CERTIFICADO DE PROFESIONALIDAD

PROGRAMACIÓN DE BASES DE DATOS RELACIONALES

MF0226_3

FRANCISCO JAVIER MARTÍNEZ LÓPEZ
AMALIA GALLEGOS RUIZ

Ra-Ma®
www.ra-ma.es/cp

PROGRAMACIÓN DE BASES DE DATOS RELACIONALES

PROGRAMACIÓN DE BASES DE DATOS RELACIONALES

FRANCISCO JAVIER MARTÍNEZ LÓPEZ
AMALIA GALLEGOS RUIZ

La ley prohíbe
fotocopiar este libro

Editado por:
RA-MA Editorial
Calle Jarama, 3A, Polígono Industrial Igarsa
28860 PARACUELLOS DE JARAMA, Madrid
Teléfono: 91 658 42 80
Fax: 91 662 81 39
Correo electrónico: editorial@ra-ma.com
Internet: **www.ra-ma.es** y **www.ra-ma.com**
ISBN: 978-84-9964-696-1
Depósito Legal: M-520-2017
Maquetación: Antonio García Tomé
Diseño de Portada: Antonio García Tomé
Filmación e Impresión: Copias Centro
Impreso en España en febrero de 2017

A nuestros padres, por su eterna entrega
y dedicación incondicional.

Índice

Introducción

Los **certificados de profesionalidad** son titulaciones oficiales válidas en todo el territorio nacional que acreditan la capacitación para el desarrollo de una actividad laboral. Para su obtención es necesario superar todos los módulos formativos que integran dichos certificados.

La presente obra se ha tratado de ajustar en lo posible a los contenidos oficiales del módulo formativo de 210 horas de duración llamado "**Programación de bases de datos relacionales" (MF0226_3)**, incluido en los certificados de profesionalidad "**Programación con lenguajes orientados a objetos y bases de datos relacionales**" (IFC0112) y "**Programación en lenguajes estructurados de aplicaciones de gestión**" (IFCD0111), ambos de nivel 3, el nivel más alto que se otorga a una cualificación profesional.

Dicho módulo trata de ofrecer una visión general de las bases de datos relacionales, incluyendo los principios generales de los sistemas gestores de bases de datos, metodología de análisis y diseño para implementarlas, así como los principales lenguajes de programación para implementarlas y explotarlas.

Aunque durante el libro se quiere dar una visión general de este tipo de bases de datos y de sus elementos, también se describen DBMS concretos y se realizan ejercicios guiados con sistemas concretos y con herramientas muy diversas, con el objetivo de llevar a la práctica lo estudiado a lo largo de los capítulos que integran el trabajo.

1 Introducción a las bases de datos

1.1 EVOLUCIÓN HISTÓRICA DE LAS BASES DE DATOS

Podemos remontar los orígenes de las bases de datos a la antigüedad, donde ya existían bibliotecas con ciertos sistemas de catalogación de obras y relacionadas con otros ámbitos prácticos como la recogida y archivo de información sobre cosechas o censos. Sin embargo, el gran problema de esos sistemas de almacenamiento de datos primitivos residía en que las búsquedas eran lentas y poco eficaces, ya que todos los procesos de búsqueda, consulta, actualización, etc., eran manuales.

De este modo, el término de bases de datos siempre ha estado relacionado con el de informática, a partir de la necesidad de almacenar grandes cantidades de datos, para su posterior consulta. Así, diferentes autores sitúan la aparición del término de bases de datos, como hoy lo conocemos, a partir de las necesidades producidas por las nuevas industrias de la revolución Industrial.

NOTA

Herman Hollerith (Buffalo, 29/02/1860 - 17/11/1929) está considerado como el primer ingeniero estadístico de la Historia, es decir, el primero que logró el tratamiento automático de información.

Hollerith observó que las preguntas contenidas en los censos se podían contestar con un "sí" o un "no". Entonces ideó una tarjeta perforada, que consistía en una cartulina en la que, según estuviera perforada o no en determinadas posiciones, se contestaban este tipo de preguntas.

Podemos obtener más información de este estadounidense en Wikipedia, entre otras fuentes: *https://es.wikipedia.org/wiki/Herman_Hollerith*

Década de 1950

En esta década se inventan las cintas magnéticas, las cuales se emplearon para suplir las necesidades de información de las nuevas industrias como, por ejemplo, para automatizar la información de las nóminas (como el aumento de salario). Consistía en leer una cinta o más de una y pasar sus datos a otra (también se podía pasar información desde tarjetas perforadas). Este sistema también aportaba la posibilidad de disponer de un *backup*[1]. A través de este mecanismo se comenzó a automatizar la información, con el problema de que solo se podía leer de forma secuencial y ordenadamente.

1 El *backup* consiste en tener una copia de seguridad o de respaldo de la información importante.

Figura 1.1. *Lectores de cinta de IBM System*

Primera generación de las bases de datos: década de 1960

En esta época coincidió, por un lado, la masiva adquisición de ordenadores por parte de compañías privadas debido a una bajada de sus precios, y, por otro, el uso de los discos, aspecto este que supuso un adelanto muy importante, ya que por medio de este soporte se podía consulta información directamente (en milisegundos), sin necesidad de conocer su ubicación exacta. Esto se debía a que, a diferencia de las cintas magnéticas, ya no era necesaria la secuencialidad.

En esta década también se inició el trabajo con las primeras generaciones de bases de datos en red y las bases de datos jerárquicas, ya que surgen estructuras de datos como listas y árboles para trabajar con la información.

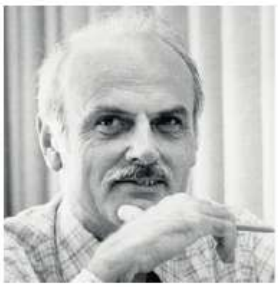
En 1967 se crea la organización Data Base Task Group (DBTG), dando origen a la Conference on Data Systems Languages (CODASYL), con el objetivo de regular el desarrollo de un lenguaje de programación estándar que pudiera ser utilizado en multitud de ordenadores, fruto del cual nació el lenguaje COBOL, aunque nunca llegaron a establecer estándar alguno, que llegaría años después con ANSI.

Segunda generación de las bases de datos: década de 1970

La segunda generación de las bases de datos surgió de la mano de Edgar Frank Codd (23/08/1923 – 18/04/2003), ingeniero de la empresa IBM, con su artículo "Un modelo relacional de datos para grandes bancos de datos compartidos"[2], en 1970, en el cual definió el modelo relacional y las famosas 12 reglas de Codd para los sistemas de datos relacionales. Por tal motivo, en la literatura, Codd está considerado como el padre de las bases de datos relacionales. De este modo, también surgirían las bases de datos comerciales.

A partir de los trabajos de Codd se desarrollaron los primeros Sistemas de Gestión de Bases de Datos Relacionales (DBMS relacionales), destacando el proyecto denominado System R, de IBM, el cual mejoró notablemente el rendimiento del modelo relacional, haciéndolo más competitivo que los anteriores modelos jerárquicos y de red. Otro DBMS relacional importante es el de Oracle Corporation (empresa que toma como nombre el de dicho *software*, Oracle).

INFORMACIÓN



Os recomendamos, para ampliar los conocimientos sobre las 12 reglas de Codd, que las consultes por Internet, por ejemplo, en el siguiente enlace de la enciclopedia Wikipedia:

https://es.wikipedia.org/wiki/12_reglas_de_Codd

O, incluso, si tienes un nivel aceptable de inglés, leas su artículo original, en el siguiente enlace:

https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf

Tercera generación de las bases de datos: décadas de 1980 y 1990

Años más tarde, el lenguaje de bases de datos que se utilizó para el proyecto System R, denominado SEQUEL[3], evolucionó al hoy extendido lenguaje SQL (*Structure Query Language*), el cual fue adoptado por una gran cantidad de compañías, como Oracle o Sybase, para implementar sus productos, convirtiéndose, de este modo, en un estándar industrial en lo que respecta a las bases de datos relacionales. De hecho, en 1986 el ANSI adoptó SQL como estándar para los lenguajes relaciones y, en 1987, se transformó en estándar ISO.

Por su parte, en la década de 1990, la investigación en bases de datos evolucionó hacia las bases de datos orientadas a objetos, las cuales pueden trabajar con relativa facilidad con datos complejos. De esta forma, surgen herramientas tan cotidianas en la actualidad como Microsoft Excel y Microsoft Access, como claro exponente de este tipo de bases de datos orientadas a objetos.

2 El título original es "A Relational Model of Data for Large Shared Data Banks" y fue publicado por la revista *Communications of the ACM* (*http://cacm.acm.org/*).

3 SEQUEL es el acrónimo de *Structure English Query Language.*

Posteriormente, estos dos últimos modelos de bases de datos han evolucionado hacia bases de datos objeto-relacionales, que es una extensión de las bases de datos relacionales tradicionales, a las que se les proporcionan características de la programación orientada a objetos.

Presente y futuro de las bases de datos

A finales del siglo XX se caracterizó por la aparición de la www (World Wide Web) y ello afectó a las bases de datos, ya que por este medio se facilitaba la consulta de las bases de datos. Actualmente, Internet nos ofrece una gran capacidad de almacenamiento de información "en la nube", sin interrupciones, con rapidez (debido a la trepidante evolución de los servicios de Internet), entre otras características interesantes, que hacen de la *Cloud Computing* el futuro de la programación y, por ende, de las bases de datos.

INFORMACIÓN

Podemos obtener más información sobre la historia de las bases de datos y, en general, sobre los orígenes de la Informática en el **Museo de Informática de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia**, que visita virtualmente en el siguiente enlace: *http://histinf.blogs.upv.es/*

1.2 VENTAJAS E INCONVENIENTES DE LAS BASES DE DATOS

Pese a que hemos estado comprobando que la evolución clara del manejo de información está relacionada con el desarrollo de los Sistemas de Gestión de Bases de Datos (DBMS), en esta apartado repasaremos cuáles son las principales ventajas e inconvenientes del uso de bases de datos.

Comenzando por las ventajas, podemos destacar las siguientes:

- **Independencia física:** la forma de almacenar los datos (servidores locales, servidores "en la nube", ordenador local, etc.) no influye en su manipulación lógica.
- **Independencia lógica:** las aplicaciones que utilizan la base de datos no deben ser modificadas porque se modifiquen elementos de la base de datos.
- **Flexibilidad y seguridad:** las bases de datos ofrecen distintas vistas, en función de los usuarios y aplicaciones que trabajen sobre ellas, por lo que, a su vez, también restringen el acceso a los elementos que no se deban mostrar.

  Del mismo modo, las bases de datos permiten el **acceso simultáneo a los datos**, facilitando el control de acceso concurrente.
- **Uniformidad:** las estructuras lógicas siempre tienen una única forma conceptual (las tablas).
- **Menor redundancia y mayor integridad de datos:** las bases de datos reducen la repetición de datos y generan mayor dificultad de perder información. Todo ello **reduce la probabilidad de incoherencias** con los datos.

  Otras ventajas relacionadas con esto es la **reducción del espacio de almacenamiento** y el **acceso más eficiente** a la información.
- **Sencillez.**

Del mismo modo, entre las desventajas podemos destacar las siguientes:

- **Necesidad de personal cualificado:** para aprovechar las ventajas anteriormente comentadas y en función de las necesidades de la base de datos, del *software* que funciona con ella y, en definitiva, de las necesidades de la empresa que la explota (número de perfiles de usuarios distintos, necesidades de mantenimiento, *backups*, velocidad de respuesta, etc.), podría ser necesaria la especialización de personal o la formación de programadores y analistas sobre las posibilidades y las limitaciones de las bases de datos.
- **Requerimientos adicionales *hardware*:** generalmente la implantación de un DBMS recomienda la adquisición de equipos *hardware* adicionales, tales como servidores locales, ampliación de memoria, discos duros, servidores en la nube, etc.

  Por este motivo, antes de implantar un DBMS es preciso analizar si realmente es necesario. En ocasiones, si se tienen pocos datos, son usados por un único usuario simultáneamente y no hay que realizar consultas complejas sobre dichos datos, quizás sea mejor solución usar una hoja de cálculo.
- **Falta de rentabilidad a corto plazo:** debido a los altos costes *software*, *hardware* y de personal en el momento de la implantación de la base de datos.
- **Ausencia de estándares reales,** que se traduce en excesiva dependencia de los sistemas comerciales del mercado (este problema cada vez es menor).

1.3 CONCEPTOS GENERALES

Para familiarizarnos con el lenguaje básico de las bases de datos, incluiremos algunas definiciones interesantes que emplearemos a lo largo de todo este libro.

1.3.1 CONCEPTO DE BASES DE DATOS

Una base de datos es un sistema formado por una colección de datos almacenados, relativos a diversas temáticas y categorizados de distinta manera, pero que comparten entre sí algún tipo de vínculo o contexto que permiten el acceso directo a ellos, así como su relación. Estos datos se encuentran almacenados sobre un soporte físico.

Según Llanos Ferraris, *"La aparición del concepto de bases de datos se produce a comienzos de la década de 1960, concretamente en un simposio que se celebró en Santa Mónica (California, EE.UU.)* (…).

Una base de datos es un conjunto, colección o depósito de datos almacenados en un soporte informático de acceso directo. Los datos deben estar relacionados y estructurados de acuerdo con un modelo capaz de recoger el contenido semántico de los datos almacenados. Dada la importancia que tienen en el mundo real las relaciones entre los datos, es imprescindible que la base de datos sea capaz de almacenar estas interrelaciones. Esta es una de las principales diferencias respecto a los ficheros tradicionales, en los que no se almacenan dichas relaciones. Además, las bases de datos modernas también almacenan las restricciones semánticas que están presentes en los datos y a las que se les está concediendo una importancia creciente".

La creación de una base de datos es un proceso complejo, que parte de la necesidad de almacenar información del mundo real para que su acceso sea rápido y eficiente. Un buen diseño de nuestra base de datos nos va a dar un mayor rendimiento, mejor velocidad de acceso a los datos, eliminación de las redundancias y, en general, un mejor aprovechamiento de los recursos de los que se disponen.

Por el contrario, un diseño inapropiado de la base de datos puede dar lugar a los siguientes problemas:

- Redundancias
- Incoherencias
- Pérdidas de dependencias funcionales

1.3.2 OBJETIVOS DE LOS SISTEMAS DE BASES DE DATOS

El **sistema gestor de bases de datos** es un conjunto de programas de propósito general que facilita la definición, construcción y manipulación de las bases de datos.

El principal objetivo de un sistema de bases de datos consistirá en reducir la probabilidad de que la base de datos incorpore los problemas que a continuación describiremos.

1.3.2.1 Redundancia e inconsistencia de datos

La **redundancia** no es más que repetición innecesaria de la información, que no solo incrementará el espacio de almacenamiento dedicado a la base de datos, sino que en el futuro provocará inconsistencia de dicha información.

Por su parte, la **inconsistencia** se produce al haber elementos duplicados. En esa situación, la base de datos será inconsistente o incoherente consigo misma, cuando tenga dos elementos que deberían tener los mismos valores, pero tienen valores distintos.

Por ejemplo, si en una base de datos tenemos información de personas en almacenes diferentes, puede que mi número de teléfono sea diferente en un almacén (ya que mantiene el antiguo, al no haber sido actualizado con mi nuevo número) y en otro tenga el actual.

1.3.2.2 Dificultad para tener acceso a los datos

Un buen DBMS es el que permite a todos los usuarios del sistema, en virtud de sus privilegios, el acceso a los datos, consultas y operaciones que le correspondan.

Esto se materializa en la asignación de las correspondientes claves de acceso a dichos usuarios y asignación de roles (con sus correspondientes privilegios sobre la base de datos).

Además de permitir el acceso a la base de datos, en función del privilegio de cada tipo de usuario, se deberá poder realizar consultas sencillas sobre la base de datos con lenguajes específicos, como SQL, programar aplicaciones que funcionen sobre la base de datos o poder simplemente operar con esas aplicaciones (caso de los usuarios más inexpertos).

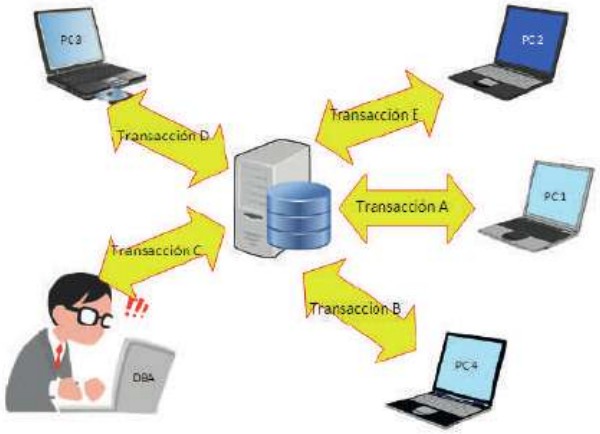
1.3.2.3 Aislamiento de los datos

Relacionado con la independencia lógica que deben mantener las bases de datos, las aplicaciones que utilizan la base de datos no deben ser modificadas porque se modifiquen elementos de la base de datos.

De este modo, el objetivo de un DBMS es que no sea necesaria la reprogramación de una aplicación ante cambios en los elementos de la base de datos, ni que, al contrario, se tengan que reestructurar las bases de datos ante cambios en las aplicaciones que trabajan con ellas.

1.3.2.4 Anomalías del acceso concurrente

Dado que muchas bases de datos deben permitir que múltiples usuarios actualicen y consulten los datos simultáneamente, en un entorno de tanta concurrencia en el acceso a la información se podrían dar resultados inconsistentes. Así, uno de los objetivos del DBMS consistirá en el control y supervisión de las transacciones las bases de datos.



1.3.2.5 Problemas de seguridad

La **seguridad** dentro de una base de datos consiste en controlar que los usuarios estén autorizados para llevar a cabo las operaciones que realizan en todo momento.

Un usuario de una base de datos tiene una serie de permisos sobre los objetos que componen dicha base de datos. Estos permisos los crea el DBA (administrador de la base de datos) y dependerán del uso que los usuarios puedan hacer de la misma. Además, cada usuario tendrá solo acceso a los objetos o parte de los objetos que se crean necesarios. Algunos usuarios dispondrán, a su vez, de la posibilidad de transferir sus permisos a otros usuarios de la base de datos.

El administrador cuenta con dos herramientas principalmente para mantener la seguridad en la base de datos:

- **Las vistas:** se trata de una "tabla virtual", ya que esta en realidad no existe, sino que está generada con datos de otras tablas.
- **Los permisos:** cualquier usuario que quiera realizar una operación sobre un objeto de la base de datos debe tener permiso para ejecutarlo.

1.3.2.6 Problemas de integridad

El término **integridad** de datos se refiere a la corrección de los mismos en una base de datos, así como que estos estén bien relacionados cuando se ubican en entidades distintas a lo largo de la misma.

Cuando se aplican ciertas operaciones a la base de datos, tales como *Insert, Delete* o *Update*, la integridad de los datos almacenados puede corromperse. Algunas consecuencias de estas operaciones pueden ser la que vemos en el siguiente ejemplo.

EJEMPLO 1.1

Tomaremos, para ejemplificar estos problemas típicos de integridad, el supuesto de una base de datos que almacene la gestión de cursos, alumnos y profesores en un centro de formación:

- **Se pueden añadir datos no válidos:** tener un curso de formación en el que aparece como alumno una persona que no existe.
- **Se pueden modificar datos existentes tomando un valor incorrecto:** se asigna un profesor a un curso que no existe.
- **Los cambios en la base de datos pueden perderse, debido a un error.**
- **Los cambios en la base de datos pueden ser aplicados parcialmente:** añadir un curso sin especificar la modalidad del mismo (presencial, a distancia, etc.).
- Etcétera.

De hecho, una de las funciones más importantes del DBMS es mantener la integridad de los datos en la base de datos.

NOTA

La enciclopedia Wikipedia, en su artículo "Integridad de datos", *https://es.wikipedia.org/wiki/Integridad_de_datos* define los principales "Tipos de restricciones de integridad:

- **Datos Requeridos:** establece que una columna tenga un valor no *NULL*. Se define efectuando la declaración de una columna es *NOT NULL* cuando la tabla que contiene las columnas se crea por primera vez (…).
- **Chequeo de Validez:** cuando se crea una tabla, cada columna tiene un tipo de datos y el DBMS asegura que solamente los datos del tipo especificado sean ingresados en la tabla.
- **Integridad de entidad:** establece que la clave primaria de una tabla debe tener un valor único para cada fila de la tabla (…).
- **Integridad referencial:** asegura la integridad entre las claves foráneas y primarias (relaciones padre/hijo). Existen cuatro actualizaciones de la base de datos que pueden corromper la integridad referencial:
  - La inserción de una fila hijo se produce cuando no coincide la clave foránea con la clave primaria del padre.
  - La actualización en la clave foránea de la fila hijo, donde se produce una actualización en la clave ajena de la fila hijo con una sentencia *UPDATE* y la misma no coincide con ninguna clave primaria.
  - La supresión de una fila padre, con la que, si una fila padre (que tiene uno o más hijos) se suprime, las filas hijos quedarán huérfanas.
  - La actualización de la clave primaria de un fila padre, donde si en un fila padre, que tiene uno o más hijos se actualiza su clave primaria, las filas hijos quedarán huérfanas".

