las tuplas de la tabla.

```
DELETE FROM nombre_tabla
WHERE condicion;
```

#### Ejemplo en MySQL

Imaginemos que partimos de esta tabla:

ID	NOMBRE	POBLACIÓN
11	Valencia	800000
12	Málaga	574000
13	Zaragoza	674000

Y supongamos que deseamos eliminar el registro cuya id es 13. Para conseguirlo, deberíamos realizar la siguiente sentencia:

```
DELETE FROM municipios WHERE id = 13;
```

En MySQL, para eliminar todos los registros de una tabla, y por tanto dejarla vacía, podemos usar DELETE FROM sin la cláusula WHERE, pero disponemos también del comando TRUNCATE, que permite eliminar todos los registros de una tabla de una manera más rápida y eficiente:

```
TRUNCATE municipios;
```

## 5.4.4. CONSTRUCCIÓN DE CONSULTAS. LA SELECCIÓN SIMPLE

La operación de consultar una BD se encuentra entre las más utilizadas. Se realiza con el comando `SELECT ... FROM ...`. Gracias a este comando podemos consultar la información de los registros. Una consulta en SQL tiene básicamente el siguiente formato:

```
SELECT column1, column2...
FROM nombre_tabla;
```

A medida que vayamos avanzando en los distintos apartados, veremos la posibilidad de ampliar esta cláusula.

La sentencia `SELECT-FROM` posee varias cláusulas posibles, que podemos añadir para crear ciertas restricciones. Por ejemplo, que solo aparezcan los registros de una tabla que cumpliesen con una condición o ciertos atributos por cada registro.

En la primera parte de la sentencia, en la sección de **SELECT**, tendremos que indicar el nombre de los campos de los cuales deseamos tener información.

En la parte del **FROM**, pondremos el nombre de la tabla que interviene en la consulta. Por tanto, los campos (atributos) que hayamos indicado en la parte del `SELECT` deben pertenecer a la tabla que indiquemos en `FROM`.

### Ejemplo en MySQL

Supongamos que partimos de la tabla *municipios* con el siguiente contenido:

ID	NOMBRE	POBLACIÓN
11	Valencia	800000
12	Málaga	574000
13	Zaragoza	674000
14	Sevilla	688000

Para obtener toda la información de esta tabla, es decir, para obtener todos los registros de la tabla junto con todos sus campos deberíamos realizar la siguiente orden:

```
SELECT * FROM municipios;
```

```

$re = array();
$re = mysql::query("SELECT * FROM image_date ORDER BY shot_date DESC");

while($day = mysql::fetch($re)) {
    $day->list = array();
    $shots_result = mysql::query("SELECT DISTINCT(studio) as studio, COUNT(*) as count FROM image WHERE day_id = '$day->id' AND enabled='Y' GROUP BY studio");
    while($studio_list = mysql::fetch($shots_result)) {
        $day_info = metadata::day_info($day->shot_date, $studio_list->studio, "quick");
        $tmp_studio_list[] = array("studio" => $studio_list->studio, "count" => $studio_list->count, "title" => $day_info->title);
    }
    $day->studio_list = $tmp_studio_list;
    $return[$day->shot_date] = $day;
}

return $return;
}

static function day_images_list($date, $studio) {
    global $global_studio_list;
    if(!in_array($studio, $global_studio_list)) die("error studio");
    $date = mysql::escape($date);
    if(mysql::count("image_date", "shot_date = '$date'") == 0) die("date not found");
    $studio = intval($studio);

    $return = array();

    $result = mysql::query("SELECT image.id as image_id, image, image.date WHERE image.date.id=image.day_id AND image.date.shot_date='$date' AND image.enabled='Y'");
    while($image = mysql::fetch($result)) {
        $image->copyright = metadata::get_copyright($image->image_id);
        $image->models = metadata::get_models($image->image_id);
        $return[$image->image_id] = $image;
    }
}

```

1273

1274

1275

1276

1277

1278

1279

Esta sentencia nos devolvería los cuatro registros de la tabla con tres campos cada uno, tal como se muestra en la tabla anterior.

Con el símbolo \* indicamos que deseamos ver todos los campos de cada registro. Sin embargo, si quisiéramos ver todos los registros de la tabla, pero que solo aparezca el nombre de cada uno, deberíamos realizar la siguiente sentencia:

```
SELECT nombre FROM municipios;
```

Lo cual nos devolvería lo siguiente:

NOMBRE
Valencia
Málaga
Zaragoza
Sevilla

## 5.4.5. CONSTRUCCIÓN DE CONSULTAS DE SELECCIÓN CON RESTRICCIÓN Y ORDENACIÓN

Con la cláusula WHERE podemos definir la condición que deben cumplir los registros para que se muestren en la consulta.

```
SELECT columna1, columna2, ...
FROM nombre_tabla
WHERE condicion;
```

### Ejemplo en MySQL:

Supongamos que partimos de la tabla municipios con sus cuatro registros (Valencia, Málaga, Zaragoza y Sevilla) y queremos que el SGBD nos muestre el nombre y número de habitantes de aquellos municipios que posean más de 600.000 habitantes. Para tal propósito, podemos realizar la siguiente sentencia:

```
SELECT nombre, poblacion FROM municipios
WHERE poblacion > 600000;
```

Esto nos devolvería la siguiente información:

NOMBRE	POBLACIÓN
Valencia	800000
Zaragoza	674000
Sevilla	688000

El municipio *Málaga* no aparece porque no cumple la condición de tener el campo *Población* con un valor mayor a 600.000, tal como se determina en la cláusula WHERE.

Con la cláusula **ORDER BY** podemos ordenar los datos resultantes de una consulta. Podemos hacerlo de manera ascendente o descendente. Su sintaxis es la siguiente:

```
SELECT columna1, columna2, ...
FROM nombre_tabla
ORDER BY columna1, columna2, ... ASC|DESC;
```

El valor por defecto sería *Ordenar de manera ascendente*, por tanto, no habría que indicarlo.

### Ejemplo en MySQL

Supongamos que queremos que el SGBD nos devuelva el id y nombre de todos los municipios de la tabla ordenados por número de habitantes, de menor a mayor. Es decir, de forma ascendente. Para tal propósito usaríamos la siguiente sentencia:

```
SELECT id, nombre FROM municipios ORDER BY
poblacion ASC;
```

Lo que nos devolvería los registros ordenados tal que así:

ID	NOMBRE
12	Málaga
13	Zaragoza
14	Sevilla
11	Valencia

Las dos cláusulas que hemos visto en este apartado se pueden complementar con una consulta con restricción y que sus valores salgan ordenados.

```
SELECT column1, column2, ...
FROM nombre_tabla
WHERE condicion
ORDER BY column1, column2, ... ASC|DESC;
```

### Ejemplo en MySQL

Podríamos combinar dos de los requerimientos anteriores, es decir, que solo aparezcan los municipios de más de 600.000 habitantes y que aparezcan ordenados de menor a mayor. Para tal propósito, usaríamos esta sentencia:

```
SELECT nombre, poblacion FROM municipios
WHERE poblacion > 600000 ORDER BY poblacion
ASC;
```

Lo que nos devolvería la siguiente lista:

ID	NOMBRE
13	Zaragoza
14	Sevilla
11	Valencia



#### 5.4.6. CONSTRUCCIÓN DE CONSULTAS DE SELECCIÓN UTILIZANDO CLÁUSULAS DEL LENGUAJE PARA LA AGRUPACIÓN

La sentencia **GROUP BY** se utiliza para agrupar por algún campo de la tabla. Se puede usar juntamente con una función de agregación, aunque no es obligatorio:

- **COUNT**: devuelve la cantidad de registros.
- **MAX**: devuelve el valor máximo de los registros en el campo seleccionado.
- **MIN**: devuelve el valor mínimo de los registros en el campo seleccionado.
- **SUM**: devuelve la suma de los registros en el campo escogido.
- **AVG**: retorna la media aritmética de los valores del campo.

La sintaxis se desarrollaría de la siguiente manera:

```
SELECT column1, funcion_agregacion(nombre_columna)
      FROM nombre_tabla
      WHERE condicion
      GROUP BY nombre_columna;
```

#### 5.4.7. CONSTRUCCIÓN DE CONSULTAS UTILIZANDO LAS FUNCIONES AÑADIDAS DEL LENGUAJE TRATANDO LOS VALORES NULOS

Un campo con valor **NULL** contiene un valor nulo, es decir, sin valor. En este apartado vamos a utilizar las cláusulas **IS NULL** o **IS NOT NULL**, ya que no es posible utilizar operadores de comparación con valores **NULL**.

Veamos la sintaxis de una consulta evaluando valores nulos:

```
SELECT columnal,columna2...
FROM nombre_tabla
WHERE columna IS NULL / IS NOT NULL;
```

#### 5.4.8. CONSTRUCCIÓN DE CONSULTAS PARA CONSULTAR MÁS DE UNA TABLA

El apartado de las consultas se puede desarrollar ampliamente debido a la existencia de numerosas sentencias que podemos utilizar al diseñarlas. El uso de más de una tabla en una consulta hace que esta operación sea mucho más completa. La sintaxis para realizar una consulta donde intervenga más de una tabla es la siguiente:

```
SELECT tabla.columnal,tabla.columna2...
FROM nombre_tabla1, nombre_tabla2
WHERE tabla1.campo_clave=tabla2.campo_clave_ajena;
```

Por ejemplo, si quisieramos saber el departamento de un empleado llamado *Juan Pérez*, lo desarrollaríamos de la siguiente manera:

```
SELECT e.nombre, e.apellidos, d.nombre
FROM empleado e, departamento d
WHERE d.cod=e.dep AND e.nombre="JUAN" AND e.apellido= "PEREZ";
```

#### 5.4.9. CONSTRUCCIÓN DE SUBCONSULTAS

La construcción de subconsultas se realiza cuando queremos que sobre una consulta intervengan más de dos tablas. Para diseñar una consulta con varias tablas a la vez podemos combinar consultas sencillas entre ellas, siempre que tengamos un campo que pueda unir ambas consultas.

En el ejemplo siguiente podemos observar una subconsulta donde se consultan los códigos de los libros alquilados por la socia *Paula Robinson González*:

```
SELECT cSignatura
FROM TPrestamo
WHERE Cnif = (SELECT Cnif
               FROM TSocio
              WHERE cApellidos = "Robinson González"
                AND cNombre = "Paula");
```



### ponte a prueba

A través del SGBD MySQL queremos crear una tabla con el nombre “profesor”, que tenga los atributos de identificador, edad y asignatura. ¿Cómo lo haremos?

- a) CREATE TABLE profesor (identificador INT PRIMARY KEY, edad INT) ENGINE=INNODB;
- b) CREATE TABLE profesor (identificador INT PRIMARY KEY, edad INT, asignatura VARCHAR (15)) ENGINE=INNODB;
- c) USE TABLE profesor (identificador INT PRIMARY KEY, edad INT, asignatura VARCHAR (15)) ENGINE=INNODB;
- d) CREATE TABLE edad (profesor INT PRIMARY KEY, edad INT, identificador VARCHAR (15)) ENGINE=INNODB;

A través del SGBD MySQL, ¿qué sentencia usaremos si queremos borrar la tabla profesor?

- a) DROP TABLE profesor;
- b) DROP DATABASE profesor;
- c) DROP TABLE edad;
- d) DROP profesor;

Dada la siguiente tabla, ¿cuál de las siguientes sentencias utilizaremos para obtener el número de productos que tenemos en la tabla “Productos”?

IDProducto	Nombre	Descripción	Stock
1	Silla	Silla de madera	8
2	Mesa	Mesa de madera	5
3	Lámpara	Lámpara de aluminio	3

- a) SELECT COUNT(IDProducto) AS NúmeroProductos FROM Productos;
- b) SELECT SUM(IDProducto) AS NúmeroProductos FROM Productos;
- c) SELECT MAX(IDProducto) AS NúmeroProductos FROM Productos;
- d) SELECT SUM(Stock) AS NúmeroProductos FROM Productos;

Dada la siguiente tabla, ¿cuál de las siguientes sentencias utilizaremos para obtener la cantidad total de stock que disponemos en la tabla “Productos”?

IDProducto	Nombre	Descripción	Stock
1	Silla	Silla de madera	8
2	Mesa	Mesa de madera	5
3	Lámpara	Lámpara de aluminio	3

- a) SELECT SUM(Stock) AS NúmeroProductos FROM Productos;
- b) SELECT COUNT(Stock) AS NúmeroProductos FROM Productos;
- c) SELECT MAX(Stock) AS NúmeroProductos FROM Productos;
- d) SELECT SUM(IDProducto) AS NúmeroProductos FROM Productos;

## 5.5. EXTENSIONES Y OTRAS CLÁUSULAS DEL LENGUAJE

Con la profundización en el estudio del lenguaje SQL y sus derivados, vamos introduciendo más y más sentencias que nos ayudarán a implementar consultas muy complejas que satisfagan nuestras necesidades con respecto a las BBDD.

A continuación, vamos a ver la cláusula UNION que, como su nombre indica, nos ayuda a unir el resultado de dos o más consultas.

Presenta el siguiente formato:

```
SELECT nombre_columna(s) FROM tabla
UNION
SELECT nombre_columna(s) FROM tabla2;
```

Para que esta operación se pueda llevar a cabo de una forma satisfactoria debe cumplir las siguientes condiciones:

- Las consultas, de forma individual, deben tener el mismo número de campos y con los mismos tipos de datos representados en el mismo orden.
- Por defecto, este operador solo selecciona para sus resultados los valores distintos de los campos que lo forman. En el caso de desear valores duplicados, tendríamos que añadir la opción UNION ALL.

Ejemplo:

```
SELECT * FROM ALUMNOS WHERE FECHA_NACIMIENTO <='1970-01-01'
UNION
SELECT * FROM ALUMNOS WHERE FECHA_NACIMIENTO >='1980-01-01';
```

En el siguiente apartado vamos a ver los **INNER JOIN**, junto con sus variantes LEFT y RIGHT JOIN. El objetivo de todas las sentencias JOIN es calcular la intersección entre dos tablas o relaciones.

Es una segunda forma de realizar consultas donde intervengan varias tablas.

Su formato es el siguiente:

```
SELECT columna(s)
FROM tabla1
INNER JOIN tabla2
ON tabla1.columna=tabla2.columna; .
```

Una variedad de esta sentencia es RIGHT JOIN. Esta devuelve las tuplas comunes a las dos tablas y no comunes de la segunda tabla. Mientras que la cláusula LEFT JOIN devuelve las comunes y no comunes de la primera tabla.