WIKIPEDIA
The Free Encyclopedia

1.3.3 ADMINISTRACIÓN DE LOS DATOS Y ADMINISTRACIÓN DE BASES DE DATOS

El **administrador de datos (DA)** es la persona que tendrá la responsabilidad central sobre los datos. Dado que los datos son uno de los activos más valiosos de la empresa, organización o grupo, es imperativo que exista una persona que los entienda, junto con las necesidades de la citada empresa, organización o grupo con respecto a ellos, a un nivel de administración superior. Por lo tanto, es labor del administrador decidir, en primer lugar, qué datos deben ser almacenados en las bases de datos y establecer políticas para mantener y manejar esos datos una vez almacenados.

El **administrador de base de datos (DBA)** es el técnico responsable de implementar las decisiones del administrador de datos, esto es, se responsabilizará de la instalación, configuración y administración del DBMS relacional. Por lo tanto, debe ser un profesional en Tecnologías de la Información. El trabajo del DBA consiste en crear la base de datos real e implementar los controles técnicos necesarios para hacer cumplir las diversas decisiones de las políticas hechas por el DA.

El DBA también es responsable de asegurar que el sistema opere con el rendimiento adecuado y de proporcionar una variedad de otros servicios técnicos.

RECUERDA

En la sección 1.3.10 ahondaremos en las funciones del DBA.

1.3.4 NIVELES DE ARQUITECTURA: INTERNO, CONCEPTUAL Y EXTERNO

Comenzaremos desarrollando este apartado explicando que existen tres características propias de los DBMS: la separación entre los programas de aplicación y los datos, la necesidad de emplear un diccionario de datos para almacenar el esquema de la base de datos y el uso de múltiples vistas por parte de los usuarios.

Para poder cumplir estas tres importantes características, el comité ANSI-SPARC propuso, en 1975, una arquitectura de tres niveles para los DBMS, dividiéndola en los niveles de abstracción interno, conceptual y externo.

- **Nivel interno o físico:** su responsabilidad reside en el almacenamiento físico de los datos (el tamaño de los bloques de datos, los métodos de direccionamiento, los índices, etc.), así como los métodos de acceso. Es el considerado como más exhaustivo.
- **Nivel conceptual o lógico:** se encarga de la descripción de la estructura de los datos y de sus relaciones (describe las entidades, atributos, relaciones, restricciones, etc.).
- **Nivel externo:** define las vistas, es decir, las partes de la base de datos visibles para las distintas aplicaciones y usuarios que trabajan sobre ella.

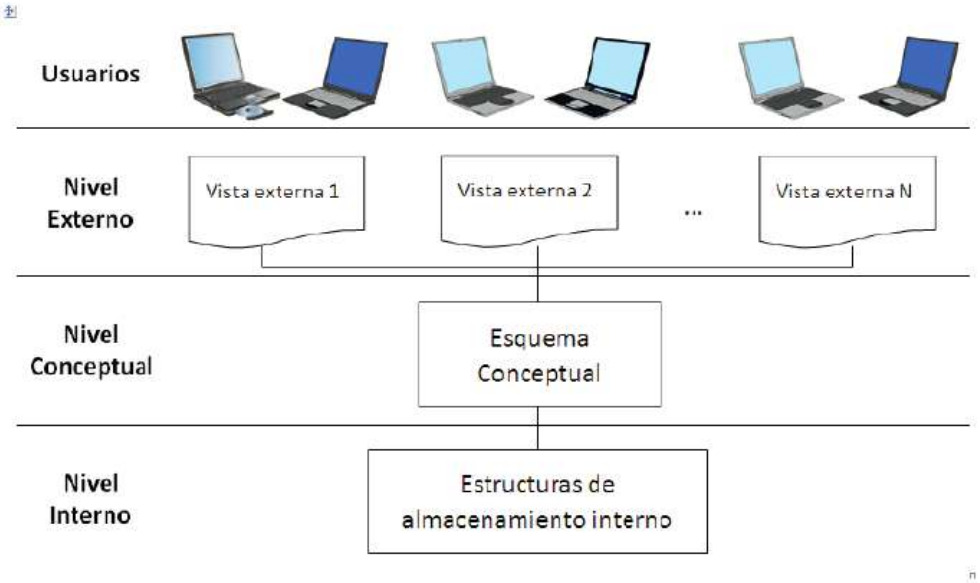


Figura 1.2. *Esquema de la arquitectura de un DBMS*

NOTA

La denominación de comité ANSI-SPARC es acrónimo de American National Standard Institute - Standards Planning and Requirements Committee.
Las arquitecturas de bases de datos han evolucionado desde 1975, aunque la considerada estándar en la actualidad data de finales de la década de 1970. Se trata de la descrita por el comité ANSI/X3/SPARC *(*Standard Planning and Requirements Committee of the American National Standards Institute on Computers and Information Processing*).*

1.3.5 MODELOS DE DATOS. CLASIFICACIÓN

Podemos definir el **modelo de datos** como una colección de herramientas conceptuales, reglas y convenciones utilizadas para describir los datos, sus relaciones, sus dominios y las restricciones de integridad. Los modelos de datos proporcionan la abstracción necesaria a la base de datos para describir:

- Datos
- Relaciones entre datos
- Significado de los datos
- Restricciones de consistencia

Podemos clasificar los modelos de datos en tres tipos:

Modelos lógicos basados en objetos

Estos modelos se utilizan para describir datos en los niveles externo y conceptual, es decir, con este modelo representamos los datos tal y como los captamos en el mundo real. Poseen una capacidad de definición bastante flexible y permiten especificar restricciones de integridad de los datos. Pese a que el modelo Entidad-Relación será el que más desarrollemos en este libro y, en definitiva, el más extendido, otros también importantes son:

- **Modelo de Entidad-Relación:** se trata del modelo de datos más extendido en el mundo. Presenta un alto nivel de abstracción y es un modelo basado en percibir la realidad como una serie de entidades (objetos que existen en la realidad) y de relaciones entre esos objetos. Tanto las entidades como las relaciones contienen atributos (información que los definen), que nos servirán para diferenciar cada entidad de otras similares.

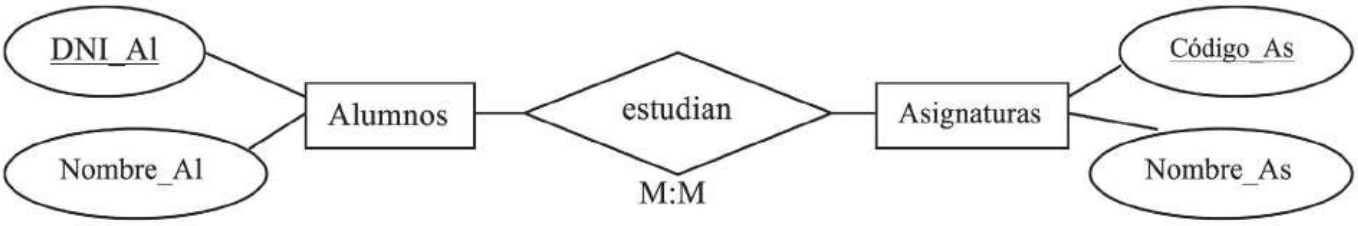


Figura 1.3. *Ejemplo de modelo de Entidad-Relación (nótese que en el presente esquema se ha llevado a cabo una simplificación tanto en el número de atributos, que en sucesivas figuras se ampliarán, como en la notación de la cardinalidad, que veremos en capítulos siguientes)*

- **Modelo Orientado a Objetos:** este modelo también se basa en una colección de objetos, con valores almacenados (como variables de instancia) dentro de ellos. La principal diferencia radica en que estos objetos también almacenan fragmentos de código, con las operaciones fundamentales de estos, llamados *métodos*.

  La forma en la que un objeto puede acceder a los datos de otro es mediante la invocación de los métodos del segundo, acción se conoce como *paso de mensajes*.

  Al contrario que las entidades del modelo E-R, cada objeto dispone de identidad propia, aunque pueda poseer la misma información que otro objeto de los de su clase. Dicha distinción se establece a nivel físico (mediante la asignación de una posición en memoria diferente para almacenar el citado objeto).

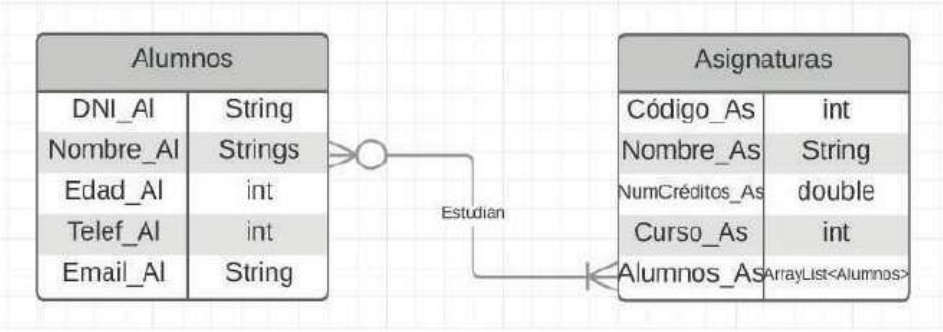


Figura 1.4. *Ejemplo de un modelado OO equivalente al modelo E-R de la Figura 1.3*

Del mismo modo, existen otros modelos, tales como:

- Modelo de datos semántico
- Modelo de datos funcional

Modelos lógicos basados en registros

Es un modelo de menor abstracción, que se emplea para describir los datos en los niveles conceptual y físico. A diferencia de los modelos anteriores, la base de datos se estructura en registros de formato fijo, diferentes, en función del tipo de elemento a almacenar. De esta forma, para cada tipo de registro se define un registro de longitud fija y de número de campos también fijo, lo que simplifica la implementación de la base de datos.

Al igual que ocurría en el anterior modelo, nos encontramos con diversos modelos, fundamentalmente diferenciados en las estructuras de datos para almacenar estos registros y para relacionarlos.

- **Modelo relacional:** en este modelo se emplea una colección de tablas para expresar tanto los datos, como sus relaciones. Cada tabla contiene varias columnas (con nombre único) y las relaciones se representan mediante columnas comunes en dichas tablas.

Tabla "Alumnos"

DNI_Al	Nombre_Al	Edad_Al	Teléfono_Al	Email_Al
11111111A	Francisco Javier	12	666111111	fran@gmail.com
22222222B	Amalia	10	666222222	amalia@gmail.com
33333333C	Manuel	15	666333333	manuel@gmail.com
44444444D	Mª del Mar	12	666444444	mmar@gmail.com

Tabla "Asignaturas"

Código_As	Nombre_As	NumCréditos_As	Curso_As
001	Bases de Datos	6	3
002	Redes	9	3
003	Programación	6	1
004	Sistemas Operativos	4,5	2

Tabla "Estudian"

DNI_Al	Código_As
11111111A	003
22222222B	004
33333333C	001
33333333C	002
44444444D	003

Figura 1.5. *Ejemplo de un modelado relacional equivalente al modelo E-R de la Figura 1.3*

- **Modelo de red:** los datos del modelo de red se representan mediante colecciones de registros, pero sus relaciones se representan mediante *enlaces* o *punteros* a otros registros. El esquema final del modelado se asemeja a un grafo dirigido.

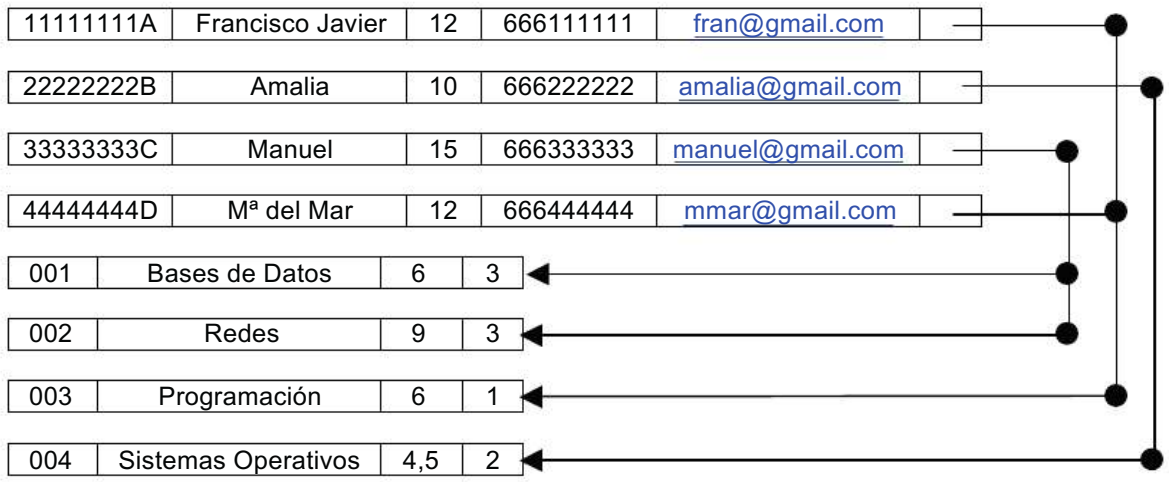


Figura 1.6. *Ejemplo de un modelado de red equivalente al modelo E-R de la Figura 1.3*

- **Modelo jerárquico:** este modelo es muy similar al modelo de red, con la diferencia de que los registros se organizan como colecciones de árboles, en lugar de como grafos dirigidos.

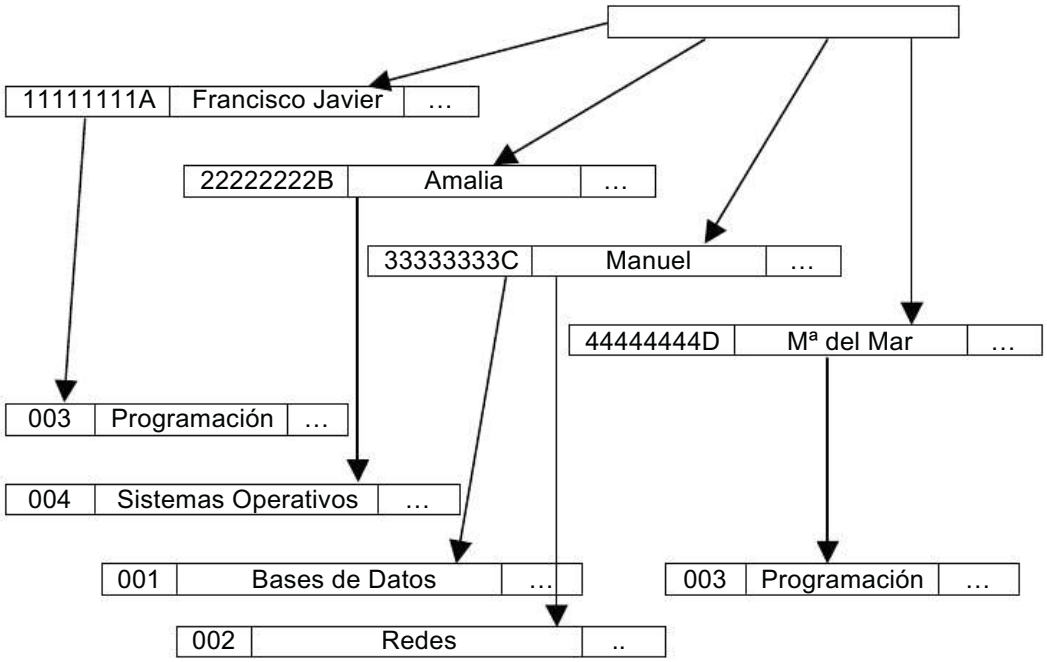


Figura 1.7. *Ejemplo de un modelado jerárquico equivalente al modelo E-R de la Figura 1.3*

Modelos físicos

Se trata de estructuras de datos de bajo nivel implementadas dentro del propio manejador. A diferencia del modelo de datos lógico, hay pocos modelos de datos físicos que se usan en la práctica. Ejemplos de estas estructuras son los Árboles B+ o las Tablas Hash.

1.3.6 INDEPENDENCIA DE LOS DATOS

Podemos utilizar la arquitectura de niveles, estudiada en el apartado 1.3.4, para explicar este concepto de independencia de los datos. Así, la **independencia de los datos** se puede definir como la capacidad para modificar el esquema de un nivel de la arquitectura, sin estar obligados a modificar el esquema del nivel inmediatamente superior. De este modo, se establecen dos tipos de independencia de datos:

- **Independencia lógica:** mediante la cual podemos modificar el esquema conceptual, sin tener que cambiar las vistas externas ni los programas de aplicación.
- **Independencia física:** mediante la cual podemos modificar las estructuras de almacenamiento interno, sin tener que alterar el esquema conceptual.

No obstante, también tenemos que tener en cuenta que en los DBMS con arquitectura de varios niveles, como los que proponemos en este libro, es necesario ampliar el diccionario de datos, para que incluya la información sobre cómo establecer la correspondencia entre las peticiones de los usuarios finales y los datos, entre los diversos niveles de abstracción. Para ello, el sistema de bases de datos empleará una serie de procedimientos, haciendo uso de las especificaciones incluidas en dicho diccionario de datos. A su vez, esto supondrá un coste extraordinario durante la ejecución de las operaciones sobre la base de datos.

1.3.7 LENGUAJE DE DEFINICIÓN DE DATOS

Con el lenguaje de definición de datos (DDL, *Data Definition Language*), podemos informar al DBMS sobre la estructura y los componentes de la base de datos a nivel conceptual e interno (tablas, atributos, restricciones, etc.). Del mismo modo, también permite definir vistas e índices.

NOTA

En algunos DBMS hay una separación entre los niveles conceptual e interno también a nivel de lenguaje de definición. En estos casos el DDL solo se utiliza para definir el esquema conceptual, mientras que el esquema interno se define con un *lenguaje de definición de almacenamiento* (DSDL, *Data Storage Definition Language*).

1.3.8 LENGUAJE DE MANEJO DE BASES DE DATOS. TIPOS

El lenguaje de manejo de bases de datos, también llamado *lenguaje de manipulación de bases de datos* (DML, *Data Manipulation Language*), es el utilizado para realizar consultas y actualizaciones en la base de datos.

Estos lenguajes DML pueden ser, a su vez, clasificados en:

- **Procedimentales:** el usuario debe indicar las operaciones a realizar y la secuencia en la que se llevan a cabo. Se basan en el álgebra relacional. Un exponente de este tipo de lenguajes es SQL.
- **No procedimentales:** el usuario se limita a describir datos y efectuar consultas sin indicar el modo en que se realizarán. Un ejemplo de estos lenguajes es QUEL y QBE.

1.3.9 EL SISTEMA DE GESTIÓN DE LA BASE DE DATOS (DBMS). FUNCIONES

Dentro de las funciones que debe cumplir un DBMS, podemos destacar las siguientes:

- **Definición de datos:** los lenguajes de definición de datos (DDL) permiten especificar el esquema o diseño de una base de datos, mediante la definición de los objetos que la constituyen, la estructura y las interrelaciones.
- **Manipulación de datos:** consiste en realizar ciertas acciones sobre los datos existentes, o añadir nuevos. Estas acciones son:
  - Consultas
  - Insertar nuevos datos
  - Eliminar datos
  - Modificaciones o actualizaciones

  Para poder realizar estas tareas, se utilizan lenguajes de manipulación de datos (DML).
- **Seguridad:** la información contenida en la base de datos solo debe ser utilizada por determinados usuarios (a todos o parte de los datos). Para aumentar la privacidad se establecen y gestionan contraseñas.
- **Recuperación:** se debe prever que se produzcan fallos, debiendo existir mecanismos para recuperar los datos ante errores. Dichos fallos pueden ser:

- En la transacción o del *software*
- En el sistema o de *hardware*
- Del medio de almacenamiento

- **Integridad:** los datos que relacionan una tabla con otra deben ser coherentes.
- **Concurrencia:** una base de datos debe asegurar la consistencia de los datos, que se puede ver alterada ante un sistema multiusuario, que permite la multiprogramación.
- **Diccionario de datos:** el diccionario de datos es una base de datos del sistema que contiene metadatos, o lo que es lo mismo, incluye datos referentes a los datos que constituyen la base de datos original. El DBMS es el responsable de su definición y mantenimiento.

1.3.10 EL ADMINISTRADOR DE LA BASE DE DATOS (DBA). FUNCIONES

El administrador de la base de datos es la persona encargada de gestionar los datos que almacenamos, asegurar su corrección y el buen funcionamiento de la misma. Es la persona encargada de velar por la seguridad, integridad y confidencialidad de la información.

Dependiendo del tamaño de la base de datos, este cargo puede ostentarlo una única persona o un grupo de personas. En bases de datos pequeñas, suele ser el administrador de la red la persona que asume estas funciones.

El grupo ANSI/X3/SPARC divide sus funciones en tres niveles:

- **Administrador de la empresa:** persona encargada del diseño lógico y conceptual de la base de datos.
- **Administrador de la base de datos:** es el encargado del diseño físico, mantenimiento, rendimiento y seguridad de la base de datos.
- **Administrador de aplicaciones:** creador de vistas, aplicaciones y programas necesarios para manejar la base de datos.