A continuación, vamos a explicar las sentencias anteriores con un ejemplo:

```
SELECT columna(s)
FROM tabla1
RIGHT JOIN tabla2
ON tabla1.columna=tabla2.columna;
```

```
SELECT columna(s)
FROM tabla1
LEFT JOIN tabla2
ON tabla1.columna=tabla2.columna;
```

Para terminar, vamos a enumerar una serie de cláusulas relevantes a la hora de diseñar una consulta:

- **DISTINCT**: suele indicarse en la parte de SELECT para no devolver valores repetidos.

```
SELECT DISTINCT columna1, columna2, ...
FROM tabla1;
```

### Ejemplo:

Para seleccionar solo los valores DISTINCT de la columna *País* en la tabla *Consumidores*.

```
SELECT DISTINCT Pais FROM Consumidores;
```

- **HAVING**: cláusula que tenemos en cuenta a la hora de agrupar por alguna condición.

```
SELECT columna, funcion_agregacion(columna)
FROM tabla
WHERE condicion
GROUP BY columna
HAVING condicion_funcion_agregacion;
```

### Ejemplo:

```
SELECT COUNT(idConsumidor), Pais
FROM Consumidores
GROUP BY Pais
HAVING COUNT(idConsumidor) > 5;
```

## 5.6. HERRAMIENTAS DE LA BDD PARA OPTIMIZAR CONSULTAS

Cuando vayamos a desarrollar una BD es fundamental saber qué finalidad va a tener, organizar toda la información de la que dispongamos, definir las relaciones entre las diferentes tablas y elementos que sean necesarias y, finalmente, normalizarla. De esta forma, conseguiremos que la información no esté duplicada y podremos desarrollar una metodología eficiente de la información para un futuro.

Cuando ya la tengamos diseñada, la optimizaremos siguiendo una serie de recomendaciones:

- Eliminar las tablas que hemos creados previamente y que no utilizamos.
- Optimizar los índices de las tablas para asegurarnos de su correcto funcionamiento.
- No dejar consultas abiertas que puedan ralentizarnos el trabajo.
- No almacenar imágenes en la BD, es mucho mejor referenciar la ruta.
- Nombres simples en claves y campos de las tablas para facilitar el trabajo.





# **6** ESTRATEGIAS PARA EL CONTROL DE LAS TRANSACCIONES Y DE LA CONCURRENCIA

## 6.1. CONCEPTO DE INTEGRIDAD

La **integridad** es un factor fundamental en las BBDD. Se encarga de que los datos que la componen sean lo más correctos posible. Estos datos almacenados deben cumplir una serie de restricciones con el fin de facilitar al usuario el trabajo de manipulación de los datos.

En el caso de las claves primarias y ajena, la integridad puede verse afectada si, por ejemplo, borramos un registro de nuestra tabla principal que está relacionado con uno o varios registros de otras tablas secundarias que tengan una clave ajena a la tabla principal, se provocará un error al detectar un fallo de integridad.

Un ejemplo podría ser en el caso de tener las tablas:

```
PAÍSES (id, nombre, extensión)
MUNICIPIOS (id, nombre, población, país)
```

Siendo el atributo *PAÍS* de la tabla *MUNICIPIOS* una clave ajena al id de la tabla *PAÍSES* que nos indica el país al que pertenece ese municipio.

Supongamos que tenemos los siguientes datos:

**Tabla PAÍSES**

ID	NOMBRE	EXTENSIÓN
33	Francia	506000
34	España	644000

**Tabla MUNICIPIOS**

ID	NOMBRE	POBLACIÓN	PAÍS
120	Montpellier	285000	33
130	Pontevedra	83000	34

Si de la tabla *PAÍSES* eliminamos el registro de *Francia*, en la tabla *municipios* nos quedará un registro, el de *Montpellier*, con la clave ajena apuntando a un país que ya no existe en la tabla *PAÍSES*. Esto podría provocar un fallo de integridad.

Sin embargo, según como pretendamos diseñar una BD, podrían existir claves secundarias que refieran a una clave principal que ya no está. Así, con el lenguaje DDL podemos aplicar una serie de opciones que se añaden detrás de la cláusula **REFERENCES** como:

- **ON DELETE SET NULL:** si en la tabla principal se borra un registro, se asignarán valores nulos a aquellas claves ajenas de otra tabla que estuvieran referenciando al registro de la tabla principal que se ha borrado.
- **ON UPDATE SET NULL:** si en la tabla principal se modifica la clave primaria de un registro, se asignarán valores nulos a aquellas claves ajenas de otra tabla que estuvieran referenciando el registro de la tabla principal que se ha modificado.
- **ON DELETE CASCADE:** si eliminamos un registro en la tabla principal, también eliminaremos los registros de otras tablas que tuviesen una clave ajena referenciando al mismo registro de la principal que se ha eliminado.
- **ON UPDATE CASCADE:** si modificamos un registro en la tabla principal también modificaremos todos los registros de otras tablas que tuviesen una clave ajena referenciando al mismo registro de la principal que se ha modificado.
- **ON DELETE SET DEFAULT:** si eliminamos un registro en la tabla principal se asignará un valor por defecto en aquellas claves ajenas de otras tablas que estuvieran referenciando el registro de la tabla principal que se acaba de borrar.
- **ON UPDATE SET DEFAULT:** si modificamos un registro en la tabla principal se asignará un valor por defecto en aquellas claves ajenas de otras tablas que estuvieran referenciando el registro de la tabla principal que se acaba de borrar.
- **ON DELETE NO ACTION:** no permite la eliminación de la clave primaria si esta tuviera claves ajenas referenciándola con esta restricción.
- **ON UPDATE NO ACTION:** no permite la modificación de la clave primaria si esta tuviera claves ajenas referenciándola con esta restricción.

Nota: En MySQL, *NO ACTION* y *RESTRICT* son equivalentes.

Existen, además, reglas de integridad que debemos controlar si nos encontramos con violaciones de la integridad. Estas reglas se dividen en dos principales:

- **Reglas de integridad de dominios:** si se le asigna un valor a un atributo sin saber la relación que este tiene con los demás que forman la BD.
- **Reglas de integridad de relaciones:** cuando se admite una tupla dada para ser insertada o bien cuando se van a relacionar varias tuplas.



### ponte a prueba

**Dadas las siguientes opciones, ¿cuál de las siguientes sentencias se añadirá detrás de la cláusula REFERENCES para evitar que al eliminarse un registro de la tabla principal nos pueda dar problemas?**

- a) ON DELETE SET DEFAULT
- b) ON UPDATE CASCADE
- c) ON UPDATE SET NULL
- d) ON DELETE ACTION NULL

## 6.2. CONCEPTO DE TRANSACCIÓN. CONTROL

Cuando hablamos de una **transacción** nos referimos a un conjunto de diferentes acciones capaces de realizar transformaciones sobre los estados de un sistema conservando su integridad. Una transacción puede ser cualquier tipo de operación atómica que se realice con éxito.

Las acciones que se realizan en una transacción son independientes entre ellas, aunque puedan estar relacionadas. El primer paso de una transacción es la apertura, después la ejecución de las acciones de forma correcta y, por último, la confirmación y cierre de la transacción.

Sin embargo, si se observa cualquier tipo de error en ellas, la transacción se deshace. De esta forma se tiene siempre en cuenta la integridad de los datos.

Un buen ejemplo sería una transferencia de dinero de una cuenta a otra:

Supongamos que en la BD de un banco se desea realizar una transferencia de 1.000 euros de la cuenta de *Alejandro* a la cuenta de *Isabel*. Para realizarla deberíamos seguir estos pasos:

1. Comprobamos si en la cuenta de Alejandro hay 1.000 euros o más.
2. En caso de que sí los hubiera, restamos 1.000 euros al saldo de Alejandro.
3. Sumamos 1.000 euros de saldo en la cuenta de Isabel.