Podemos clasificar las principales tareas del DBA en las siguientes:

- **Definir el esquema conceptual de la base de datos:** una vez que el DA decide el contenido de la base de datos en un nivel abstracto, es el DBA el que debe diseñar el esquema conceptual correspondiente, definiendo las entidades a contemplar, los atributos de las mismas, así como las relaciones entre ellas.
- **Definir el esquema físico de la base de datos:** a partir del diseño conceptual, se llevará a cabo el diseño físico que consistirá en definir el DBMS concreto en el que integrar todos los elementos de la base de datos, junto a sus características y restricciones físicas necesarias para su correcto funcionamiento.
- **Definir los perfiles de usuarios:** identificar los privilegios de los usuarios y dar acceso a la base de datos según dichos privilegios.
- **Ser responsable de la integridad de los datos y la disponibilidad.**
- **Garantizar la seguridad de las bases de datos, realizar copias de seguridad y llevar a cabo la recuperación ante pérdidas de información.**
- **Diseñar planes de contingencia.**
- **Analizar y reportar datos corporativos que ayuden a la toma de decisiones en la inteligencia de negocios.**

Para cumplir con estas funciones, el DBA dispone de las siguientes herramientas:

- **Lenguaje de definición de datos:** es la herramienta más usada. El administrador podrá crear tablas, vistas, definir la organización física de datos, mejorar el rendimiento de la base de datos, definir las restricciones de integridad y la seguridad del sistema.
- **Utilidades del DBMS:** las cuales posibilitan acciones tales como crear copias de seguridad, restaurar los datos, recuperación frente a fallos, creación de usuarios, etc.
- **Simuladores:** que pueden generar estadísticas de uso de nuestra base de datos.
- **Herramientas CASE:** permiten llevar una metodología automática de ingeniería del *software*, para poder automatizar muchas tareas que antes se realizaban de forma manual.
- **Diccionario de datos**[4]**.**

1.3.11 USUARIOS DE LAS BASES DE DATOS

Los usuarios que pueden interaccionar con una base de datos se clasifican en cinco tipos en función de la interacción que lleven a cabo con esta:

- **Administrador (DBA):** es el usuario principal, ya que sobre él recae la tarea de diseñar y mantener la base de datos. Sus funciones han sido estudiadas en el apartado anterior.
- **Programadores de aplicaciones:** estos usuarios crean programas que acceden a la base de datos. Para ello, pueden utilizar lenguajes de manipulación de datos (DML) incrustado dentro de sus programas.
- **Usuarios sofisticados:** los usuarios sofisticados directamente escriben sus propias consultas, sin tener que usar aplicaciones para ello.
- **Usuarios especializados:** pertenecerían al grupo de los usuarios sofisticados, pero además crean aplicaciones especializadas (CAD, SIG).
- **Usuarios ingenuos:** usan aplicaciones desarrolladas por los programadores.

1.3.12 ESTRUCTURA GENERAL DE LA BASE DE DATOS. COMPONENTES FUNCIONALES

En la literatura existen diversas formas de realizar esta clasificación por lo que optamos por una de las más extendidas, la cual divide los componentes de la base de datos en:

Usuarios

Se trata de las personas que interactúan en función de sus privilegios, con la base de datos.

Pese a que ya lo hemos estudiado en el apartado 1.3.11, recordemos que, a su vez, los usuarios se pueden clasificar en:

4 De esta herramienta hablaremos más adelante.

- Administrador (DBA)
- Programadores de aplicaciones
- Usuarios sofisticados
- Usuarios especializados
- Usuarios ingenuos

Hardware

Engloba a todos los dispositivos que participan en el almacenamiento físico, así como aquellos que van a permitir la gestión de los mismos (disco duro, CPU, unidades de E/S, etc.).

Software

El *software* que permite manipular los datos sin necesidad de conocer cómo se han almacenado físicamente, como bien sabemos, se denomina DBMS. El acceso a la base de datos será dirigido siempre por el DBMS, estando integrado por una serie de rutinas que realizan un trabajado determinado (como ya hemos visto en anteriores apartados), como son:

- Integridad
- Seguridad de acceso
- Seguridad y recuperación de datos
- Acceso concurrente
- Interactuar con el gestor de archivos

Datos

Se refiere a la información que conforma la base de datos, es decir, tanto los datos como las relaciones entre ellos.

1.3.13 ARQUITECTURA DE SISTEMAS DE BASES DE DATOS

Como vimos en el apartado 1.3.4, se establecían tres niveles de arquitectura para estructurar los DBMS, siguiendo el estándar ANSI/SPARC. No obstante, para el desarrollo de este último apartado realizaremos una clasificación desde otra perspectiva, concretamente analizando aspectos como la conexión a red y la posibilidad de distribuir la carga operacional y la explotación de técnicas de paralelismo. De este modo, un DBMS puede clasificarse principalmente en cuatro subclases:

La arquitectura centralizada

Una base de datos centralizada es aquella que se encuentra almacenada en su totalidad en un solo lugar físico, es decir, en una sola máquina, en donde los usuarios trabajan en terminales que solo muestran resultados.

Los sistemas de bases de datos centralizadas son aquellos que se ejecutan en un único sistema informático sin interaccionar con ninguna otra computadora. Tales sistemas comprenden el rango desde los sistemas de bases de datos monousuarios (ejecutándose en computadoras personales), hasta los sistemas de bases de datos de alto rendimiento que se ubican en grandes sistemas.

Esta arquitectura se caracteriza por su sencillez y robustez, mas también por la posibilidad de provocar "cuellos de botella". Del mismo modo, cuando el sistema es multiusuario, es necesario implementar control de concurrencia para gestionar que múltiples usuarios accedan a la información simultáneamente, sin que haya pérdida de integridad.

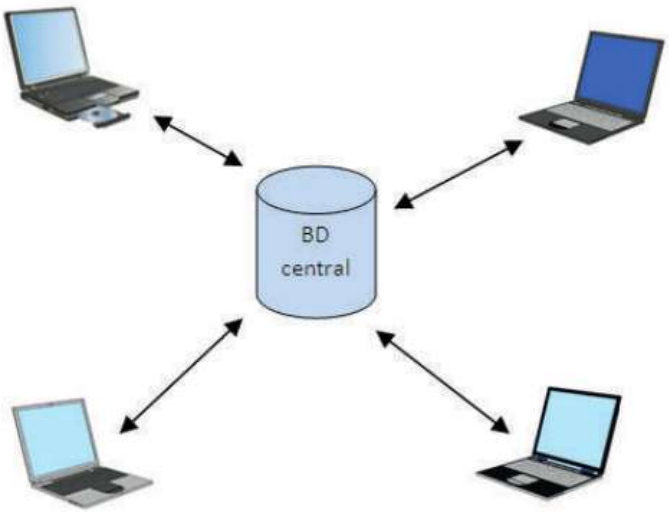


Figura 1.8. *Vista de un esquema de arquitectura centralizada*

La arquitectura cliente-servidor

Un sistema cliente-servidor es aquel en el que existen uno o varios clientes, así como uno o varios servidores (siempre y cuando los servidores no procesen la misma base de datos[5]), forman un sistema compuesto que permite realizar transacciones, análisis y presentación de los datos.

Los clientes, a través de la red, pueden realizar consultas al servidor, el cual posee el control de la información.

Esta arquitectura presenta las siguientes características:

- El servidor ofrece a sus clientes una única interfaz con la que ellos van a interactuar, por lo que los cambios a nivel de servidor afectarán poco o nada a los clientes.
- El cliente solo utiliza dicha interfaz externa para operar, sin necesidad de conocer la estructura interna, la ubicación física, el sistema operativo que emplea, etc. de la base de datos.

Dentro de la arquitectura cliente-servidor podemos apreciar dos tipos de arquitecturas:

- **Arquitectura de dos capas.** En la que existen dos capas:
  - Cliente. Es el que solicita los servicios al servidor.
  - Servidor. Es el proveedor de la información y los servicios.

  Y, en dichas dos capas, están incluidos los tres componentes distribuidos:
  - Interfaz de usuario
  - Gestión del procesamiento
  - Gestión de la base de datos

5 En el caso de que existieran varios servidores operando en la misma base de datos ya no sería una arquitectura cliente-servidor sino una arquitectura distribuida

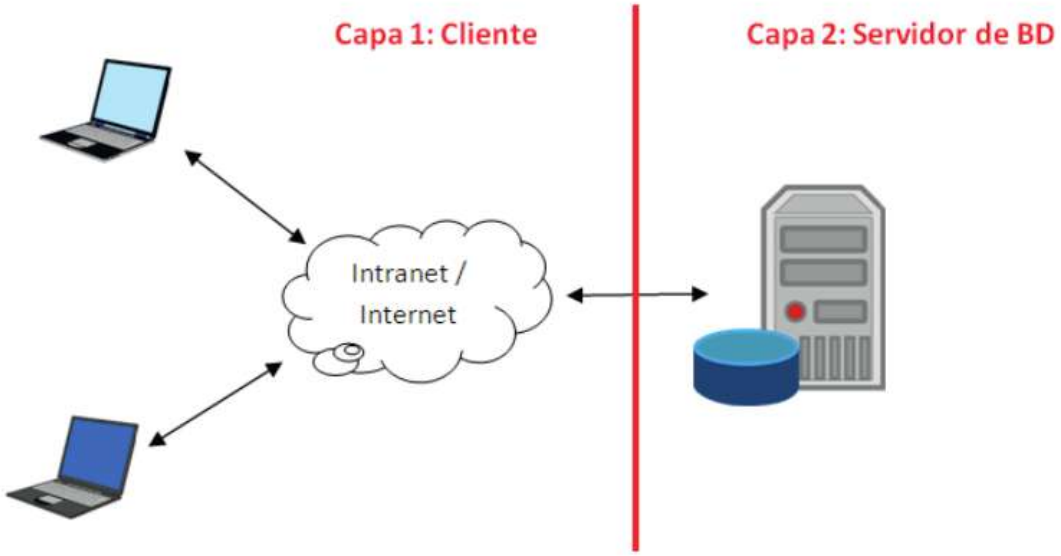


Figura 1.9. *Esquema de la arquitectura cliente-servidor de dos capas*

Asimismo, podemos distinguir dos tipos de arquitecturas cliente-servidor en dos capas:

- **Clientes obesos (*thick clients*):** en esta configuración, la mayor parte de la gestión del procesamiento reside junto a la interfaz de usuario, es decir, en el lado del cliente. Por su parte, la gestión de la base de datos es propia del servidor.
- **Clientes delgados (*thin clients*):** por el contrario, en esta configuración, solo la interfaz de usuario reside en el cliente, mientras que la gestión del procesamiento y de la base de datos está ubicada en el servidor.

NOTA

Del mismo modo, también es posible que un servidor funcione como cliente de otro servidor, lo cual es denominado como **diseño de dos capas encadenado.**

- **Arquitectura de tres capas:** en esta arquitectura se ubica una tercera capa, denominada **servidor intermedio**, entre las dos anteriormente comentadas, y en ella se ejecutan las reglas y lógica de procesamiento (gestión de procesamiento).

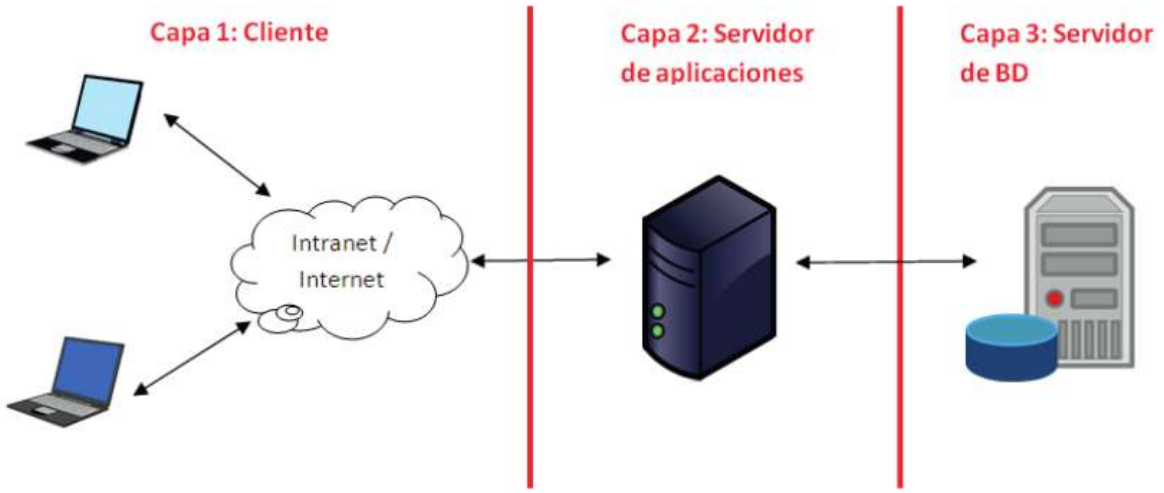


Figura 1.10. *Esquema de la arquitectura cliente-servidor de tres capas*

La arquitectura distribuida

Podemos definir una **base de datos distribuida** como aquella cuyos datos están repartidos entre más de una máquina, mientras que un **sistema de gestión de base de datos distribuidas** es el *software* que gestiona la anteriormente mencionada base de datos distribuida, haciendo que la distribución de la información sea transparente para los usuarios de la base de datos. De hecho, una base de datos distribuida es considera por los usuarios finales como una base de datos centralizada, lo cual implica una tarea más compleja por parte del DBMS distribuido, para mantener la coherencia e integridad de la información.

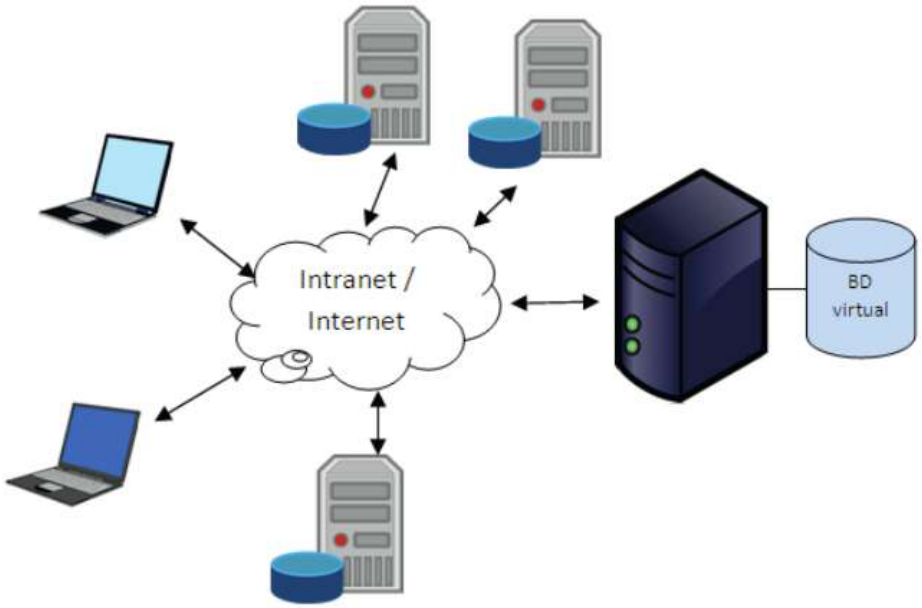


Figura 1.11. *Esquema de la arquitectura distribuida*

1.4 EJERCICIOS PROPUESTOS

1. Analiza las 12 reglas de Codd para los sistemas de datos relacionales y explica razonadamente cuáles son las tres reglas que consideras más importantes.

2. Busca en Internet las diferencias fundamentales entre el modelo relacional, el modelo jerárquico y el modelo de red. Plasma esta información en un esquema conceptual.

3. Busca información sobre el lenguaje SEQUEL: año de creación, principales palabras reservadas, sistemas que lo utilizaron, etc.

4. Explica con tus palabras los siguientes conceptos relacionados con bases de datos:

- Datos
- Base de datos
- Sistema de gestión de bases de datos
- Redundancia
- Incoherencia
- Privilegio
- Base de datos centralizada
- Base de datos distribuida

5. Busca en Internet un ejemplo de producto *software* que emplea bases de datos distribuidas.

1.5 TEST DE CONOCIMIENTOS

1 ¿Cuál fue el primer soporte para almacenar la información informatizada de la historia de las bases de datos?:
a) Discos
b) Cintas
c) DVD
d) Listas y Árboles

2 Microsoft Access es un ejemplo de base de datos…
a) Relacional
b) Jerárquica
c) De red
d) Orientada a objetos

3 ¿Cuál de estos conceptos NO es un problema que resuelve una base de datos?:
- **a)** Redundancia
- **b)** Incoherencias
- **c)** Polimorfismo
- **d)** Pérdida de dependencias funcionales

4 Señala la afirmación correcta:
- **a)** El administrador de una base de datos tiene dos principales herramientas para mantener la seguridad en la base de datos: las vistas y los permisos.
- **b)** El administrador de una base de datos tiene tres principales herramientas para mantener la seguridad en la base de datos: las vistas, los índices y los permisos.
- **c)** El administrador de una base de datos tiene dos principales herramientas para solucionar los problemas de integridad en la base de datos: las vistas y los permisos.
- **d)** El administrador de una base de datos tiene tres principales herramientas para solucionar los problemas de integridad en la base de datos: las vistas, los índices y los permisos.

5 El nivel que tiene como responsabilidad el almacenamiento físico de los datos y el acceso a ellos es el…
- **a)** Nivel interno
- **b)** Nivel físico
- **c)** (a) y (b) son correctas
- **d)** Nivel conceptual

6 El nivel responsable de la descripción de la estructura de los datos y de sus relaciones es el…
- **a)** Nivel físico
- **b)** Nivel externo
- **c)** Nivel conceptual
- **d)** Ningún nivel tiene esa responsabilidad

7 Indica cuál de los modelos no corresponde a un modelo basado en registros:
- **a)** Modelo relacional
- **b)** Modelo de red
- **c)** Modelo jerárquico
- **d)** Modelo físico

8 Señala la afirmación incorrecta:
- **a)** Los lenguajes de manejo de bases de datos procedimentales se basan en el álgebra relacional y su lenguaje más representativo es SQL.
- **b)** En los lenguajes de manejo de bases de datos no procedimentales el usuario indica la secuencia en la que se llevan a cabo las operaciones a realizar.
- **c)** QUEL y QBE son dos ejemplos de lenguajes no procedimentales.
- **d)** En los lenguajes no procedimentales el usuario solo puede describir los datos y las consultas.

9 ¿Cuáles de estas no es una función del DBMS?:
a) Definir los datos.
b) Aportar la seguridad a la base de datos.
c) Recuperar el sistema frente a fallos.
d) Definir el diseño lógico y conceptual.

10 Los usuarios de una base de datos se clasifican en:
a) DBA, programadores, usuarios sofisticados, usuarios especializados y usuarios ingenuos
b) DBA, usuarios sofisticados y no sofisticados
c) Depende del nivel de seguridad de la base de datos
d) (b) y (c) son correctas

11 En una arquitectura distribuida…
a) La base de datos se encuentra almacenada en su totalidad en un solo lugar físico, conectado o no a Internet.
b) Puede haber varios servidores que almacenen la base de datos, siempre y cuando no procesen los mismos datos.
c) Pueden presentar una arquitectura en dos o en tres capas.
d) Los datos de la base de datos están repartidos en más de un servidor.

12 Los clientes obesos y los clientes delgados pertenecen al paradigma de…
a) La arquitectura cliente-servidor de tres capas
b) La arquitectura distribuida de dos capas
c) La arquitectura paralela
d) Ninguna de las anteriores

13 ¿Puede un servidor funcionar como cliente de otro servidor?:
a) Nunca, porque provocaría incoherencias.
b) Sí, pero necesitaría una arquitectura de tres capas.
c) Sí, en arquitecturas de dos capas.
d) Sí, solo para base de datos distribuidas.

NOTA

Las cuestiones 14 y 15 pertenecen a un ejercicio de las **Oposiciones al Cuerpo de Técnicos Auxiliares de Informática de la Administración del Estado (Convocatoria de 2015)**.

14 ¿Cuál de las siguientes opciones NO representa uno de los niveles de abstracción de base de datos?:
a) Interno
b) Externo
c) Sintáctico
d) Conceptual

15 El modelo de datos relacional fue introducido por:
a) Codd
b) Hamming
c) Ritchie
d) Adleman

2 Modelos conceptuales de bases de datos

2.1 EL MODELO ENTIDAD-RELACIÓN

Propuesto por Peter Chen en 1976, en modelo Entidad-Relación (también conocido como modelo E-R) es el modelo conceptual de alto nivel más extendido en la actualidad. Presenta un alto nivel de abstracción y se fundamenta en percibir la realidad como una serie de entidades (objetos que existen en la realidad) y de relaciones entre esos objetos. Tanto las entidades como las relaciones contienen atributos.

2.1.1 ENTIDADES

Una entidad es un objeto que existe y se puede distinguir de otros. El conjunto de las entidades del mismo tipo se denomina *Conjunto de Entidades*[6].

Un ejemplo de entidades, siguiendo la Figura 1.3, serían las entidades *Alumnos* y *Asignaturas*.

2.1.2 INTERRELACIONES: CARDINALIDAD, ROL Y GRADO

Las interrelaciones o relaciones representan asociaciones entre entidades. El conjunto de las relaciones del mismo tipo se denomina *Conjunto de Relaciones*.

Siguiendo el mismo ejemplo anterior, en la base de datos de la academia de enseñanza donde almacenamos *Alumnos* y *Asignaturas,* existirá una relación que asocie ambas entidades, que hemos denominado *Estudian* la cual informa sobre qué alumnos están estudiando qué asignaturas.

Grado

El grado hace referencia al número de entidades relacionadas. Los tipos de grado pueden ser:

- **Relaciones de grado 1 (reflexivas):** en la que se relaciona una entidad consigo misma.

  Por ejemplo una relación *Es continuación* que especifique si una/s *Asignatura/s* son continuación de otra *Asignatura* concreta.

- **Relaciones de grado 2 (binarias):** cuando se relacionan dos entidades. Se trata del grado más usual.

  El ejemplo de este tipo de grado lo tendríamos en la relación *Estudian*, antes comentada.

- **Relaciones de grado superior a 2 (ternarias, cuaternarias, etc.):** aquellas que asocian más de dos entidades en la relación.

  Por ejemplo, si en el ejemplo de la Figura 1.3 tuviéramos otra entidad *Libro de Texto*, se podría relacionar qué *Alumno* estudia qué *Asignatura* y, además, con qué *Libro de Texto* estudia.

Cardinalidad

Los conjuntos de relaciones pueden tener restricciones. La más frecuente es la cardinalidad de asignación, que limita el número de entidades relacionadas con una entidad del otro conjunto de entidades.

6 Por economía, al conjunto de entidades y al conjunto de relaciones se le denomina entidades y relaciones, respectivamente.

Siguiendo el mismo ejemplo, el conjunto de relaciones binarias entre *Alumnos* y *Asignaturas* podría tener cuadro posibles restricciones de cardinalidad:

- **Uno a Uno (1:1):**
  - 1 alumno estudia 1 asignatura.
  - 1 asignatura tiene 1 alumno.
- **Uno a Muchos (1:M):**
  - 1 alumno estudia muchas asignaturas.
  - 1 asignatura tiene 1 alumno.
- **Muchos a Uno (M:1):**
  - 1 alumno estudia 1 asignatura.
  - 1 asignatura tiene muchos alumnos.
- **Muchos a Muchos (M:M):**
  - 1 alumno estudia muchas asignaturas.
  - 1 asignatura tiene muchos alumnos.

De los cuatro casos, habría que buscar el que correspondiera a la realidad del problema. En este ejemplo concreto, la cardinalidad a escoger sería la última, es decir, M:M.

Rol

Referido a las relaciones de grado 1, opcionalmente se puede especificar el rol que desempeña cada parte de la entidad en la relación.

2.1.3 DOMINIOS Y VALORES

Podemos definir el **dominio** de un atributo (cuya definición la veremos en el siguiente apartado) como la especificación de todos los valores que pueden estar contenidos en dicho atributo. Se trata pues de una restricción de cuáles son los **valores** aceptables para ese atributo concreto.

Algunos dominios usuales para una base de datos suelen ser *cadenas de texto*, carácter, *enteros*, *reales*, fechas, etc.

Los dominios pueden ser también compuestos, a partir de otros (*año*, *mes* y *día*→*fecha*).

2.1.4 ATRIBUTOS

Los atributos almacenan las propiedades que nos interesa conservar de las entidades, ya que las propiedades que definen dichas entidades.

Por ejemplo, en la Figura 1.3 podemos identificar las siguientes entidades con sus correspondientes atributos:

- Entidad *Alumnos*→ Atributos: *DNI_Al* y *Nombre_Al*
- Entidad *Asignaturas*→ Atributos: *Código_As* y *Nombre_As*

2.1.5 PROPIEDADES IDENTIFICATORIAS

Tenemos tres tipos de atributos:

- Atributos de una entidad.
- Atributos que sirven para identificar entidades, conocidos como *clave*. Estos son los que poseen la **propiedad identificatoria**. Hablaremos de ellos en el apartado 3.3.
- Atributos de una relación, conocidos como *atributos descriptivos*.

2.1.6 DIAGRAMAS ENTIDAD-RELACIÓN. SIMBOLOGÍA

Las **entidades** se representan mediante un rectángulo. Por ejemplo:

Figura 2.1. *Símbolo correspondiente a una entidad en el modelo E-R*

A su vez, las **relaciones** se representan mediante un *rombo,* que une la entidad o las entidades involucradas en dicha relación.

Siguiendo con los ejemplos expuestos en el apartado 2.1.2, tenemos las siguientes relaciones:

- **Relaciones de grado 1 (reflexivas):** una relación *Es continuación* que especifique si una/s *Asignatura* / *s* son continuación de otra *Asignatura* concreta.

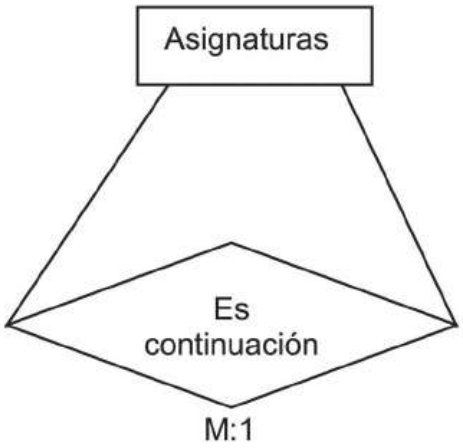


Figura 2.2. *Representación en el modelo E-R de una relación reflexiva*

- **Relaciones de grado 2 (binarias):** el ejemplo de la relación *Estudian*, entre las entidades *Alumnos* y *Asignaturas*.

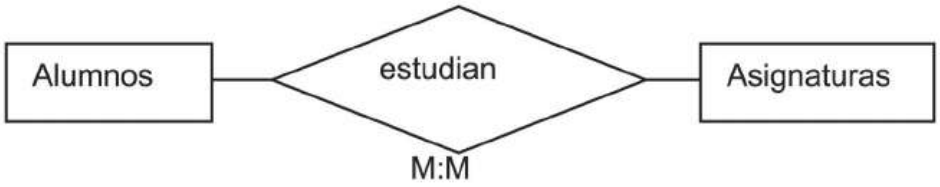


Figura 2.3. *Representación en el modelo E-R de dos entidades relacionadas (relación binaria)*

- **Relaciones de grado superior a 2 (ternarias, cuaternarias, etc.):** ejemplo en el que involucramos otra entidad *Libro de Texto*, y una relación que asocia qué *Alumno* estudia qué *Asignatura* y, además, con qué *Libro de Texto* estudia.

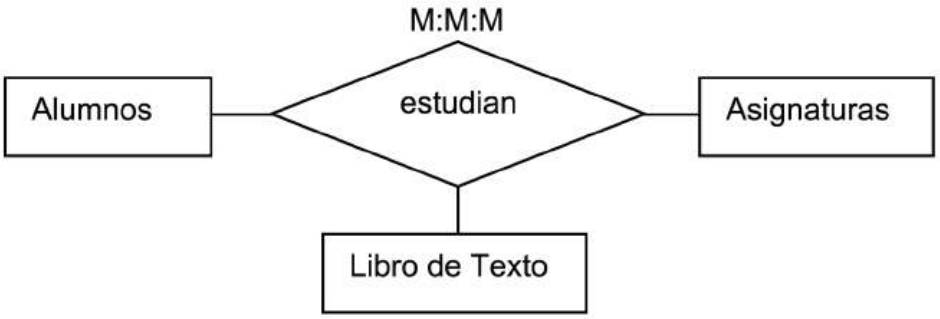


Figura 2.4. *Representación en el modelo E-R de una relación ternaria*

Del mismo modo, se reflejan las restricciones de cardinalidad mediante la numeración que acompaña la citada relación, con los valores posibles que hemos estudiado en el apartado 2.1.2.

Por su parte, los **atributos** también se dividían en tres tipos:

- **Atributos de una entidad.**
- **Atributos que sirven para identificar tuplas, conocidos como clave primaria.**
- **Atributos de una relación, conocidos como atributos descriptivos.** Hablaremos de este tipo de atributos en el apartado siguiente.

Dichos atributos se representan mediante una elipse, subrayándose el nombre del atributo, en caso de ser clave primaria.

En la siguiente figura podemos ver un ejemplo de dos entidades relacionadas y atributos en las dos entidades.

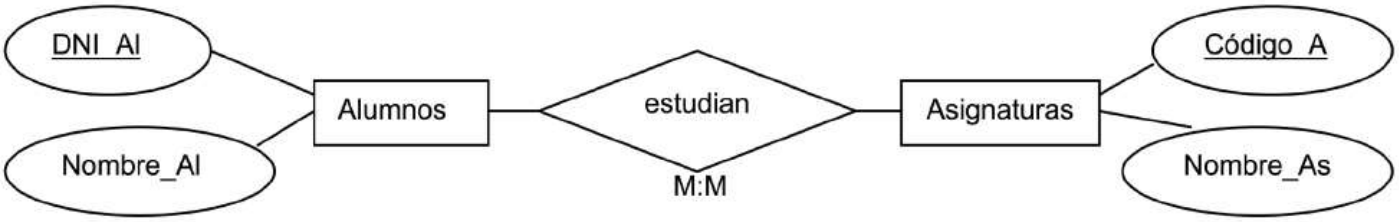


Figura 2.5. *Representación en el modelo E-R de dos entidades (con sus atributos) relacionadas*

2.2 EL MODELO ENTIDAD-RELACIÓN EXTENDIDO

Al modelo Entidad-Relación clásico de Peter Chen, se le han ido incorporando sucesivas ampliaciones, también consideradas por otros autores como simplificaciones del modelo, o incluso mejoras al mismo, según otros.

En la modesta opinión de quienes firman este libro, el modelo entidad-relación extendido, en términos generales, no es más que una simplificación en notación conceptual, ya que, al fin y al cabo, el diseño físico que realmente se genera es prácticamente análogo usando el modelo clásico de Chen o el extendido.

Así pues, a continuación veremos algunas de las notaciones extendidas que más se emplean en la literatura y que emplearemos en las actividades del presente libro. Del mismo modo, citaremos, como notas, algunas otras extensiones que, aunque no las vamos a utilizar, pueden aparecer en otros libros de texto.

Cardinalidad extendida en relaciones

Puede que esta extensión sea la única que no supone una simplificación sino una información más rica acerca de la cardinalidad de la relación entre *N* entidades.

Así, esta cardinalidad extendida supone la obligación de especificar el número mínimo y máximo de entidades de un conjunto de entidades que pueden estar relacionados con una entidad del otro conjunto de entidades (u otros) que participan en la relación.

Para ejemplificar este nuevo concepto, aplicaremos la notación de cardinalidad extendida a las anteriores Figuras 2.2, 2.3 y 2.4:

- **Relaciones de grado 1 (reflexivas):** según la cardinalidad del modelo clásico, la relación *Es continuación* significaba que *"1 Asignatura está precedida por 1 Asignatura y que 1 Asignatura podía ser predecesora de Muchas Asignaturas"*.

  Pues ahora, con la nueva notación extendida, tenemos la siguiente información:

  – *1 Asignatura puede estar precedida como mínimo de 0 Asignaturas (caso de asignaturas que, por ejemplo, son de primer curso y no son continuación de ninguna) y como máximo de 1 Asignatura (caso de sí tener continuación de otra asignatura anterior).*

  – *1 Asignatura puede ser predecesora como mínimo de 0 Asignaturas (caso de que sea una asignatura de último curso y no continúe con otra) y como máximo de Muchas Asignaturas (caso de que tenga varias asignaturas como desarrollo de esta).*

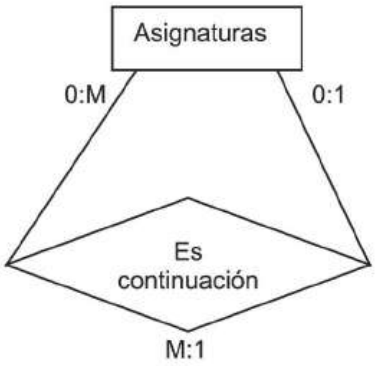


Figura 2.6. *Representación en el modelo E-R extendido de una relación reflexiva*

- **Relaciones de grado 2 (binarias):** el ejemplo de la relación *Estudian*, entre las entidades *Alumnos* y *Asignaturas* expresaba que *"1 Alumno puede estudiar Muchas Asignaturas y que 1 Asignatura puede ser estudiada por Muchos Alumnos"*.

  Mientras que con el modelo extendido tenemos que:

  – *1 Alumno puede estudiar como mínimo 1 Asignatura (si no fuera así, no estaría en la base de datos de la academia de formación) y como máximo Muchas Asignaturas.*

  – *1 Asignatura puede ser estudiada como mínimo por 0 Alumnos (caso de asignaturas que no hayan tenido demanda o sean de otra convocatoria diferente a la actual) y como máximo por Muchos Alumnos.*

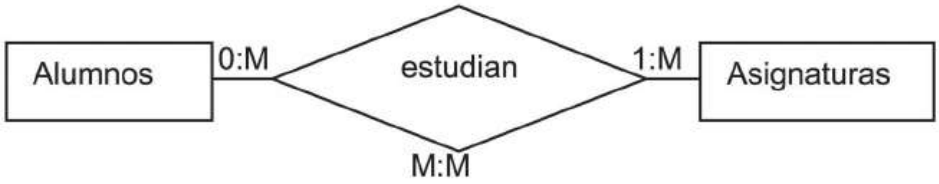


Figura 2.7. *Representación en el modelo E-R extendido de dos entidades relacionadas (relación binaria)*

- **Relaciones de grado superior a 2 (ternarias, cuaternarias, etc.):** en este caso, la información que nos proporcionaba la cardinalidad de Chen consistía en que:

  – "Muchos Alumnos pueden estudiar 1 Asignatura con 1 determinado Libro de Texto".
  – "Muchos Libros de Texto pueden ser estudiados por 1 Alumno en 1 Asignatura".
  – "Muchas Asignaturas pueden ser estudiadas por 1 Alumno usando 1 determinado Libro de Texto".

  Mientras que, siguiendo el modelo E-R extendido, tenemos que:

  – "Como mínimo 1 y como máximo Muchos Alumnos pueden estudiar 1 Asignatura con 1 determinado Libro de Texto".

  – "Como mínimo 0 y como máximo Muchos Libros de Texto pueden ser estudiados por 1 Alumno en 1 Asignatura".

  – "Como mínimo 0 y como máximo Muchas Asignaturas pueden ser estudiadas por 1 Alumno usando 1 determinado Libro de Texto".

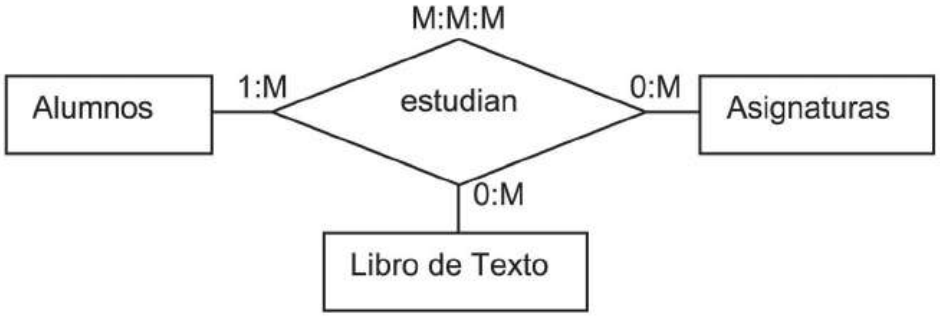


Figura 2.8. *Representación en el modelo E-R extendido de una relación ternaria*

NOTA

Como hemos podido comprobar en los ejemplos anteriores, las cardinalidades máximas coinciden con el tipo de correspondencia de modelo clásico de Chen.

NOTA

Relaciones exclusivas

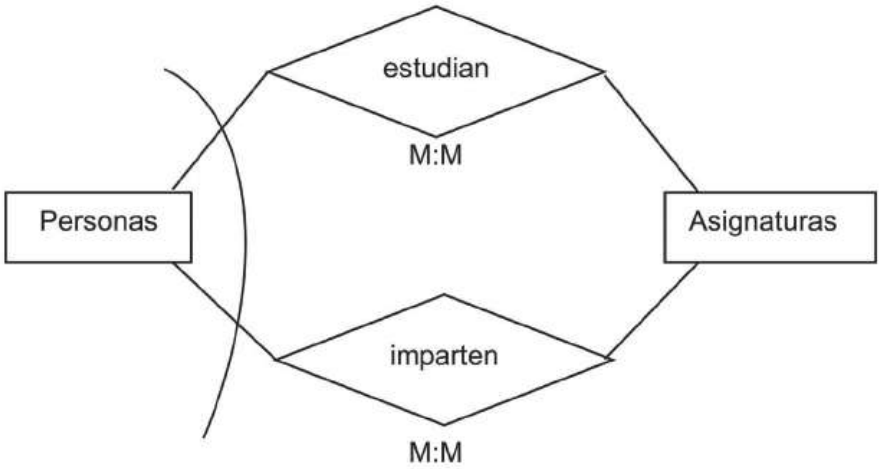
Aunque no vamos a usar estar notación extendida en este libro, una relación exclusiva se produce cuando una entidad no puede estar relacionada simultáneamente, sino que el participar en una relación es excluyente con participar con otra, y viceversa.

Un ejemplo que ilustra esta restricción en las relaciones podría considerarse si, en lugar de tener una entidad *Alumnos*, tuviéramos una entidad *Persona*, que englobara tanto a alumnos como a profesores. Entonces podríamos definir dos relaciones:

- *Imparte*, que define qué profesores imparten qué asignaturas.
- *Estudian,* que define qué alumnos reciben qué asignaturas.

De este modo, es obvio plantear que una persona no puede estudiar y a la vez impartir una asignatura.

La notación sería la siguiente:



Esta situación se podría resolver fácilmente con el uso de generalización/especialización, que veremos más adelante, no siendo necesaria esta relación exclusiva.

Entidades fuertes y entidades débiles

Se puede hacer distinción entre dos tipos de entidades:

- **Entidades fuertes:** su existencia no depende de ninguna otra entidad. Como antes comentábamos, las entidades fuertes se representan mediante un rectángulo.
- **Entidades débiles:** su existencia está condicionada a la existencia de otra entidad. Se representa mediante un doble rectángulo.

  Por ejemplo, si quisiéramos almacenar una entidad llamada *Cualidades_Alumnos* dependiente de *Alumnos*, esto se representaría de la siguiente forma:

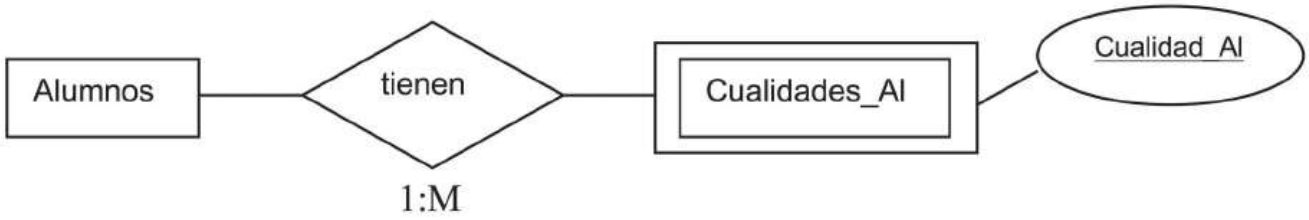


Figura 2.9. *Representación en el modelo E-R de dos entidades (una de ellas débil) relacionadas*

NOTA

Nótese que la **relación** entre una entidad débil y la entidad de la que depende **siempre seguirá una cardinalidad 1:M**, como se puede observar en la Figura 2.9.

Atributos descriptivos

En la sección anterior hablábamos de un tercer tipo de atributo que definía no una entidad, sino la relación que se producía entre una o varias entidades. A este tipo de atributos se le conoce como **atributo descriptivo** y se representa como una elipse, pero esta vez, que pende de una relación, no de una entidad.

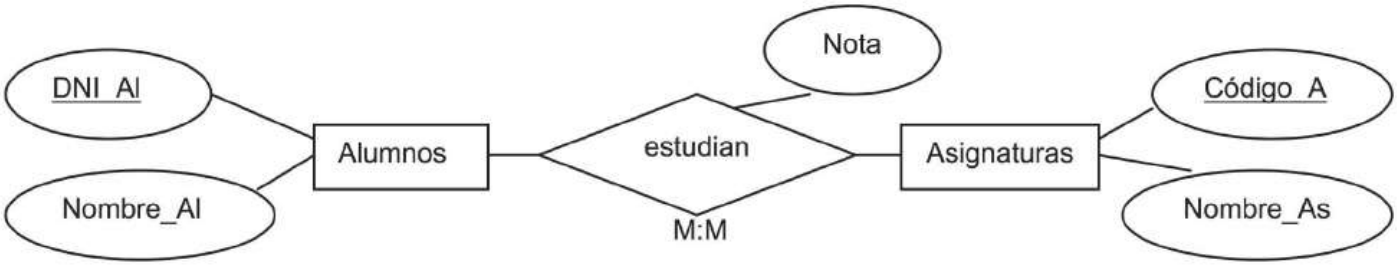


Figura 2.10. *Ejemplo de una relación que almacena, como atributo descriptivo la Nota de la asociación producida "cuando un Alumno estudia una Asignatura"* ***Atributos multivaluados***

Los **atributos multivaluados** son un claro de ejemplo de la intención de simplificación que tiene el modelo Entidad-Relación extendido.