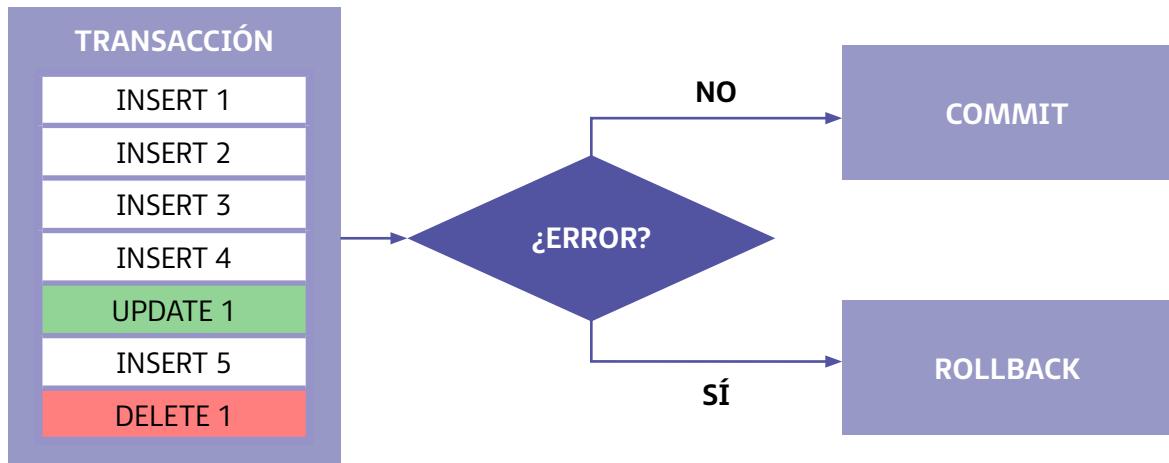
Si entre el paso 2 y el paso 3 hubiese algún tipo de error en el sistema y nunca se llegase a ejecutar el paso 3, es decir, que nunca se añadieran 1.000 euros a la cuenta de Isabel, entonces se deberían revertir los pasos anteriores dados, es decir, se deberían devolver los 1.000 euros al saldo de Alejandro.

## 6.3. PROPIEDADES DE LAS TRANSACCIONES: ATOMICIDAD, CONSISTENCIA, AISLAMIENTO Y PERMANENCIA

Existen una serie de propiedades de las transacciones que debemos conocer:

- **Atomicidad** (*Atomicity*): actúan de manera atómica, es decir, o todas las sentencias de la transacción (modificación, agregación o borrado) se ejecutan con éxito, o no lo hará ninguna.
- **Consistencia** (*Consistency*): cuando se ejecuta la transacción, el sistema debe pasar de un estado consistente a otro que también lo sea, pese a los cambios que se han realizado.
- **Aislamiento** (*Isolation*): cada transacción debe ejecutarse de forma que no afecte al resto de transacciones. De este modo, nos aseguraremos de que si dos transacciones modifican o pretenden modificar el mismo dato, primero se ejecute íntegramente una y después la otra.
- **Permanencia** (*Durability*): todos los cambios que se hayan producido cuando se realiza una transacción no se pierden, sino que permanecen.

## 6.4. ESTADOS DE UNA TRANSACCIÓN: ACTIVA, PARCIALMENTE COMPROMETIDA, FALLIDA, ABORTADA Y COMPROMETIDA

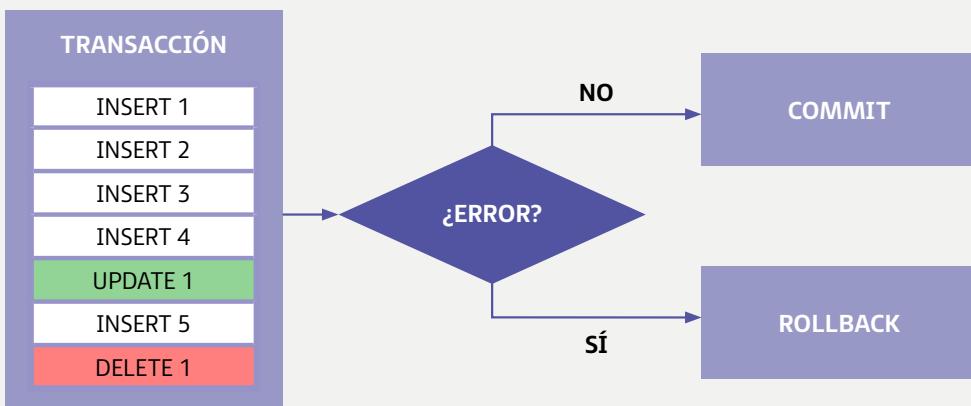


En el dibujo podemos apreciar muy bien su funcionamiento. Se realizan una serie de instrucciones en la que la última es una operación de borrado (**DELETE**). Si todo se ha realizado sin problema: COMMIT (validar) y finaliza la transacción.

Pero si se detecta algún error: ROLLBACK (cancelar) que elimina los cambios anteriores y vuelve a empezar.



**ponte a prueba**



Si visualizamos la imagen anterior, es posible observar el estado en una transacción. En caso de detectar algún error, ¿qué instrucción realizará?

- a) Rollback
- b) Commit
- c) Delete
- d) Update

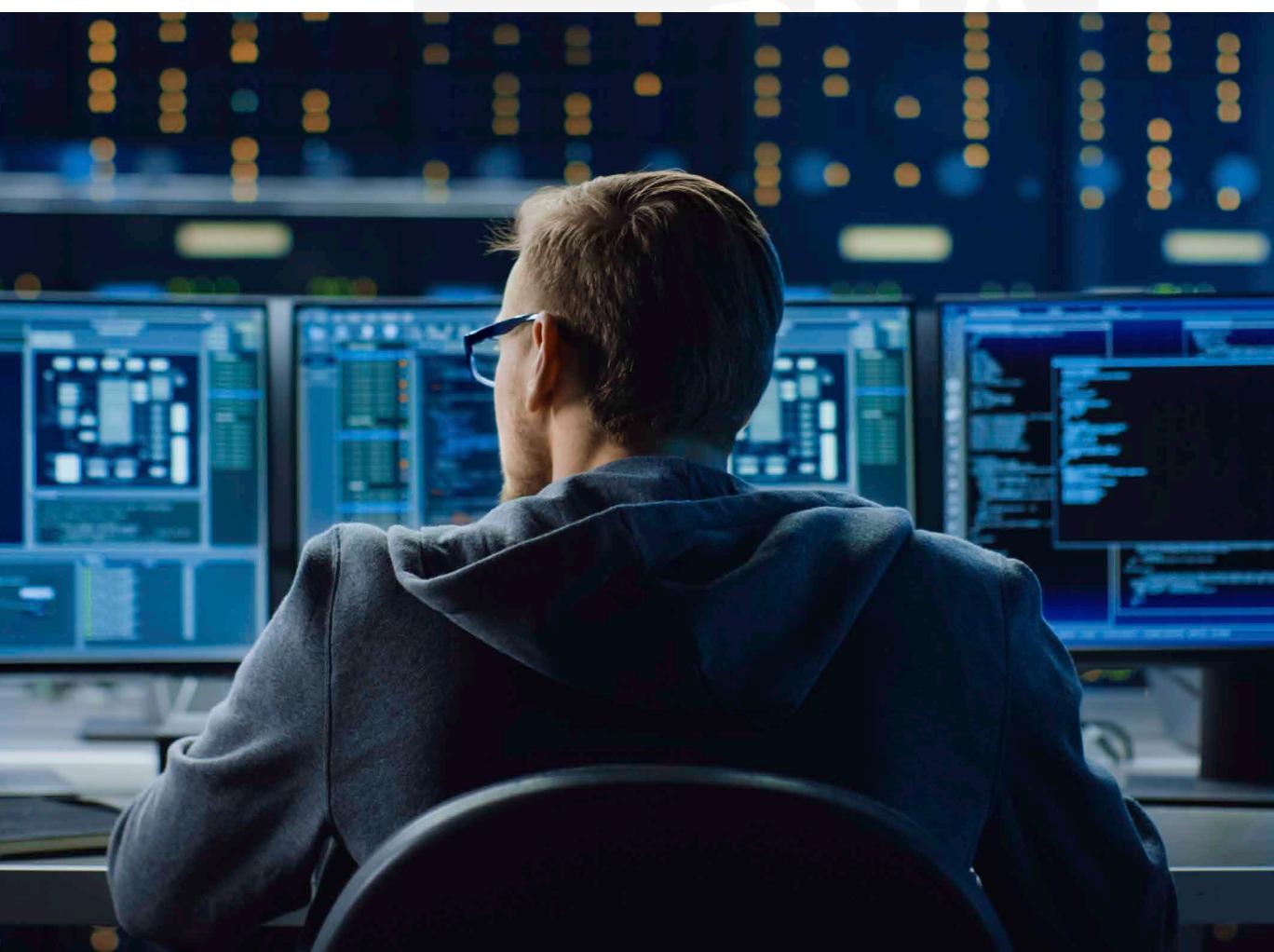
## 6.5. SENTENCIAS DE TRANSACCIONES

Debemos saber que una transacción comienza con el comando BEGIN o START TRANSACTION y puede acabar, como vimos anteriormente, en COMMIT o en ROLLBACK, para confirmar la transacción realizada, en el caso de COMMIT, o para deshacer los cambios realizados desde el comienzo de la transacción, en el caso de ROLLBACK.

### Ejemplo en MySQL

Supongamos que partimos de estos datos de la tabla MUNICIPIOS:

ID	NOMBRE	POBLACIÓN	PAÍS
120	Montpellier	285000	33
130	Pontevedra	83000	34



Y supongamos que deseamos intercambiar los nombres de *Pontevedra* y *Montpellier* con una transacción. El código MySQL a emplear sería el siguiente:

```
START TRANSACTION;
UPDATE municipios SET nombre = "Montpellier" WHERE id=130;
UPDATE municipios SET nombre = "Pontevedra" WHERE id=120;
COMMIT;
```

De modo que la tabla municipios pasa a tener estos datos:

ID	NOMBRE	POBLACIÓN	PAÍS
120	Pontevedra	285000	33
130	Montpellier	83000	34

Si en vez de finalizar la transacción con COMMIT lo hubiéramos hecho con ROLLBACK, los cambios en la tabla no se habrían producido.

## 6.6. PROBLEMAS DERIVADOS DE LA EJECUCIÓN CONCURRENTE DE TRANSACCIONES

Anteriormente, hemos visto el conjunto de normas que deben cumplir las transacciones. Además, también es importante que nos detengamos un poco ante los problemas más frecuentes que pueden ocasionar.

Uno de los problemas principales es el que ocurre cuando dos transacciones intentan acceder al mismo dato de manera simultánea. A esto le llamamos problema de concurrencia. Podemos diferenciar entre **tres tipos diferentes de problemas de concurrencia**:

- **Dirty Read** (lectura sucia): cuando una transacción consulta datos escritos de otra que aún no ha sido confirmada.
- **Non Repeatable Read** (lectura irrepetible): cuando una transacción vuelve a hacer una lectura de unos datos que ya había leído y comprueba entonces que han sido modificados en alguna transacción.
- **Phantom Read** (lectura fantasma): cuando una transacción realiza una consulta y encuentra datos que antes eran inexistentes. Alguna transacción los ha insertado.

## 6.7. CONTROL DE CONCURRENCIA: TÉCNICAS OPTIMISTAS Y PESIMISTAS

Uno de los principales problemas que se ocasionan en las transacciones de los datos de una BD es que se solicite el acceso a un mismo dato desde dos lugares diferentes. En esa situación se necesita un **control de concurrencia** para dar una solución.

El encargado de este control de concurrencia es el **planificador**. Este va a realizar diferentes esquemas para que las transacciones no se solapen entre ellas.

Siempre es conveniente que el planificador no realice ningún cambio en el sistema, tanto si las transacciones se ejecutan de forma concurrente, como si lo hacen una detrás de otra.

Veamos qué tipo de técnicas podemos utilizar para evitar este tipo de problemas:

- **Técnicas pesimistas**

- **Técnicas de bloqueo (*locks*)**

Su tarea principal es poder bloquear aquellos datos para que no se acceda a ellos desde diferentes transacciones (sincronizar el acceso).

Para ello va a hacer uso del cerrojo (**lock**) que actuará como una variable para poder controlar el estado de los datos según las operaciones permitidas. Esto conlleva un riesgo importante como es el bloqueo (**deadlock**).

Aunque, es cierto que estos cerrojos no nos garantizan la serialización por sí solos y necesitan el uso de un protocolo para que podamos tener un control del posicionamiento de aquellas operaciones que se hayan realizado dentro de una transacción.

Uno de los más utilizados es el protocolo de bloqueo en dos fases. En la primera fase (o de crecimiento) es donde van a solicitarse los locks mientras que, en la segunda (denominada devolución), va a ser donde se realizan los *unlocks*.

Este protocolo sí garantiza la serialización, aunque no libera ninguno cerrojo desde que comienza hasta que finaliza.

- **Técnicas de marcas de tiempo (*time-stamping*)**

Las marcas de tiempo se utilizan para que exista un único identificador para cada transacción. Estas marcas deben ir en orden para poder el acceso a los diferentes datos sin que estos se solapen.

- **Técnicas optimistas**

También conocidas como técnicas de validación o de certificación. Estas técnicas no llevan impuestas ninguna restricción específica ni ningún bloqueo. Aunque, al final, hacen una comprobación de tres fases diferentes que se pueden dar: lectura, validación y escritura.

Son bastante adecuadas cuando existen pocas transacciones, así hay menos operaciones.