Por definición, los atributos no pueden tener dominios que, a su vez, sean subconjuntos o, dicho de otra forma, un atributo solo puede tener un único valor. Así, si queremos definir una entidad con varios valores para un mismo atributo tendríamos que generar una entidad débil, como hemos visto en la Figura 2.9, en la que queremos asignarle a los *Alumnos* más de una *Cualidad_Al*.

El uso de un atributo multivaluado simplificaría la notación anteriormente expuesta, sustituyendo la entidad débil por un atributo multivaluado, aunque la posterior implementación física generaría el mismo resultado.

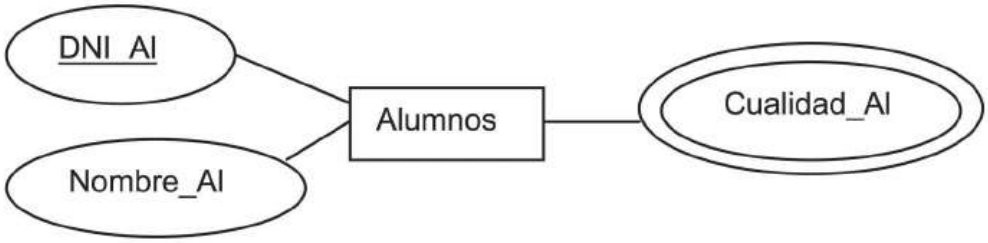


Figura 2.11. *Ejemplo de un modelo E-R extendido con una atributo multivaluado*

Generalización/Especialización

Se trata de un mecanismo muy útil de abstracción, que permite definir relaciones y asignar atributos de subclases entre objetos.

Explicamos ambos fenómenos de manera conjunta ya que ambos son el antónimo recíproco del otro. Por lo tanto, podemos definirlos de la siguiente forma:

- **Generalización:** proceso de abstracción que lleva a la definición de un nuevo tipo de entidad (superclase) a partir de elementos "similares" de otras (subclases).
- **Especialización:** proceso de reclasificación que permite obtener nuevas entidades (subclase) a partir de las "diferencias" entre los elementos de una clase superior (superclase).

NOTA

Notación y otros aspectos a tener en cuenta:

- La cardinalidad es siempre 1:1 en la superclase y 0:1 o 1:1 en las subclases.
- Las subclases heredan los atributos de la superclase.
- Se denota con un triángulo con la base paralela a la superclase (en el triángulo se puede incluir un atributo, llamado **atributo selector**).
- Se pueden añadir dos propiedades a la generalización/especialización:
  - **Exclusión/Solapamiento:** denotado por un arco.
  - **Totalidad/Parcialidad:** denotado por un círculo en el arco.

A continuación veremos una ilustración de cómo resolvemos el problema que reflejábamos en la nota sobre el uso de las *relaciones exclusivas*.

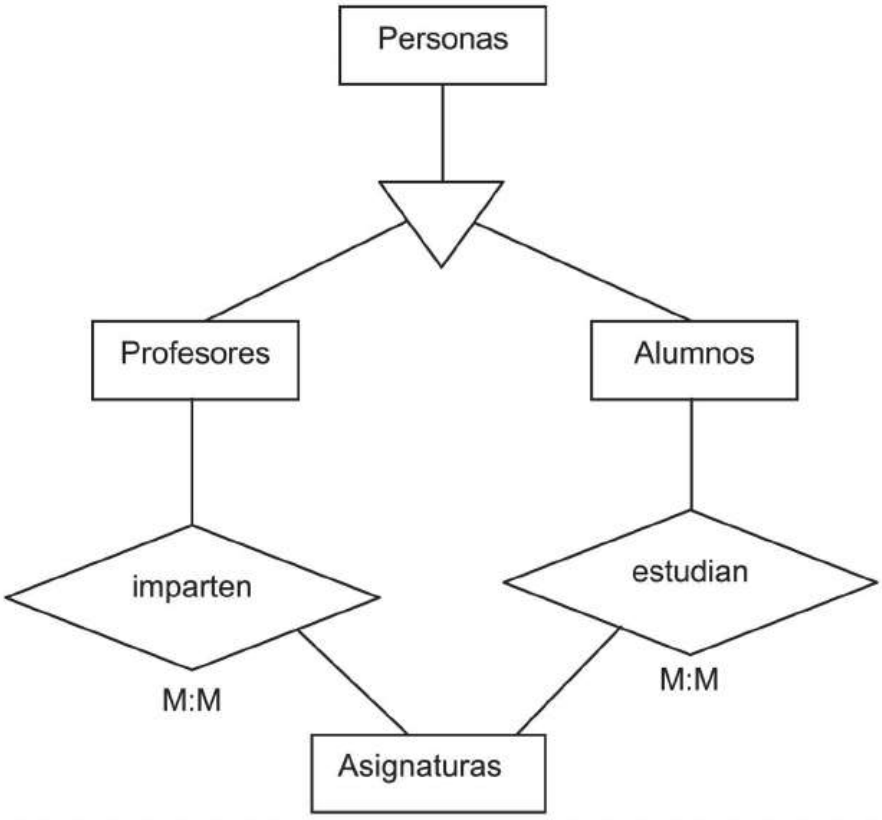
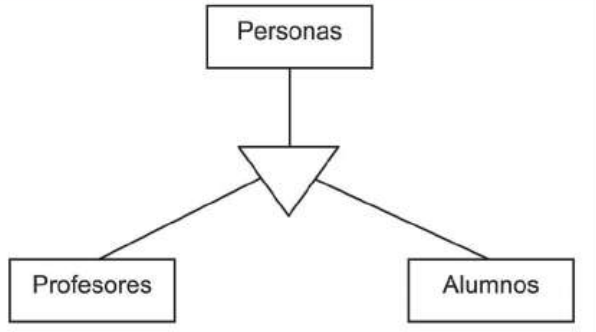
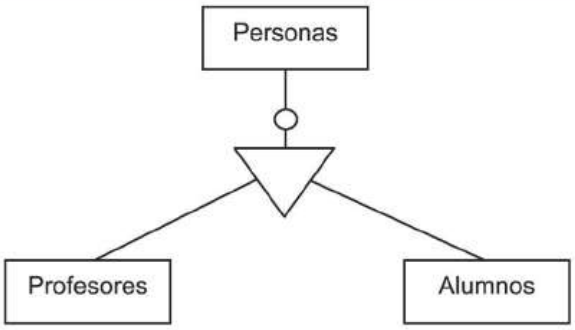
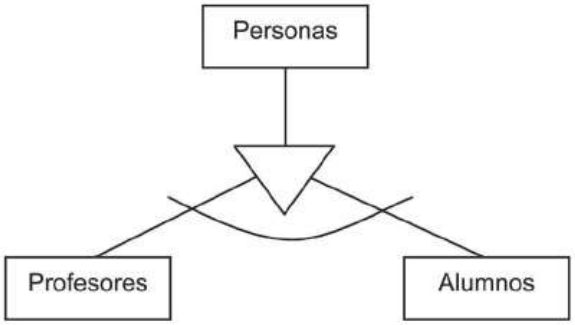
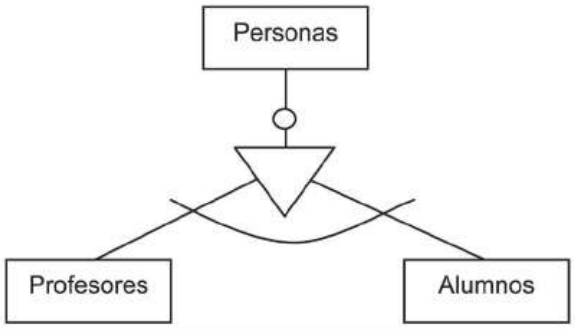


Figura 2.12. *Ejemplo de un modelo E-R extendido con una generalización*

Y, por último, ahondaremos en el significado de las dos propiedades que acabamos de definir con ayuda del siguiente cuadro comparativo:

Propiedades de la generalización	Significado
	**Generalización parcial y con solapamiento** Significa que puede haber *Personas* que no sean ni *Profesores* ni *Alumnos*, por ejemplo *Administrativos* o *Directivos* de la academia **(parcial)** y, por otro lado, que una *Persona* puede actuar, a veces como *Profesor*, a veces como *Alumno* **(solapamiento)**[1].

7 En los centros de formación sucede en ocasiones que un profesor también aprovecha para formarse.

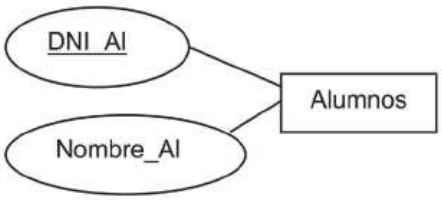
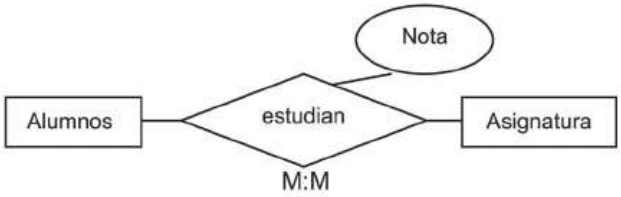
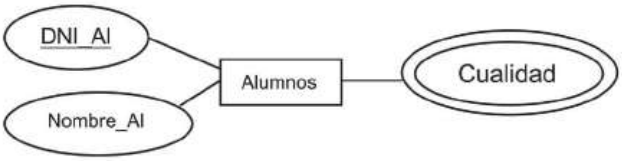
Propiedades de la generalización	Significado
	**Generalización total y con solapamiento (o solapada)** Significa que todas las *Personas* deben ser exclusivamente o *Profesores* o *Alumnos* (no pueden pertenecer a otros sectores) **(total)** y, por otro lado, que una *Persona* puede actuar, a veces como *Profesor*, a veces como *Alumno* **(solapada)**.
	**Generalización parcial y exclusiva** Significa que puede haber *Personas* que no sean ni *Profesores* ni *Alumnos*, por ejemplo *Administrativos* o *Directivos* de la academia **(parcial)** y, por otro lado, que una *Persona* solo puede actuar como *Profesor* o como *Alumno* pero no se pueden alterar sus roles **(exclusiva).**
	**Generalización total y exclusiva** Significa que todas las *Personas* deben ser exclusivamente o *Profesores* o *Alumnos* (no pueden pertenecer a otros sectores) **(total)** y, por otro lado, que una *Persona* solo puede actuar como *Profesor* o como *Alumno* pero no se pueden alterar sus roles **(exclusiva).**

Agregación

Vamos a comenzar exponiendo una comparativa de cómo podemos contemplar un atributo, en base a dos características propias:

- **Número de elementos en su dominio,** es decir, si dicho atributo contiene solo un valor o puede contener varios valores.
- **A cuántas entidades define,** o lo que es lo mismo, si el atributo califica una sola entidad o varias (esto es, a alguna relación entre entidades).

Veamos pues el resultado de la comparativa:

<table>
<tr>
<td>1 valor, 1 entidad → Atributo
 
</td>
<td>1 valor, M entidades → Atributo descriptivo
 
</td>
</tr>
<tr>
<td>M valores, 1 entidad → Atributo multivaluado
 
</td>
<td>M valores, M entidades →
AGREGACIÓN</td>
</tr>
</table>

De esta forma, podemos concluir que la **agregación** es una combinación de características y soluciones en el modelo E-R extendido que engloba:

- Los atributos descriptivos, ya que esos atributos no califican a una sola entidad sino a un grupo de ellas (a través de una relación).
- Los atributos multivaluados, ya que los dominios de esos atributos, a su vez, son subconjuntos.

En lo que respecta al modelado E-R, esos atributos, al ser multivaluados, deben representarse mediante una entidad, y, por otro lado, como califican a varias entidades, dicha nueva entidad la tenemos que relacionar con la relación.

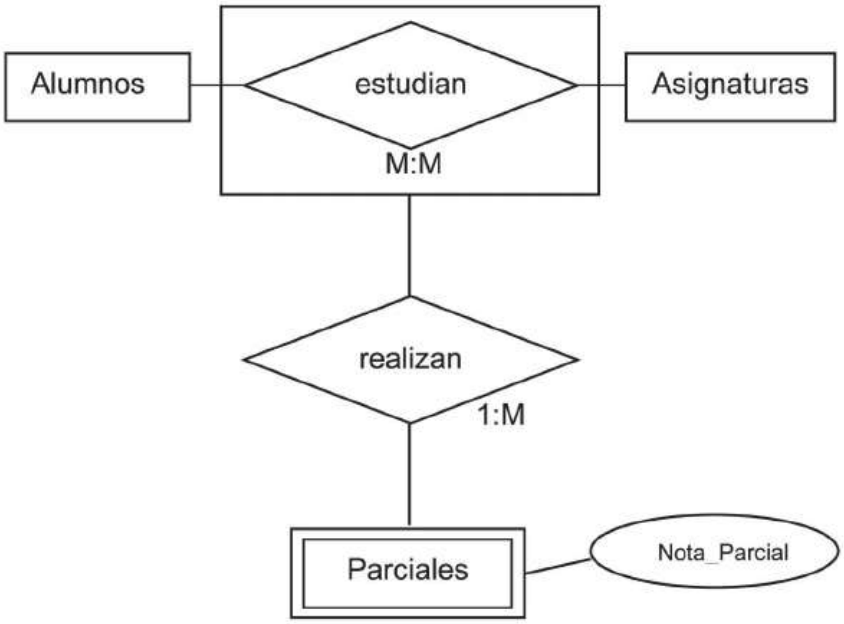
Para resolver este problema, surge el concepto de **agregación**, que es un mecanismo de abstracción que lleva a considerar una relación y las entidades que participen en ella, como una nueva entidad.

EJEMPLO 2.1

Supongamos que queremos almacenar las notas de todos los parciales que ha realizado un alumno en cada asignatura.

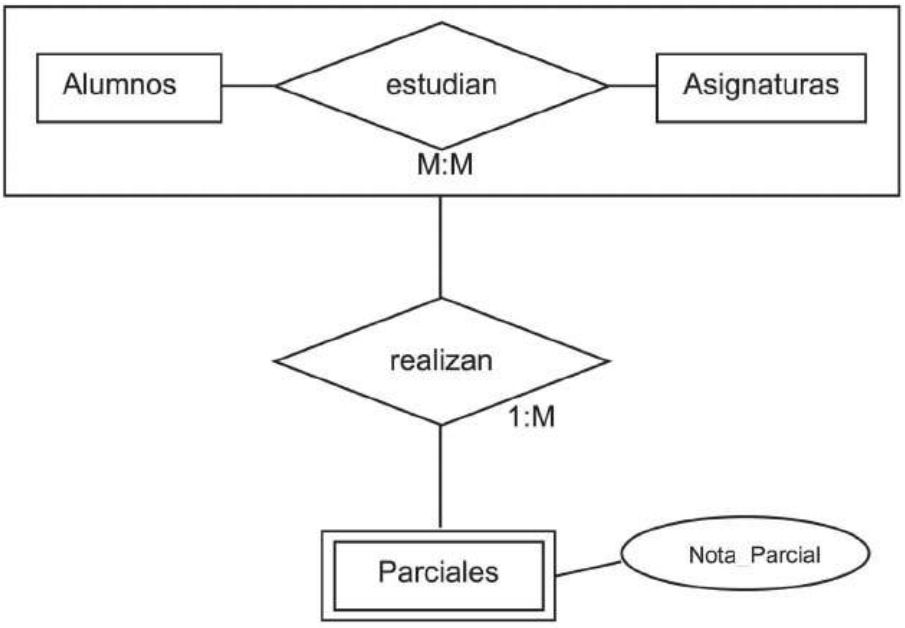
Ante este planteamiento, debemos tener en cuenta que el atributo *Nota_Parcial* no tiene un único valor, sino que dependiendo de la *Asignatura* y de cuándo el *Alumno* la ha cursado, tendremos más o menos notas. A su vez, las notas no califican a un *Alumno* ni a una *Asignatura* únicamente, sino que califican a la relación entre ambos, es decir, a la relación *Estudian*.

Por tanto, el modelo Entidad-Relación que plantearíamos ante este problema sería:



NOTA

Otros autores simbolizan la **agregación** enmarcando no solo la relación sino la relación y las entidades relacionadas. Luego quedaría de la siguiente forma:



EJERCICIO GUIADO 2.1

Vamos realizar el diseño conceptual de una base de datos mediante un diagrama E-R. La base de datos gestionará la información de una cadena de supermercados, la cual tiene varios supermercados. Cada uno de los supermercados tiene un código único, un domicilio, una superficie y un teléfono. La cadena tiene empleados, que pueden trabajar en varios supermercados. Para cada empleado, tenemos su NIF, nombre, domicilio y capacidades. Cada empleado desempeña una función en cada supermercado en el que trabaja.

Los supermercados sirven pedidos a domicilio. Para cada pedido, se anota un número de pedido (que puede repetirse en distintos supermercados), la fecha, la hora y el total. El pedido incluye el código, descripción, precio y cantidad de los artículos suministrados.

Cada pedido se sirve a un cliente, del que tenemos su NIF, nombre y domicilio y es encargado por un empleado.

Antes de mostrar el diagrama E-R resultante, vamos a analizar la información más relevante de este enunciado:

- *Capacidades* del *Empleado* no tiene por qué tener un solo valor. En algunos casos, un *Empleado* tendrá varias capacidades (ejemplo, "Ser ordenado" y "Ser limpio"), en otras, solo tendrá una capacidad e, incluso, puede haber empleados que carezcan de capacidad alguna.

  Por tanto, *Capacidades* no puede ser un atributo, será un atributo multivaluado.

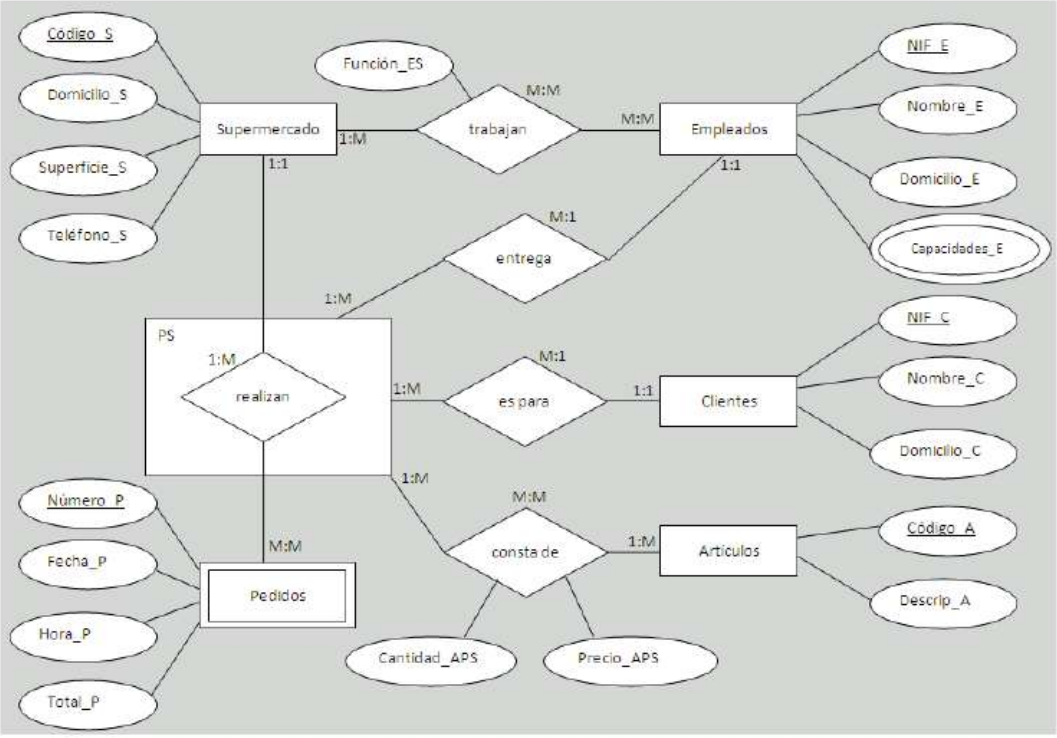
- El atributo *Función_Empleado* no califica a un *Empleado*, sino que indica "qué función desempeña un empleado en un supermercado", que puede ser distinta en un supermercado de los que trabaja que en otro.

  Esto significa que el atributo será un atributo descriptivo y estará asociado a la relación entre las entidades *Empleado* y *Supermercado*.

- El atributo *Número_Pedido* no identifica unívocamente un *Pedido* sino que necesitará también el atributo *Código_Supermercado* para identificarse. Así pues, podemos concluir que la entidad *Pedidos* será una entidad débil.

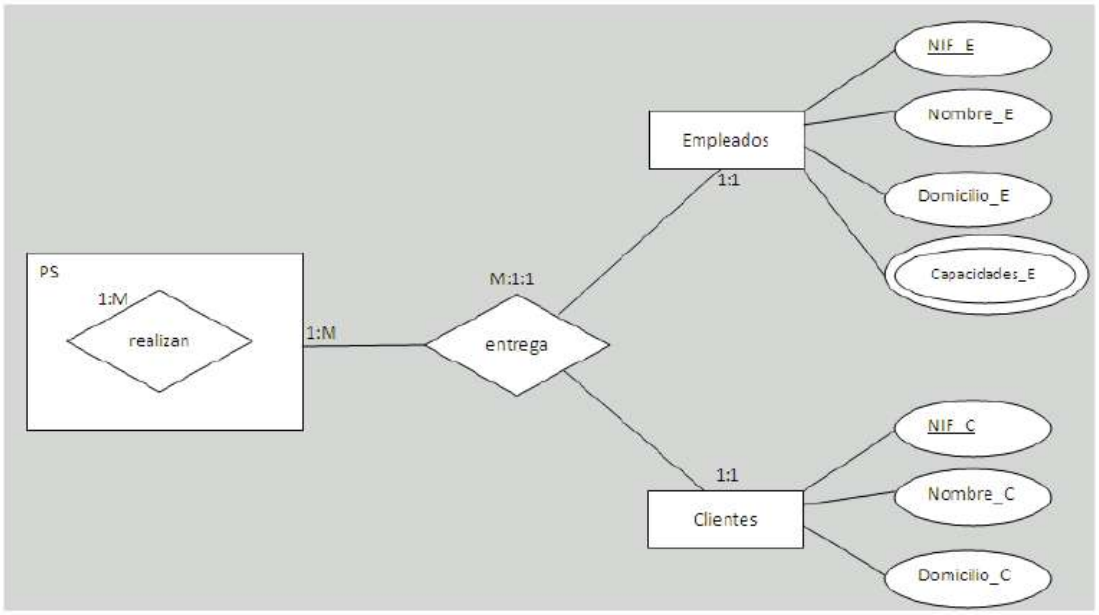
- La *Cantidad* y el *Precio* de los *Artículos* incluidos en cada *Pedido* serán diferentes, por lo que dichos atributos serán considerados atributos descriptivos.

Tras estas notas iniciales, observamos el diagrama Entidad-Relación resultante:



NOTA

Para representar la relación *"un pedido es servido a un cliente y entregado por un empleado"*, también podemos emplear una relación ternaria entre esas tres entidades.



2.3 RESTRICCIONES DE INTEGRIDAD

Se trata de unas condiciones de obligado cumplimiento por los datos de las bases de datos. Las hay de dos tipos:

- Restricciones inherentes.
- Restricciones explícitas, también llamadas del usuario o semánticas.

2.3.1 RESTRICCIONES INHERENTES

Las **restricciones inherentes** son aquellas que no son determinadas por los usuarios, sino que son definidas por el hecho de que la base de datos sea relacional. Por ejemplo, podemos citar las siguientes:

- No puede haber dos entidades iguales.
- El orden de las entidades no importa.
- El orden de los atributos, dentro de una entidad, no importa.

2.3.2 RESTRICCIONES EXPLÍCITAS

El modelo relacional permite a los usuarios incorporar restricciones personales a los datos. Los más extendidos en la literatura y en la práctica, son:

- **Clave primaria**. Hace que los atributos marcados como *clave primaria* no pueden repetir valores.
- **Unicidad.** Impide que los valores de los atributos marcados de esa forma puedan repetirse.
- **Obligatoriedad.** Prohíbe que el atributo marcado de esta forma no tenga ningún valor.
- **Integridad referencial.** Prohíbe colocar valores en una clave externa que no estén reflejados en la tabla donde ese atributo sea clave primaria.
- **Regla de validación.** Condición que debe cumplir un dato concreto para que sea actualizado.

2.4 EJERCICIOS RESUELTOS

NOTA

Este extracto de ejercicio pertenece a un ejercicio de las **Oposiciones a cuerpos docentes en Andalucía (Convocatoria de 2004), concretamente para la especialidad de *Sistemas y Aplicaciones Informáticas.***

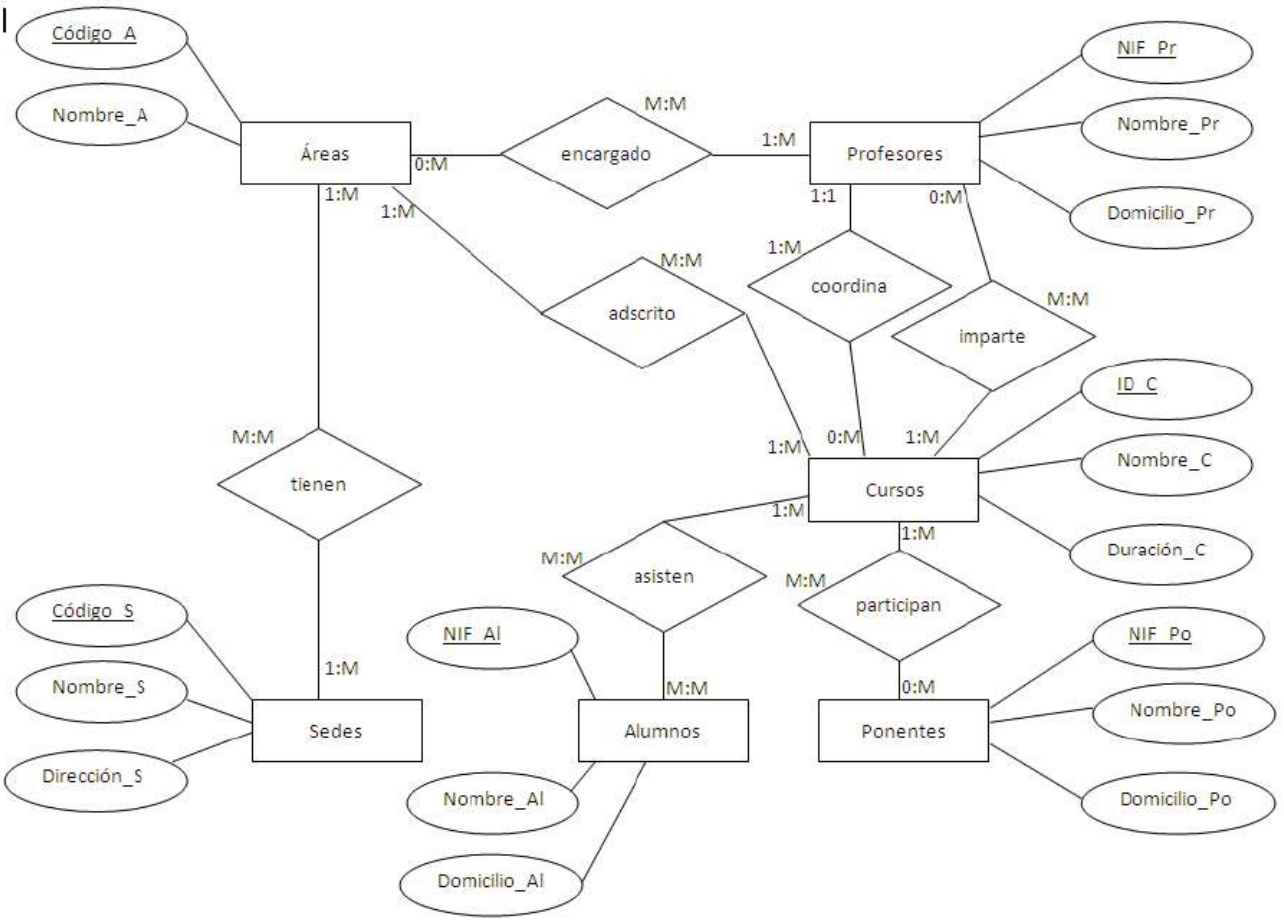
1. Diseñar una base de datos que recoja la organización de un centro de formación, dedicado a la gestión y desarrollo de cursos. Realizar el modelo de datos conceptual con el modelo Entidad-Relación.

Se considera que:

- El centro distribuye su actividad por áreas de conocimiento (matemáticas, francés, lengua, etc.), a cargo de cada una de ellas hay uno o varios profesores.
- El centro dispone de varias sedes (Sevilla, Málaga, Algeciras, etc.), no todas ellas con el mismo número de áreas, y un área puede estar en varias sedes.
- Un curso puede estar adscrito a una o varias áreas.
- Los cursos tienen un coordinador (uno de los profesores) y uno o varios ponentes. Estos ponentes pueden ser profesores del centro o ajenos al mismo.
- Los alumnos pueden realizar varios cursos.

Propuesta de resolución, teniendo en cuenta que:

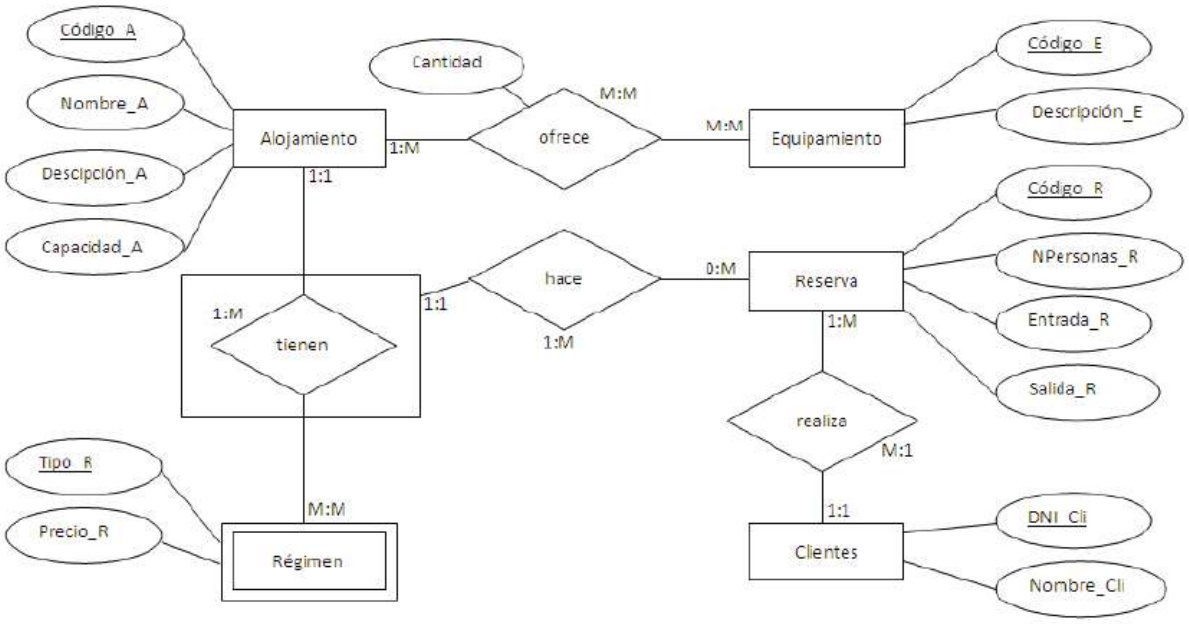
- Este tipo de ejercicios son subjetivos y admiten diversas resoluciones correctas.
- No dan información sobre los atributos a almacenar en cada entidad, por lo que nos "inventaremos" algunos.



2. Diseñar una base de datos que cumpla las siguientes premisas:

- Una agencia de alojamientos rurales desea crear una base de datos para su gestión. Cada alojamiento tiene un nombre, un código, una descripción y una capacidad. Además, tiene un conjunto de equipamientos, de los que anotamos el código, descripción y la cantidad por alojamiento.
- En cada alojamiento se pueden hacer reservas eligiendo el régimen de alojamiento (por ejemplo, solo alojamiento, alojamiento más desayuno, etc.). Para cada régimen se guarda el precio a pagar por persona.
- También se desean almacenar las reservas que se hacen de los alojamientos, anotando el DNI y nombre del que hace la reserva, así como el número de personas y el régimen de la reserva. Cada reserva tiene un código único y una fecha de entrada y salida.

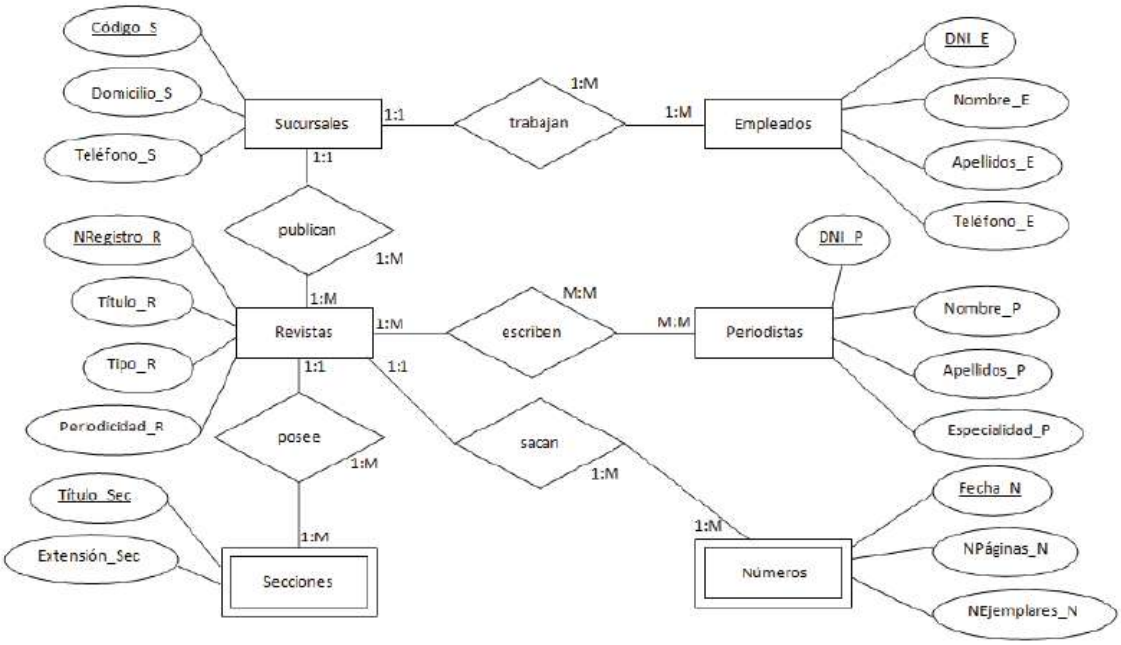
Propuesta de resolución:



3. Diseñar una base de datos sobre una cadena editorial que cumpla las siguientes premisas:

- La editorial tiene varias sucursales, con su domicilio, teléfono y un código de sucursal.
- Cada sucursal tiene varios empleados, de los cuales tendremos sus datos personales, DNI y teléfono. Un empleado trabaja en una única sucursal.
- En cada sucursal se publican varias revistas, de las que almacenaremos su título, número de registro, periodicidad y tipo.
- La editorial tiene periodistas, que pueden escribir artículos para varias revistas. Almacenaremos los mismos datos que para los empleados, añadiendo su especialidad. Guardaremos también las secciones fijas de cada revista, que tendrán un título y una extensión.
- Para cada revista, almacenaremos información de cada número, que incluirá la fecha, número de páginas y el número de ejemplares vendidos.

Propuesta de resolución:



2.5 EJERCICIOS PROPUESTOS

NOTA

Este extracto de ejercicio pertenece a un ejercicio de las **Oposiciones a cuerpos docentes en Andalucía (Convocatoria de 2006), concretamente para la especialidad de *Sistemas y Aplicaciones Informáticas.***

1. Diseñar una base de datos, mediante el modelo Entidad-Relación, que recoja el funcionamiento de una compañía de seguros en el ramo de viviendas, considerando los siguientes supuestos:

- La póliza es el elemento fundamental de información. Se identifica mediante un número único, tipo de cobertura (el contenido, el continente o ambos), e importe de cobertura máxima. La póliza pertenece a un único cliente y referencia a una única vivienda.
- Un cliente puede tener más de una póliza diferente.
- Una vivienda solo puede tener una póliza.
- De la vivienda es importante conocer su dirección completa, tipo (piso, casa adosada, chalet, etc.). Además, una vivienda puede tener una serie de extras (alarma, caja fuerte, etc.).

- Cuando se produce un siniestro, se crea un parte de accidente identificado por su número de siniestro, donde se recoge la información del mismo: póliza del cliente, datos del propietario, fecha del siniestro, número y nombre del perito que evalúa los datos, y datos de la empresa que va a llevar a cabo la reparación de cualquier tipo de siniestro.

NOTA

Este extracto de ejercicio pertenece a un ejercicio de las **Oposiciones a cuerpos docentes en Andalucía (Convocatoria de 2006), concretamente para la especialidad de *Sistemas y Aplicaciones Informáticas.***

2. Diseñar una base de datos, mediante el modelo Entidad-Relación, que recoja el funcionamiento de una cadena de herboristerías, considerando los siguientes supuestos:

- Las herboristerías están distribuidas en diferentes ciudades.
- Cada herboristería tiene un especialista y sus empleados.
- Cada herboristería tiene su stock de productos, que se mantiene por el identificador del producto y su presentación (ejemplo: infusión 20 sobre, 50 cápsulas, etc.).
- Los productos se organizan según las plantas medicinales que lo componen, la presentación, precio venta al público, cantidad en existencias, el laboratorio que las realiza y su acción terapéutica (diurético, analgésico…).
- Se podrán consultar productos por una planta medicinal determinada, por presentación, productos elaborados por un laboratorio, etc.

3. Diseñar una base de datos que cumpla las siguientes premisas:

- La empresa de limpieza "Limpiadores S.A." desea crear una base de datos para su gestión.
- La empresa tiene limpiadores, de los que se guarda su nombre, NIF, domicilio y teléfono.
- La empresa tiene clientes a los que presta servicios. Para cada cliente, se almacena su NIF, nombre, actividad y domicilio. Para cada cliente, se anotan los servicios contratados, con el código de servicio, nombre de servicio y la periodicidad de cada uno. Queremos saber cuál de los empleados realiza cada servicio, y la hora y el día en las que hace dicho servicio.
- También guardamos las facturas mensuales de cada cliente, con número de factura, importe, fecha y estado de cobro.

4. Diseñar una base de datos que cumpla las siguientes premisas:

- Una compañía aérea desea informatizar la información relativa a sus vuelos. La información con la que se cuenta es la siguiente.
- Los vuelos están identificados por un código único y tienen una fecha, un origen, un destino y una duración. Además, cada vuelo organiza sus asientos en categorías o clases (primera, turista…) de forma que en un mismo vuelo todos los asientos de una misma clase tienen el mismo precio.
- Cada vuelo se realiza en un avión del que se guarda su nombre, que es único, su modelo, capacidad y año de construcción.
- La tripulación de cada vuelo está formada por un piloto, un copiloto y personal auxiliar. De los pilotos se guarda su NIF, nombre, antigüedad y los distintos tipos de servicios que puede realizar en la compañía (nacional, continental, intercontinental). Del personal auxiliar se guarda NIF, nombre y categoría.

- Por último, se desea almacenar los datos de las reservas de cada vuelo, anotando el NIF o número de pasaporte de los pasajeros, su nombre, así como la fecha en que se realizó la reserva y el asiento que ocupa en el vuelo.

5. Diseñar una base de datos que cumpla las siguientes premisas:

Un club de tenis ha decidido crear una base de datos para facilitar su gestión. Por una parte, quiere gestionar las reservas de las pistas y, por otra, los cursos realizados dentro del club.

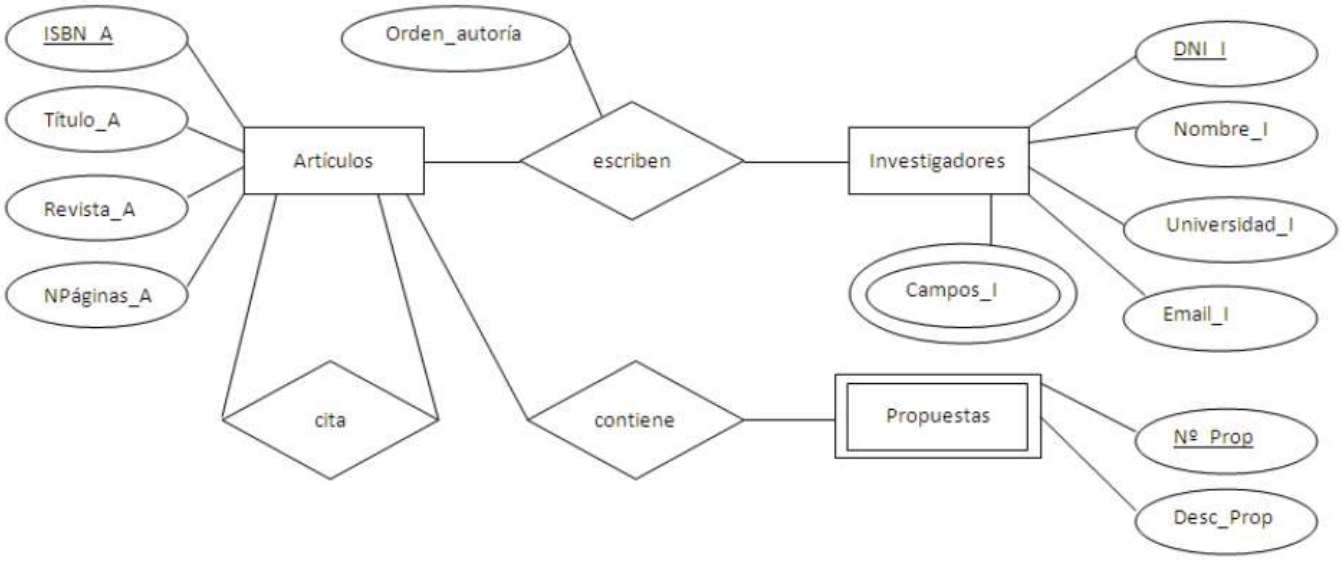
En cuanto a la parte de reservas se desea almacenar la pista en la que se realiza la reserva, la fecha de la reserva, la hora de entrada y la de salida de la pista, si se ha pagado o no, y si se ha hecho uso de la iluminación de la pista. De cada pista se almacenará un identificador, una descripción, si dispone de gradas o no, y si está homologada para campeonatos.