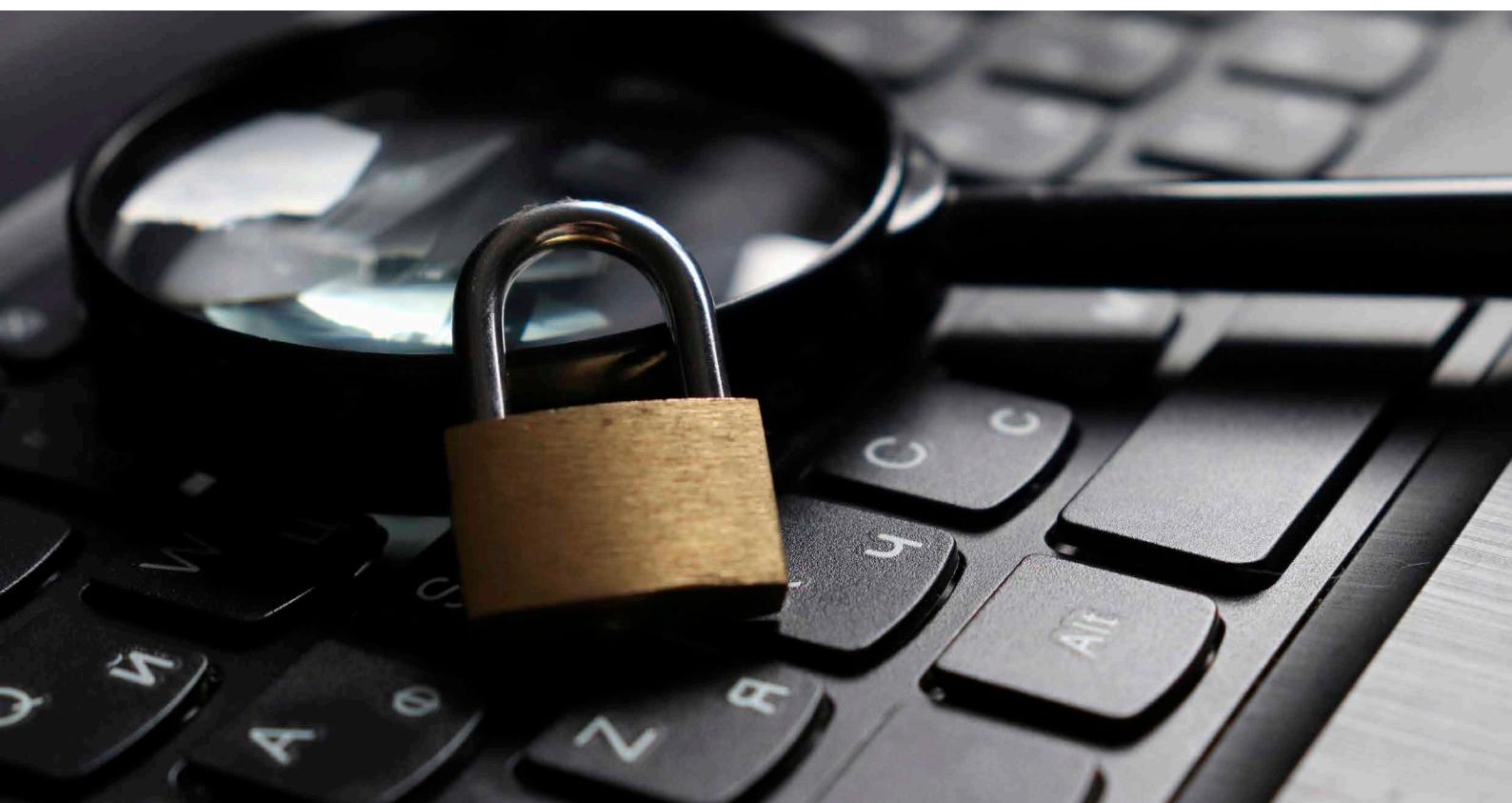
Debemos apuntar que, en la fase de validación, se va a comprobar que esta no se solapa con ninguna otra transacción que esté confirmada o en fase de validación.



**ponte a prueba**

**Para poder bloquear los datos y evitar acceder a cualquiera de ellos en una transacción. ¿Cuál de las siguientes opciones utilizaremos en forma de variable para controlar el estado de los datos?**

- a) Los *locks*.
- b) Las lecturas.
- c) Las transacciones.
- d) El control.



## 6.8. RECUPERACIÓN ANTE ERRORES. MECANISMOS PARA DESHACER TRANSACCIONES

Existen diferentes tipos de fallos que nos podemos encontrar:

1. Fallo informático.
2. Error del sistema o transacción: división por cero, errores de algún parámetro, de programación, etc.
3. Errores locales: datos que no se encuentran en una transacción, saldo insuficiente, etc.
4. Control de concurrencia: transacciones que no siguen adelante para garantizar la serializabilidad.
5. Fallo del disco (lectura/escritura).
6. Problemas físicos.

De hecho, podemos agrupar estos fallos dentro de dos grupos:

- Fallos que pierden contenido de la memoria estable como, por ejemplo, los discos.
- Fallos que pierden contenido de la memoria volátil (memoria principal).

El **mecanismo de recuperación** que tomemos ante cualquier error va a ser el encargado de **restablecer la BD** al estado anterior al fallo. Este mecanismo, además, debe reducir el tiempo de uso de la BD después de que haya habido un error.

Existen diferentes tipos de **estrategias de recuperación** que podemos llevar a cabo:

- Cuando **se daña una parte de la BD**: se puede restaurar una copia previa al fallo y reconstruir un nuevo estado volviendo a realizar las operaciones que ya están almacenadas.
- Cuando la BD **no parece dañada, pero no responde**: se pueden realizar las operaciones que nos han llevado a esta situación siguiendo un orden inverso. De esta forma comprobaremos qué acción ha producido el fallo.

**Nota:** una transacción es serializable si el resultado final de la transacción es equivalente a sus comandos ejecutados secuencialmente sin que se superpongan entre ellos.





# 7

## LENGUAJES DE LAS BBDD PARA LA CREACIÓN DE SU ESTRUCTURA

## 7.1. VISTAS Y OTRAS EXTENSIONES DEL LENGUAJE

Una vista es una consulta preestablecida sobre una o varias tablas de una BD. Las vistas no forman parte del esquema físico de la BD, sino que son tablas virtuales. Si una vista ya creada está basada en unas tablas y la información de dichas tablas es modificada, la próxima vez que se invoque a esa vista, la información también estará actualizada. Esto es debido a que las vistas no almacenan la información de las tablas, sino que se guarda la estructura de la propia vista, que es una consulta.