Las reservas se harán a nombre de los socios que harán uso de la pista, anotando cada uno de los socios que jugarán en la hora reservada. De cada socio se guarda el número de socio, NIF, nombre, apellidos, dirección, teléfono, número de cuenta corriente en el que realizar los cargos, y si es socio accionista o no.

Respecto a la parte de la gestión de los cursos, tienen como alumnos a los socios. De cada curso se guarda un identificador, el nombre del curso y el número máximo de alumnos que puede tener. Cada uno es impartido en, al menos, dos días de la semana y tiene un horario en función del día de la semana. Cada curso es impartido por un profesor, de los que guardamos el NIF, número de federación, nombre, apellidos, dirección y teléfono. Por último, también se quieren guardar los recibos mensuales de cada uno de los cursos de los que están matriculados los socios, indicando si está pagado el mes o no.

2.6 TEST DE CONOCIMIENTOS

Diagrama Entidad-Relación a utilizar en las consultas 1 – 8:



1 La cardinalidad de la relación *escriben* debe ser:

a) 1:M porque un artículo lo escribe un solo investigador.
b) M:M porque un artículo puede tener coautores y un investigador puede escribir muchos artículos.
c) M:1 porque un investigador solo puede tener un orden de autoría cuando escribe un artículo con otros investigadores.
d) Todas las respuestas son correctas.

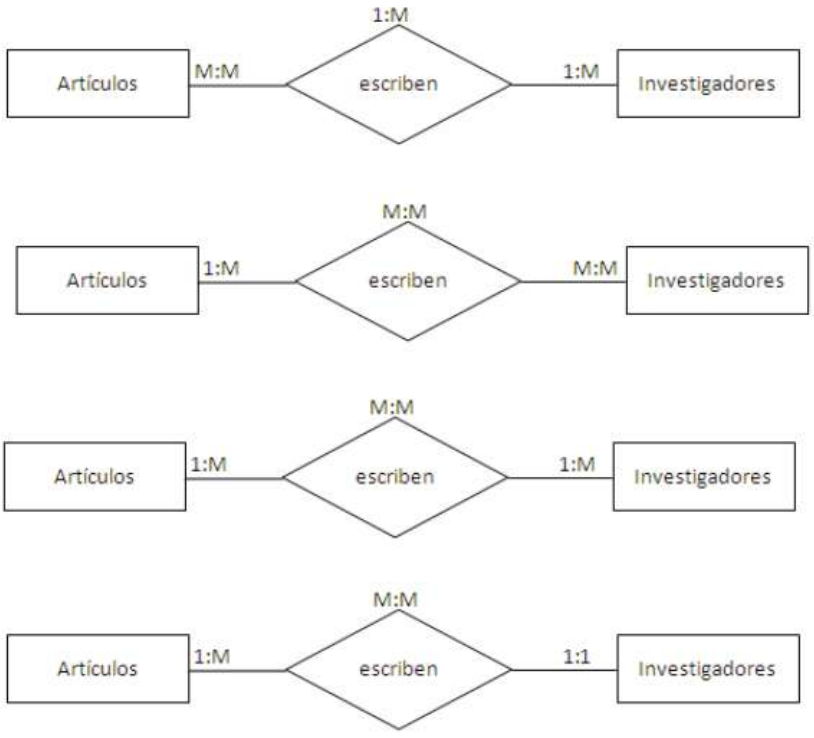
2 El atributo *Campos_I*:

a) Es un atributo descriptivo.
b) Es una agregación.
c) Es un atributo multivaluado.
d) Equivale a una entidad débil relacionada con la entidad *Investigador*.

3 El atributo *Orden_autoría*:

a) Es un atributo descriptivo.
b) Es una agregación.
c) Es un atributo multivaluado.
d) Equivale a una entidad débil relacionada con la entidad *Investigador*.

4 La cardinalidad extendida de la relación *escriben* es:



5 La relación *cita*:

a) Es una relación binaria que asocia el artículo que cita con el/los artículos que son citados.
b) Es una relación ternaria.
c) Es una relación reflexiva, ya que las entidades relacionadas son las mismas.
d) Es una relación reflexiva, ya que los atributos relacionados son los mismos.

6 La cardinalidad de la relación *cita* es:

a) M:M:M
b) M:M
c) 1:M
d) M

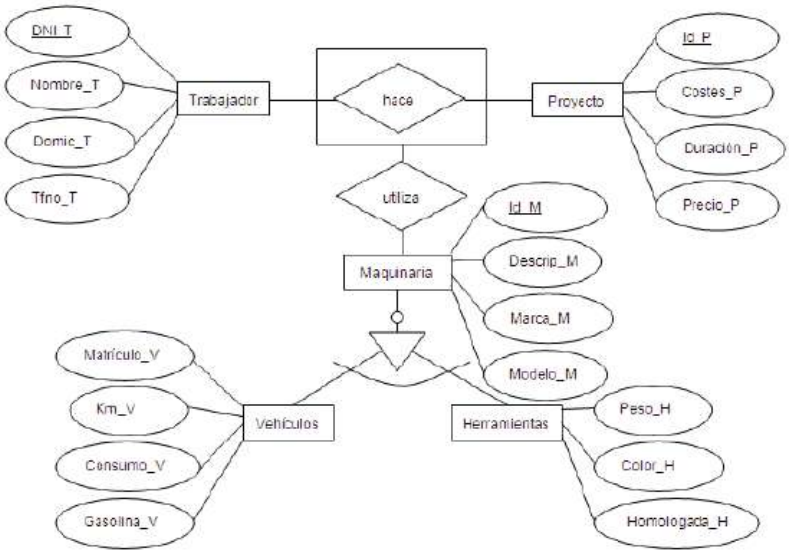
7 Señala la afirmación correcta:

a) *Propuestas* es una entidad débil porque requiere una clave foránea junto al discriminante *Nº_Prop.*
b) *Propuestas* es una entidad débil y se relaciona con la entidad *Artículos* mediante una relación de cardinalidad 1:M.
c) La relación entre *Propuestas* y *Artículos* tiene cardinalidad M:M porque un artículo puede tener muchas propuestas y una propuesta puede estar en muchos artículos.
d) La clave primaria de *Propuestas* es *Nº_Prop*.

8 Señala la afirmación incorrecta:

a) *Orden_autoría* simboliza que un investigador escribe muchos artículos y en cada artículo puede tener un orden de autoría distinto.
b) *Campos_I* simboliza que un investigador puede trabajar en más de un campo de la ciencia.
c) Cada propuesta se identifica a través del *ISBN_A* del artículo donde se publicó, junto al número de orden de las propuestas de ese artículo.
d) La relación *cita* simboliza la cita que cada autor recibe en sus artículos.

Diagrama Entidad-Relación a utilizar en las consultas 9 – 15:



9 La generalización es:

a) Parcial y con solapamiento
b) Total y con solapamiento
c) Parcial y exclusiva
d) Total y exclusiva

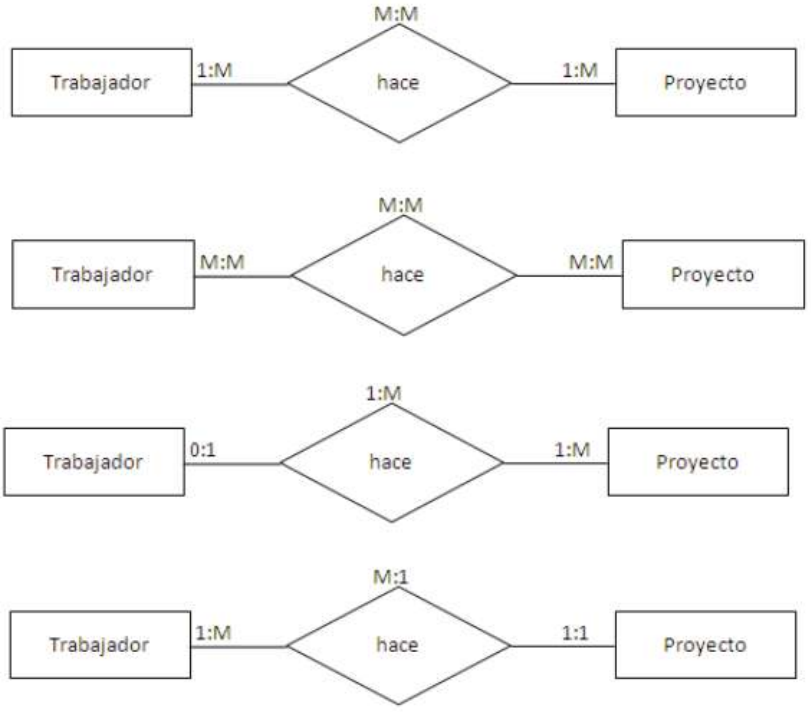
10 El significado de la generalización es:

a) La maquinaria debe ser exclusivamente vehículos o herramientas, no pudiendo ser de otro tipo.
b) No tiene sentido utilizar una generalización porque vehículos y herramientas son cosas diferentes.
c) El modelo Entidad-Relación representa los vehículos y herramientas que cada trabajador usa para cada uno de sus proyectos.
d) (a) y (c) son ciertas.

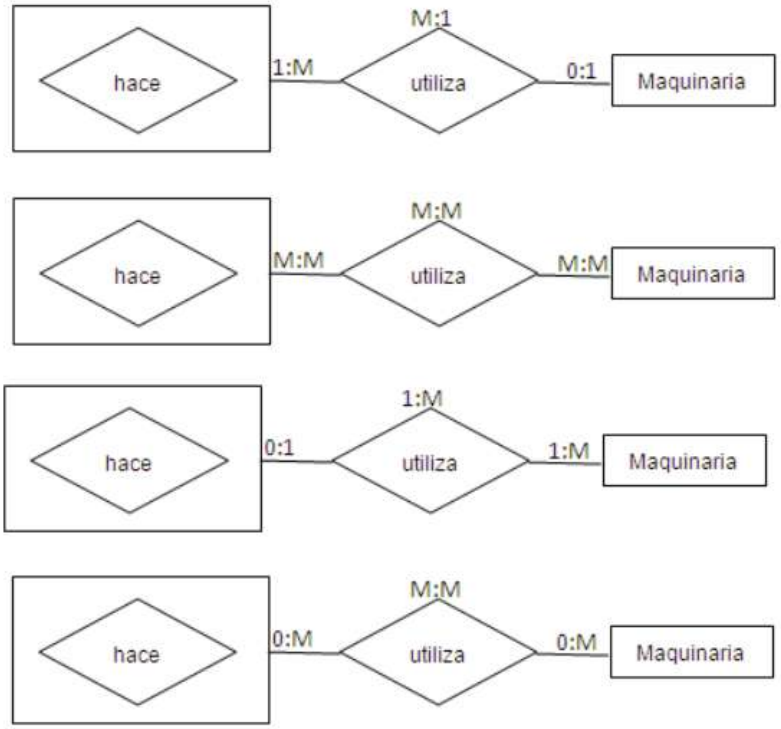
11 ¿Cuál es la clave primaria de la entidad *Herramientas*?

a) Peso_H
b) *Color_H*
c) *DNI_T* e *Id_P*
d) *Id_M*

12 La cardinalidad de la relación *hace* es:



13 La cardinalidad de la relación *utiliza* es:



14 El motivo de utilizar una agregación es:

a) La *Maquinaria* está relacionada con el *Trabajador* que realiza un *Proyecto* y será única, es decir, un *Trabajador*, al efectuar un *Proyecto*, solo podrá utilizar una máquina.

b) La *Maquinaria* está relacionada con el *Trabajador* que realizar un *Proyecto* y no será única, es decir, un *Trabajador*, al efectuar un *Proyecto*, podrá utilizar varias máquinas.

c) La *Maquinaria* depende únicamente del *Proyecto*.

d) La *Maquinaria* depende únicamente del *Trabajador*.

NOTA

Esta cuestión pertenece a un ejercicio de las **Oposiciones al Cuerpo de Técnicos Auxiliares de Informática de la Administración del Estado (Convocatoria de 2015)**.

15 Dentro de un modelo Entidad-Relación, el número de tipos de entidad que participan en un tipo de relación se denomina:

a) Participación

b) Grado

c) Agregación

d) Cardinalidad

3 El modelo relacional

3.1 EVOLUCIÓN DEL MODELO RELACIONAL

Un **modelo de datos** es una colección de conceptos, normas y acuerdos que permitirán representar los hechos que queremos plasmar y la posterior manipulación de los datos descritos.

Todo modelo, sea el que sea, se construye teniendo que diferenciar en él dos partes:

- **Estática:** es la parte no modificable en el tiempo e implica a las estructuras de datos.
- **Dinámica:** es la parte susceptible de modificación, tras la aplicación de operaciones sobre dichas estructuras de datos.

Edgar Frank Codd definió las bases del **modelo de datos relacional** a finales de los años 60. Aunque trabajaba para IBM, la empresa tardó unos pocos años en implementar sus bases. Dicho modelo se empezó a implementar cada vez más, hasta ser el modelo de bases de datos más popular.

RECUERDA

En el apartado 1.1. "Evolución histórica de las bases de datos" pueden ampliar esta información.

En las bases de Codd, se definían los objetivos de este modelo:

- **Independencia física:** la forma de almacenar los datos, no debe influir en su manipulación lógica.
- **Independencia lógica:** las aplicaciones que utilizan la base de datos no deben ser modificadas porque se modifiquen elementos de la base de datos.
- **Flexibilidad:** la base de datos ofrece fácilmente distintas vistas en función de los usuarios y aplicaciones.
- **Uniformidad:** las estructuras lógicas siempre tienen una única forma conceptual (Ias tablas).
- **Sencillez.**

Ya que hemos tratado la evolución del modelo relacional en la evolución histórica de las bases de datos, de manera esquemática, comentaremos los ítems cronológicos más destacados de este modelo:

Año	Acontecimiento destacado
1970	Codd publica las bases del modelo de datos relacional
1971-72	Primeros desarrollos teóricos del modelo de datos relacional
1973-78	Primeros prototipos de base de datos relacional: System R de IBM
1974	La Universidad de Berkeley desarrolla *Ingres*, un DBMS relacional basado en cálculo relacional. Emplea el lenguaje QUEL
1978	Aparece el lenguaje QBE (*Query By Example*), lenguaje de acceso relacional a los archivos VSAM de IBM
1979	Aparece Oracle, el primer DBMS comercial relacional, que se convertirá en líder del mercado (emplea SQL) Codd revisa su modelo relacional y lanza el modelo RM/T como un intento de subsanar sus deficiencias

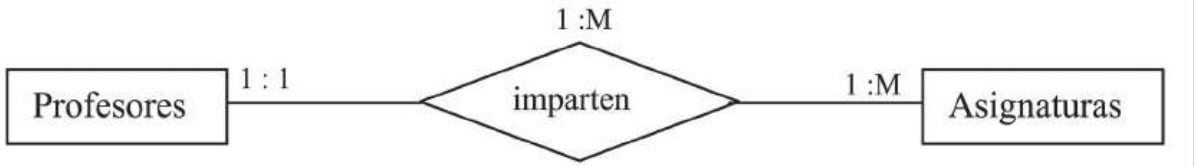
Año	Acontecimiento destacado
1981	Surge Informix como DBMS relacional para Unix Aparece SQL, que será el estándar para la creación de las bases de datos relacionales
1983	Se distribuye DB2, el sistema gestor de bases de datos relacionales de IBM
1984	Aparece la base de datos Sybase que llegó a ser la segunda más popular (tras Oracle)
1986	ANSI normaliza el SQL: SQL/ANSI.
1987	ISO también normaliza SQL: SQL ISO (9075)
1988	La versión 6 de Oracle incorpora el lenguaje procedimental PL/SQL
1989	ISO revisa el estándar y publica el estándar SQL Addendum Microsoft y Sybase desarrollan SQL Server para el sistema operativo OS/2 de Microsoft e IBM. Durante años Sybase y SQL Server fueron el mismo producto
1990	Versión dos del modelo relacional (RM/V2) realizada por Codd Surge la propuesta de añadir, al modelo relacional, operativas orientadas a objetos
1992	ISO publica el estándar SQL 92 (todavía sigue siendo el más extendido)
1995	Aparece MySQL, una base de datos relacional de código abierto con licencia GNU que se hace muy popular entre los desarrolladores de páginas web
1996	ANSI normaliza el lenguaje procedimental basado en SQL: SQL/PSM. Permite técnicas propias de los lenguajes de programación estructurada
1999	ISO publica un nuevo estándar que incluye características más avanzadas: SQL 99
2003	ISO publica el estándar SQL 2003. En él se añade SQL/PSM al estándar
2006	Estándar ISO: SQL:2006
2008	Estándar ISO: SQL:2008
2011	Estándar ISO: SQL:2011
2012	Estándar ISO: SQL:2012

3.2 ESTRUCTURA DEL MODELO RELACIONAL

Antes de comenzar con la definición de los elementos que componen el modelo relacional, a modo de nexo de unión con el tema anterior, vamos a explicar el procedimiento para pasar el diseño conceptual dado por el modelo Entidad-Relación al modelo relacional.

1. **Paso a tablas de entidades fuertes:**
   - Nombre de la tabla = Nombre de la entidad
   - Campos de la tabla = Atributos de la entidad
   - Clave primaria = Identificadores primarios de la entidad
2. **Paso a tablas de entidades débiles:**
   - Nombre de la tabla = Nombre de la entidad
   - Campos de la tabla = Atributos de la entidad + clave primaria de la entidad de la que depende
   - Clave primaria = Discriminante (identificador débil de la entidad débil) + identificador primario de la entidad de la que depende

3. **Paso a tablas de relaciones:**
   - Nombre de la tabla = Nombre de la relación
   - Campos de la tabla = Identificadores primarios de las entidades relacionadas + posibles atributos descriptivos
   - Clave primaria = Clave compuesta, como mínimo, de los identificadores principales de las entidades relacionadas
   - Excepción: Salvo las relaciones de cardinalidad M:M, en el resto de casos, las relaciones no forman tabla independiente, sino que alteran levemente al menos una de las entidades relacionadas. Estas son las cardinalidades que no pasan a tabla:
     - **Cardinalidad 1:M o M:1:** en este caso, el identificador primario de la entidad "1" se incluye como *clave externa* en la nueva tabla correspondiente a la entidad "M" (lo mismo sucede con los atributos descriptivos). Veamos un ejemplo de este paso a tablas[8]:

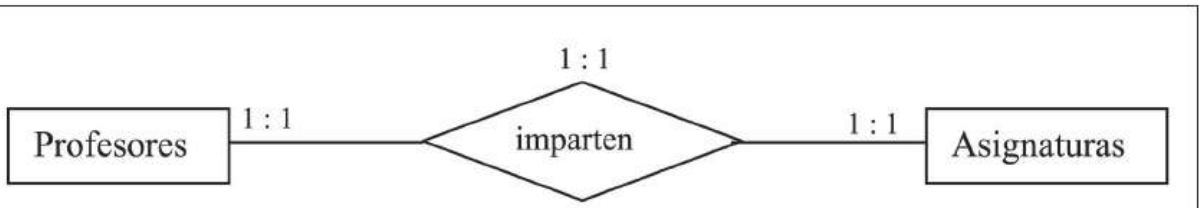


Asignaturas(Cód_As, Nombre_As, Créditos_As, Facultad_As, DNI_P)

Profesores(DNI_P, Nombre_P, Apellidos_P, NCuenta_P, Teléfono_P)

Figura 3.1. *Ejemplo de paso a tablas de una relación 1:M*

     - **Cardinalidad 1:1:** para las relaciones en la que ambas entidades tienen una relación de cardinalidad "1", la alternativa más generalizada consiste en colocar como *clave externa* el identificador primaria de una de las entidades en la nueva tabla correspondiente a la otra entidad (da igual qué identificador se escoja). Veamos, al igual que antes, un ejemplo:



Asignaturas(Cód_As, Nombre_As, Créditos_As, Facultad_As)

Profesores(DNI_P, Nombre_P, Apellidos_P, NCuenta_P, Teléfono_P, Cód_As)

Figura 3.2. *Ejemplo de paso a tablas de una relación 1:1*

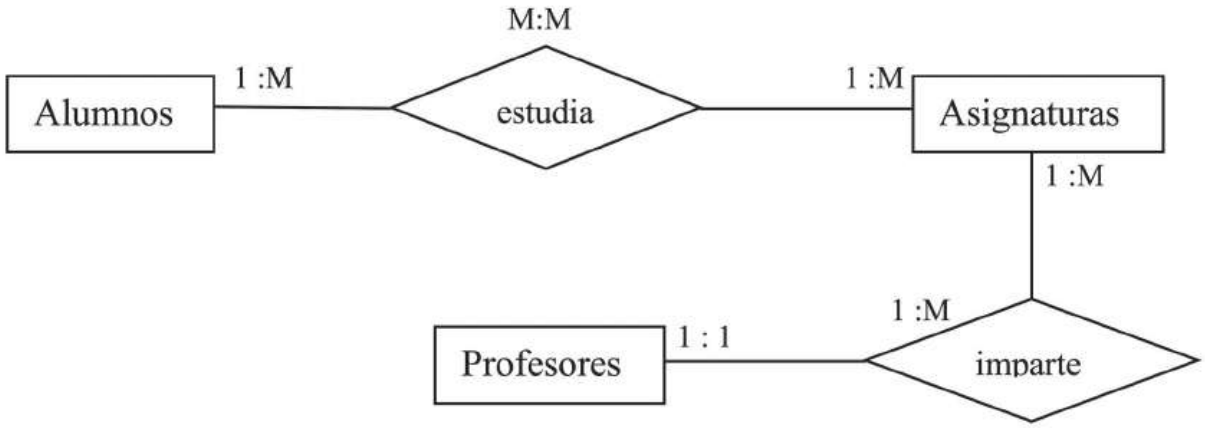
8 Nótese que en estos ejemplos los atributos de las entidades están omitidos.