### Creación

Sintaxis para crear una vista:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW nombre_vista [lista_columnas]
AS sentencia_select
[WITH CHECK OPTION [CONSTRAINT restricción]]
[WITH READ ONLY [CONSTRAINT restricción]]
```

- **OR REPLACE:** lo utilizamos si la vista ya existe, la cambia por la actual.
- **FORCE:** aunque no se disponga de los datos necesarios para realizar la consulta, crea la vista.
- **Lista\_columnas:** listado de las columnas que devuelve la consulta.
- **WITH CHECK OPTION:** ofrece la posibilidad de añadir (INSERT) o modificar (UPDATE) las filas a visualizar.
- **WITH READ ONLY:** vista de solo lectura con posibilidad de asignarle un nombre.

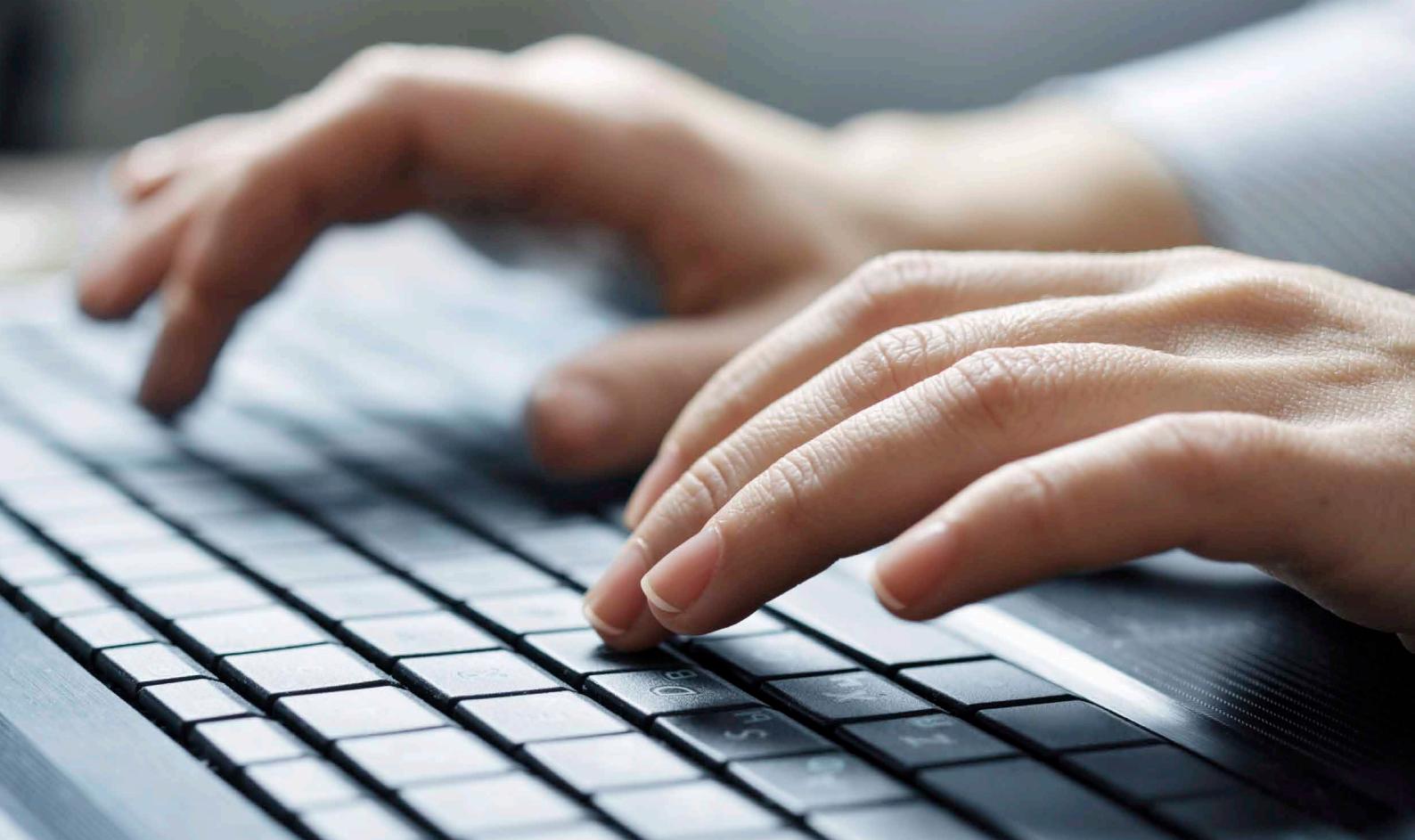
Una vez que ya se han creado las vistas, se pueden utilizar de la misma forma que si fueran tablas.

### Veamos un ejemplo de creación de vista

Creamos una vista con el nombre y apellidos de los empleados que trabajen en el departamento de ropa de unos grandes almacenes:

```
CREATE OR REPLACE VIEW vista_ejemplo_ropa
AS SELECT nombre, apellidos
FROM empleados
WHERE depot = ropa;
```

Cuando definimos una vista es habitual que usemos la sintaxis CREATE OR REPLACE. Por lo tanto, se puede utilizar para crear o modificar una vista ya definida previamente.



### Tipos de vistas

En función del número de tablas que conforman la vista podemos diferenciar entre dos tipos de vistas:

- **Simples**: son aquellas que están formadas por una única tabla y no tienen ninguna función de agrupación. Permiten operaciones DML.
- **Complejas**: son aquellas que están formadas por más de una tabla y sí utilizan funciones de agrupación. No permiten operaciones DML.

### Ejemplo en MySQL de una vista simple

Si partimos de la tabla MUNICIPIOS:

ID	NOMBRE	POBLACIÓN	PAÍS
120	Pontevedra	285000	34
130	Montpellier	83000	33
140	Lyon	513000	33
150	Valencia	800000	34

Podríamos crear una vista simple que nos mostrase solo los *Municipios de Francia* y solo los campos de *Nombre* y *Población*:

```
CREATE VIEW municipios_franceses
AS SELECT nombre, poblacion
FROM municipios
WHERE pais = 33;
```

De modo que para invocar la vista podríamos usar la siguiente sentencia:

```
SELECT * FROM municipios_franceses;
```

Lo que nos devolvería la siguiente información:

NOMBRE	POBLACIÓN
Montpellier	83000
Lyon	513000

En función de las operaciones de las vistas, podemos diferenciar entre dos tipos de vistas:

- **Solo lectura:** en este tipo de vista no se puede modificar ningún tipo de información.
- **Actualizables:** en este tipo de vista se pueden realizar las acciones de inserción, modificación o eliminación de los datos contenidos en las tablas que usa la propia vista. Para este tipo de vistas que insertan, modifican y eliminan datos se necesita tener los permisos necesarios.

## Propiedades de las vistas

Las vistas tienen las siguientes características:

- Las vistas pueden simplificar la información que hay en una o varias tablas, dado que muestra un subconjunto de información.
- Las vistas pueden añadir información que no existe en las tablas, como medias aritméticas o sumatorios.
- Ocupan muy poco espacio, dado que no se almacena la información resultante de la vista, sino que se almacena la estructura de la vista.

## Borrado de una vista

Sintaxis para borrar una tabla:

```
DROP VIEW [IF EXISTS] nombre_vista;
```

### Ejemplo en MySQL:

Para eliminar la anterior vista que hemos creado de *municipios\_franceses* utilizaríamos la siguiente sentencia:

```
DROP VIEW municipios_franceses;
```

## Tipos de restricciones

### Restricciones de clave principal

La clave principal de una tabla, en la mayoría de los casos, suele ser una columna o una combinación de columnas que van a tener una serie de valores únicos y de esta manera, ofrecerán integridad.

La restricción de clave principal, cuando interviene más de una columna, puede que se encuentre con estos valores duplicados en una misma columna, así que, para cada combinación de valores, debemos definir esta restricción de clave principal.

La tabla de nuestro ejemplo debe cumplir:

- La restricción de clave principal solo va incluida en una tabla.
- La clave principal debe ser menor a dieciséis columnas, que es la longitud máxima.
- El índice que genera la restricción de clave principal debe encontrarse entre 1 y 999.
- Las columnas generadas con la restricción de clave principal deben ser asignadas con valores nulos.

### Restricciones de clave externa

Hacen referencia a una o varias columnas que van a ser utilizadas para crear diferentes vínculos entre las distintas tablas. La intención es que estos puedan ser almacenados en una tabla de clave externa. Podemos decir que para que se cree vínculo entre dos tablas, una columna de una de las tablas debe hacer referencia a otra columna que actúa como clave principal de la otra.

Deben cumplir:

- Solo admite operaciones de DELETE cuando tiene más de 253 referencias de clave.
- Cuando una tabla se referencia a sí misma sigue teniendo 253 referencias de clave.
- No hay más de 253 referencias disponibles.



### **ponte a prueba**

**A la hora de crear una vista, ¿cuál de las siguientes opciones añadiremos para reemplazar una vista ya existente?**

- a) OR REPLACE
- b) FORCE
- c) OR FORCE
- d) REPLACE FORCE

**Dada la siguiente tabla, ¿cuál de las siguientes sentencias es correcta para crear una vista?**

ID	Marca	Modelo	Año
31351684	Ford	Focus	2019
31236312	VW	Golf	2007
36515135	Opel	Astra	2012

- a) CREATE VIEW coche\_ford AS SELECT \* FROM Coches WHERE Marca = "Ford";
- b) UPDATE VIEW coche\_ford AS SELECT \* FROM Coches WHERE Marca = "Ford";
- c) CREATE VIEW coche\_ford AS SELECT \* FROM Coches WHERE Modelo = "Ford";
- d) CREATE VIEW coche\_ford AS SELECT \* FROM Coches;

**Para eliminar una vista, a través del SGBD MySQL, ¿qué sentencia utilizaremos?**

- a) DROP VIEW nombre\_vista;
- b) REPLACE VIEW nombre\_vista;
- c) DELETE nombre\_vista;
- d) DROP OR REPLACE nombre\_vista

## BIBLIOGRAFÍA / WEBGRAFÍA

---

- ❑ Muñoz, F.J., Benítez, J. I., Lozano, A. (2005). *Sistemas Operativos en entornos Monouuario y Multiusuario*. Aravaca, Madrid.
- ❑ Ramos, M. J., Ramos, A., Rubio, S. (2005). *Instalación y mantenimiento de equipos informáticos*. Aravaca, Madrid.
- ❑ <https://sites.google.com/site/carlospesrivas/recursos/informatica-ciclos-formativos-de-grado-superior>

## solucionario

### 1.1. Evolución histórica de las BBDD

¿A partir de qué año es posible gestionar los sistemas de gestión de bases de datos a nivel de usuario?

- b) A partir de los años 80, con la aparición del lenguaje SQL.

¿Qué sistemas aparecen en los años 60?

- b) Los sistemas Batchprocessing.

### 1.3. Almacenamiento de la información

¿A qué tipo de fichero se hace referencia con esta afirmación: "Disponemos de un fichero lleno de registros, y para encontrar un registro en concreto debemos recorrer todos los registros de forma secuencia que se encuentran antes que este."?

- c) Ficheros planos.

Dadas las siguientes opciones, indica cuál no podría ser un campo clave en una base de datos.

- a) Nombre de persona.

En referencia a la clasificación de las BBDD. Según su ubicación, ¿en cuántos tipos las podemos clasificar?

- d) Todas las opciones anteriores son correctas.

### 1.7. Bases de datos centralizadas y distribuidas

Indica cuál de las siguientes opciones es una ventaja de trabajar con bases de datos distribuidas.

- a) Supone un bajo coste a la hora de crear una red de computadoras pequeña.

Indica cuál de las siguientes opciones pertenece a una base de datos centralizada.

- b) No tiene demasiados elementos de procesamiento.

El diseñador es el encargado de distribuir los datos en una base de datos. ¿Cuál de las siguientes opciones se corresponde con un esquema costoso, en el que cada uno de los nodos tendrá la información duplicada, que también dispone de mucha disponibilidad pero que resulta más lento al tener muchos datos?

- b) Distribución replicada.

Indica cuál de las siguientes opciones es una de las ventajas principales de las bases de datos centralizadas.

- c) Rendimiento óptimo al procesar datos.

### 2.2. Entidad: representación gráfica, atributos y tipos de claves

¿Cuál es el orden correcto de los nombres de los elementos de la siguiente imagen (de arriba abajo)?

- c) Atributo, Atributo multivalorado, relación, entidad, entidad débil.

¿Cuál de las dos formas es correcta para representar un diagrama E-R?

- c) Ambas opciones son correctas.

### 2.5. Modelo entidad-relación

En el modelo E-R extendido, indica a qué restricción semántica pertenece el siguiente ejemplo.

- a) Inclusiva.

Indica el orden correcto de cada diagrama en función del orden alfabético [A, B, C, D]

- b) Inclusiva parcial, inclusiva total, exclusiva parcial, exclusiva total.

Dada la siguiente jerarquía, indica cuál de los **subtipos** pueden pertenecer a esta jerarquía exclusiva.

- a) Hombre, Mujer.

### 3.1 Terminología del modelo relacional

Mostrando el siguiente ejemplo de una tabla de vehículos, ¿a qué término del modelo relacional pertenecen los nombres Marca, Modelo, Año y Precio?

- a) Atributo.

### 3.4. Concepto y tipos de clave: candidatas, primarias, ajenas, alternativas

¿Cómo se conoce al conjunto de atributos de una relación que son clave primaria de otra relación distinta y que por causas del diseño deben estar relacionadas?

- b) Clave foránea.



## solucionario

Dada la siguiente tabla, ¿cuál de los siguientes campos es la clave primaria?

a) DNI.

Dada la siguiente tabla, ¿cuál de los siguientes campos optan para ser clave candidata?

a) Código cliente y DNI.

Dadas las siguientes tablas, ¿cuál de los siguientes atributos son clave foránea?

a) Código cliente de la tabla Pedido.

### 3.7. Traducción del modelo entidad-relación al modelo relacional

¿En cuál de los siguientes casos se pueden presentar a la hora de realizar la transformación de entidad relación a modelo relacional?

d) Todas las opciones son correctas.

Indica la participación que falta en este ejemplo.

b) (1,n).

Indica la participación que falta en este ejemplo.

b) (n,m).

### 4.3. Primera forma normal (1FN)

Para que una tabla se encuentre en primera forma normal, los atributos deben contener valores atómicos, es decir, que no se puedan dividir. Dada la siguiente tabla, ¿cuál de los siguientes atributos no cumple con esta primera forma normal?

d) Emails.

### 4.5. Tercera forma normal (3FN)

Para que una tabla esté en tercera forma normal (3FN), es necesario que esté en 2FN y no disponga de dependencias funcionales en los atributos no clave, pero no es necesario que esté en 1FN.

b) Falso.

### 5. Herramientas gráficas proporcionadas por el SGBD para la edición de la BD

¿Cuál de las siguientes opciones no es una norma básica para tener en cuenta cuando diseñamos instrucciones SQL?

c) No es necesario delimitar el fin de un comando.

### 5.4 Lenguaje de manipulación de datos (DML)

A través del SGBD MySQL queremos crear una tabla con el nombre “profesor”, que tenga los atributos de identificador, edad y asignatura. ¿Cómo lo haremos?

b) CREATE TABLE profesor (identificador INT PRIMARY KEY, edad INT, asignatura VARCHAR (15)) ENGINE=INNODB;

A través del SGBD MySQL, ¿qué sentencia usaremos si queremos borrar la tabla profesor?

a) DROP TABLE profesor;

Dada la siguiente tabla, ¿cuál de las siguientes sentencias utilizaremos para obtener el número de productos que tenemos en la tabla “Productos”?

a) SELECT COUNT(IDProducto) AS NúmeroProductos FROM Productos;

Dada la siguiente tabla, ¿cuál de las siguientes sentencias utilizaremos para obtener la cantidad total de Stock que disponemos en la tabla “Productos”?

a) SELECT SUM(Stock) AS NúmeroProductos FROM Productos;

### 6.1 Concepto de integridad

Dadas las siguientes opciones, ¿cuál de las siguientes sentencias se añadirá detrás de la cláusula REFERENCES para evitar que al eliminarse un registro de la tabla principal nos pueda dar problemas?

a) ON DELETE SET DEFAULT.

### 6.4 Estados de una transacción

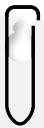
Si visualizamos la imagen anterior, es posible observar el estado en una transacción. En caso de detectar algún error, ¿qué instrucción realizará?

a) Rollback.

### 6.7 Control de concurrencia

Para poder bloquear los datos y evitar acceder a cualquiera de ellos en una transacción. ¿Cuál de las siguientes opciones utilizaremos en forma de variable para controlar el estado de los datos.

a) Los locks.



## solucionario

### 7.1 Vistas y otras extensiones del lenguaje

A la hora de crear una vista, ¿cuál de las siguientes opciones añadiremos para reemplazar una vista ya existente?

- a) OR REPLACE.

Dada la siguiente tabla, ¿cuál de las siguientes sentencias es correcta para crear una vista?

- a) CREATE VIEW coche\_ford AS SELECT \* FROM Coches WHERE Marca = "Ford";

Para eliminar una vista, a través del SGBD MySQL, ¿qué sentencia utilizaremos?

- a) DROP VIEW nombre\_vista;