NOTA

Otra alternativa para pasar a tablas relaciones 1:1 consiste en generar una única tabla con todos los atributos de ambas entidades, colocando como clave principal, cualquiera de las claves de las dos entidades. De este modo, la otra clave será clave alternativa. El nombre de la tabla sería el de la entidad más importante en el modelo conceptual.
Nosotros no seguiremos este procedimiento sino el anteriormente comentado.

EJEMPLO 3.1

Si representamos la base de datos de un centro de formación privado mediante su diagrama Entidad-Relación, este sería:



Y, por tanto, sus tablas serían las siguientes:

- **Alumnos** (DNI_Al, Nombre_Al, Apellidos_Al, Edad_Al, Teléfono_Al, Dirección_Al, Ciudad_Al)
- **Estudian** (DNI_Al, Cód_As, Nota_Al_As, Convocatoria_Al_As)
- **Asignaturas** (Cód_As, Nombre_As, Créditos_As, Facultad_As, DNI_P)
- **Profesores** (DNI_P, Nombre_P, Apellidos_P, NCuenta_P, Teléfono_P)

El modelo de Entidad-Relación extendido no presenta demasiadas novedades para el proceso de paso a tablas, sino que se pueden aplicar las mismas reglas antes comentadas.

Puede que la única excepción que es interesante comentar sea el caso de la generalización/especialización.

Para ello, tomaremos como muestra la generalización de la Figura 2.12, a la cual le añadiremos diversos atributos. Es de interés resaltar que los atributos *DNI_P* y *Nombre_P* como son compartidos por *Alumnos* y *Profesores*, han sido situados en la generalización *Personas*.

Por el contrario, como la *Edad_A* solo es relevante para la entidad *Alumnos*, y, por su parte, el *NCuenta_Pr* solo interesa a la entidad *Profesores*, se han añadido a sus correspondientes entidades.

Igualmente sucede con las relaciones, es decir, como cada subclase se relaciona con la entidad *Asignaturas* de un modo diferente, se han establecidos distintas relaciones con dichas subclases *Alumnos* y *Profesores*. Si, por el contrario, existiera una misma relación, con idéntico comportamiento para ambas subclases, la relación la asumiría la superclase *Personas*.

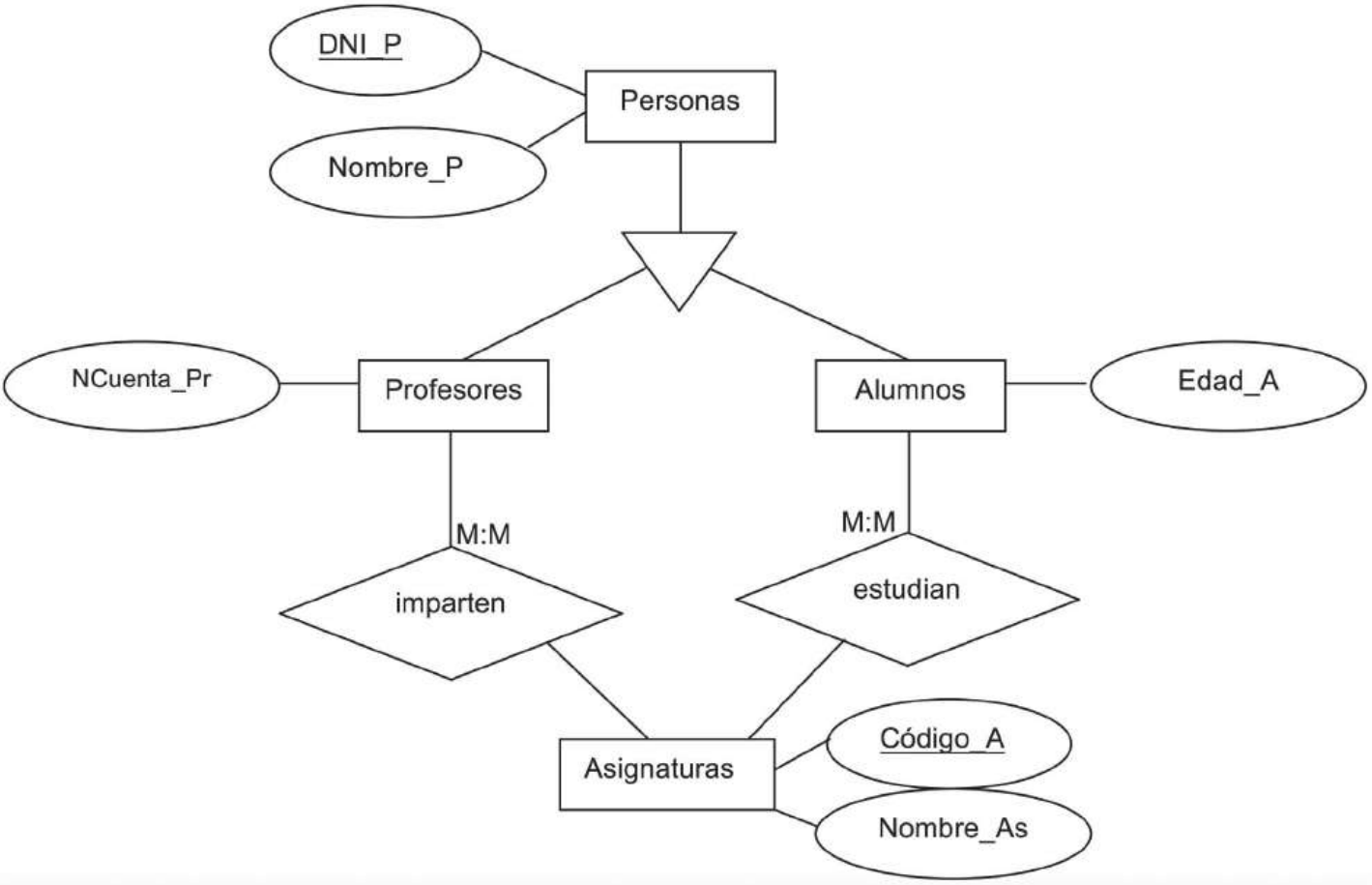


Figura 3.3. *Figura 2.12 con la inclusión de atributos para su paso a tablas*

De este modo, ante el modelo E-R de la Figura 3.3, el correspondiente paso a tablas sería el siguiente:

Personas (DNI_P, Nombre_P)

Alumnos (DNI_P, Edad_Al)

Profesores (DNI_P, NCuenta_Pr)

Estudian (DNI_P, Cód_As)

Imparten (DNI_P, Cód_As)

Asignaturas (Cód_As, Nombre_As)

3.2.1 EL CONCEPTO DE RELACIÓN. PROPIEDADES DE LAS RELACIONES

Las bases de datos relacionales se basan en el uso de **tablas** (también llamadas **relaciones**). Las tablas se representan gráficamente como una estructura rectangular formada por filas y columnas. Cada columna almacena información sobre una **propiedad** determinada de la tabla (se le llama también atributo, denominación heredada del modelo E-R, o campo de la tabla), mientras que cada fila posee una ocurrencia o ejemplar de la instancia representada en la tabla (a las filas se las llama también tuplas).

Tabla "Alumnos"

DNI_Al	Nombre_Al	Apellidos_Al	Edad_Al	Teléfono_Al	Dirección_Al	Ciudad_Al	
11111111A	Fran	Martínez	15	950000000	C/ Fran	Almería	← tupla 1
22222222B	Amalia	Gallegos	16	950111111	C/ Amalia	Almería	← tupla 2
...	...	...	...	...	...	...	
33333333C	Juan	Pérez	18	954222222	C/ Juan	Sevilla	← tupla N

Figura 3.4. *Representación de una tabla en el modelo relacional. Véase las columnas, denominadas como "DNI_Al" (clave primaria), "Nombre_Al", "Apellidos_Al", etc., y cada casilla corresponde con un determinado valor para esa tupla, en esa columna*

Las principales propiedades de las relaciones son:

- **Unicidad de nombre:** cada tabla o relación debe tener un nombre distinto al de las demás.
- **Atributos con dominios atómicos:** los dominios de los distintos atributos de una tabla no pueden tener dominios que a su vez sean subconjuntos, es decir, para cada intersección de fila y columna (o tupla) solo puede haber un valor.
- **Columnas con nombres distintos:** cada columna o atributo de una tabla debe tener un nombre único y distinto a las demás. Además, también es aconsejable que no se repitan nombres de atributos entre las distintas columnas que conforman la base de datos.

  Para ello, en este libro proponemos la siguiente notación para identificar un atributo:

  NombreAtributo_NombreTabla → Ejemplo: Nombre_Al

- Los atributos en una relación no guardan un orden concreto.
- Dos tuplas de una relación no pueden tener el mismo valor de la clave primaria.
- Las tuplas en una relación no guardan un orden concreto, aunque pueden estar indexadas.

3.2.2 ATRIBUTOS Y DOMINIO DE LOS ATRIBUTOS

Como acabamos de comentar en el apartado anterior, los **atributos**, también llamados **propiedades, columnas** o **campos**, son los valores que definen las tuplas de las que consta una relación, es decir, la información o datos que se almacenan de cada individuo.

Las restricciones de dominio suponen una gran mejora en este modelo, ya que permiten especificar los posibles valores válidos para un atributo. Cada dominio incorpora su nombre y una definición del mismo.

Algunos ejemplos de atributos y dominios son:

- Dirección: cinco caracteres.
- Nacionalidad: española, francesa, italiana, etc.

3.2.3 TUPLA, GRADO Y CARDINALIDAD

Como hablábamos en el aparado 3.2.1, una **tupla** corresponde a cada fila de la tabla o relación. Este concepto está relacionado con la idea de registro, de modelos anteriormente comentados.

Una tupla debe cumplir las siguientes propiedades:

- Cada tupla se debe corresponder con un elemento del mundo real.
- No puede haber dos tuplas iguales (con todos los valores iguales).

Por su parte, el término **grado** indica el tamaño de una relación, en base al número de columnas o atributos que contiene.

Y, por último, la **cardinalidad** de una relación es el número de tuplas que contiene, o número de filas de la tabla.

3.2.4 RELACIONES Y TABLAS

Una relación o tabla está formada por los siguientes elementos:

- **Nombre:** define la tabla y, como hemos dicho, debe ser único.
- **Cabecera:** nombre y dominio de cada columna o atributo.
- **Cuerpo:** conjunto de tuplas con sus correspondientes campos o atributos.
- **Esquema:** compuesto por el nombre de la tabla y la cabecera.
- **Estado:** compuesto por el esquema y el cuerpo.

Podemos clasificar los diferentes tipos de tablas en:

- **Persistentes.** Solo pueden ser borradas por los usuarios:
  - **Base:** son tablas independientes. Se crean indicando su estructura y sus ejemplares.
  - **Vistas:** son tablas que solo almacenan el resultado de una consulta sobre una/s tablas/, sobre vista/s o sobre tabla/s instantánea/s.

    Si los datos de origen cambian, los datos de la vista también cambiarán.
  - **Instantáneas:** son similares a las vistas (en cuanto a su forma de creación) pero, como diferencia, sí almacenan los datos que muestran. Solo se modifican sus valores si se orden que estos se "refresquen".
- **Temporales:** son tablas que son eliminadas automáticamente por el sistema al dejar de ser útiles. A su vez, se pueden subclasificar del mismo modo que las tablas persistentes.

3.3 CLAVES EN EL MODELO RELACIONAL

En el modelo relacional podemos distinguir los siguientes tipos de claves.

3.3.1 CLAVES CANDIDATAS

Podemos definir las **claves candidatas** como el conjunto de atributos seleccionados para identificar las tuplas de forma unívoca y mínima (con menos atributos posible) en una relación.

Si atendemos al ejemplo expuesto en la Figura 3.4, podemos ver que el campo *DNI_Al* posee un valor distinto para cada una de las tuplas de la relación *Alumnos*, y, por tal motivo, ese campo puede servir para discernir un *Alumno* de otro (identificar una tupla de otra en la relación). Por tanto, *DNI_Al* es una clave candidata para la relación *Alumnos*.

Otras posibles claves candidatas para esta misma relación podrían ser:

- {Nombre_Al, Apellidos_Al, Teléfono_Al}
- {Teléfono_Al, Dirección_Al}
- {Nombre_Al, Apellidos_Al, Dirección_Al}

3.3.2 CLAVES PRIMARIAS

Las **claves primarias** son aquellas claves candidatas que el diseñado de la base de datos ha escogido (porque considera más óptimas) para identificar las tuplas de una relación. Por supuesto, en caso de que una relación posee una única clave candidata, esta será la clave primaria; si, por el contrario, la relación ofrece varias claves candidatas, el diseñador de la base de datos tendrá que tomar tal decisión.

En el ejemplo de la Figura 3.5 hemos considerado oportuno escoger como clave primaria el atributo *DNI_Al*, en detrimento de las otras claves candidatas que pudieran ser propuestas.

Nótese que, tal y como hemos observado en las figuras de este tema, la notación que vamos a emplear para especificar las claves primarias consiste en subrayar el atributo o atributos que componen dicha clave.

3.3.3 CLAVES ALTERNATIVAS

Son **claves alternativas** son las claves que, siendo candidatas, no han sido seleccionadas por el diseñador de la base de datos como clave primaria.

Así, en el ejemplo que estamos desarrollando en este punto, las claves alternativas de la relación *Alumnos* serían las siguientes:

- {Nombre_Al, Apellidos_Al, Teléfono_Al}
- {Teléfono_Al, Dirección_Al}
- {Nombre_Al, Apellidos_Al, Dirección_Al}

Ya que se escogió, como clave primaria, el *DNI_Al*.

3.3.4 CLAVES AJENAS

Las **claves ajenas** son fundamentales para que el modelo de datos relacional pueda reflejar las asociaciones entre las distintas tablas de la base de datos, tal y como hemos podido estudiar en el paso del modelo Entidad-Relación al modelo relacional, visto en la sección 3.2.

Por tanto, la clave ajena es un conjunto no vacío de campos de una tabla cuyos valores deben coincidir con los valores de la clave primaria de la otra tabla con la que se esté asociando.

De esta forma, si recordamos el Ejemplo 3.1, observaremos en rojo las claves ajenas correspondientes:

Alumnos (DNI_Al, Nombre_Al, Apellidos_Al, Edad_Al, Teléfono_Al, Dirección_Al, Ciudad_Al)

Estudian (DNI_Al, Cód_As, Nota_Al_As, Convocatoria_Al_As)

Asignaturas (Cód_As, Nombre_As, Créditos_As, Facultad_As, DNI_P)

Profesores (DNI_P, Nombre_P, Apellidos_P, NCuenta_P, Teléfono_P)

3.4 RESTRICCIONES DE INTEGRIDAD

Tal y como comentábamos en la sección 2.3., las restricciones de integridad son una serie de condiciones de obligado cumplimiento por los datos de las bases de datos. Las hay de dos tipos:

- **Restricciones inherentes:** aquellas que no son determinadas por los usuarios, sino definidas por el hecho de que la base de datos sea relacional.
- **Restricciones explícitas**, también llamadas **del usuario** o **semánticas**: aquellas que son incorporadas por los usuarios. Algunas de las más extendidas en la literatura y en la práctica, son:
  - **Clave primaria:** hace que los atributos marcados como *clave primaria* no pueden repetir valores.
  - **Unicidad:** impide que los valores de los atributos marcados de esa forma puedan repetirse.
  - **Obligatoriedad:** prohíbe que el atributo marcado de esta forma no tenga ningún valor (valor *"Null"*).
  - **Integridad referencial:** prohíbe colocar valores en una clave externa que no estén reflejados en la tabla donde ese atributo sea clave primaria.
  - **Regla de validación:** condición que debe cumplir un dato concreto para que sea actualizado.

En esta sección nos centraremos en ampliar la explicación sobre alguna de las restricciones y elementos del modelo.

3.4.1 VALOR "NULL" EN EL MODELO

En ocasiones, no se conoce el valor de un atributo para una determinada tupla. En esos casos, a ese atributo de esa tupla se le asigna un valor nulo (valor *Null*), que indica que el valor de ese atributo es desconocido o, simplemente, que ese atributo no es aplicable a esa tupla.

Por ejemplo, si tenemos una relación de vehículos en la que podemos guardar tanto motos como automóviles, un atributo que indique a qué lado está el volante (para distinguir vehículos con el volante a la izquierda de los que lo tienen a la derecha), carece de sentido en motos. En ese caso, ese atributo para entidades de tipo motos será *Null*.

En otro caso, al crear una nueva columna en una relación, ya existente y con una información ya almacenada, los valores a los que se inicializa dicha columna son con el valor *Null*.

Al igual que con otros valores, también es posible operar con el valor *Null*, siguiendo las siguientes tablas de verdad:

p	q	p AND q	p OR q	NOT p
Verdadero	**Verdadero**	Verdadero	Verdadero	Falso
Verdadero	**Falso**	Falso	Verdadero	
Verdadero	**Null**	Null	Verdadero	
Falso	**Verdadero**	Falso	Verdadero	Verdadero
Falso	**Falso**	Falso	Falso	
Falso	**Null**	Falso	Null	
Null	**Verdadero**	Null	Verdadero	Null
Null	**Falso**	Falso	Null	
Null	**Null**	Null	Null	

Como ya hemos visto, además de especificar restricciones semánticas, se establecen las restricciones propias del modelo relacional con respecto a los valores *Null*, concretamente las siguientes:

- Las claves candidatas no pueden poseer valores *Null*.
- Una clave extranjera (*foreign key*) no puede poseer valores *Null*.

3.4.2 INTEGRIDAD DE LAS ENTIDADES

Mediante esta integridad inherente al modelo relacional, los atributos marcados como clave primaria no pueden repetir valores. Además, como acabamos de comentar anteriormente, esta integridad obliga a que esos atributos no puedan estar vacíos (con valor *Null)*. De hecho, si la clave primaria consta de varios atributos, ninguno de ellos podrá ser *Null*.

3.4.3 INTEGRIDAD REFERENCIAL

Se emplea con las claves externas o *foreign key*. Mediante esta integridad, no se podrán introducir valores en claves externas que no estén incluidos en los campos relacionados con esa clave (en las claves primarias correspondientes).

Es decir, si tomamos nuevamente el Ejemplo 3.1, no se podrá dar de alta una nueva *Asignatura* introduciendo un valor para el campo *DNI_P*, sin que ese *Profesor* haya sido dado de alta previamente en su correspondiente tupla.

Además de los problemas que se plantean en las inserciones de datos, principalmente esta integridad referencial presenta problemas en las operaciones de borrado y actualización de tuplas, para lo cual se proponen diversas soluciones en la base de datos ante problemas de este tipo de integridad:

- Operación prohibida (*no action*).
- Transmitir la operación en cascada (*cascade*). Esto supone que si se modifica o borra un *Profesor*, también se modificará o borrarán las *Asignaturas* de dicho *Profesor*.
- Colocar valores *Null* (*set null*). En este caso, las referencias a un *DNI_P* borrado en la relación *Profesores* haría que se asignatura el valor *null* a la clave extranjera *DNI_P* de la tabla *Asignaturas*.
- Usando el valor por defecto (*default*). Esta solución sería similar a la anterior, salvo que en lugar de almacenar un valor *Null*, se almacenaría un valor por defecto.

3.5 TEORÍA DE LA NORMALIZACIÓN

La teoría de la normalización, iniciada por el propio Edgar Frank Codd junto a otros autores entre los que podemos destacar a Raymond F. Boyce o Ronald Fagin, es una técnica que consiste en aplicar una serie de reglas a las relaciones obtenidas tras el paso a tablas del modelo Entidad-Relación.

El proceso de normalización se inicia con las tres primera formas normales (Primera Forma Normal - 1FN, Segunda Forma Normal - 2FN y Tercera Forma Normal - 3FN), definidas por Codd, pero, posteriormente, se detectaron unas anomalías, por lo que surgiría la Forma Normal de Boyce-Codd - FNBC. Después Fagin definiría la Cuarta Forma Normal - 4FN y la Quinta Forma Normal - 5FN.

Los objetivos concretos que se persiguen en la normalización de una base de datos se pueden sintetizar en los siguientes:

- Minimizar la redundancia.
- Minimizar el mantenimiento de la base de datos.
- Disminuir problemas de actualización de los datos en la base de datos.

INFORMACIÓN

Si quieres saber más sobre Ronald Fagin (en la foto) puedes consultar su biografía y enlaces a sus aportaciones a la ciencia en Wikipedia:

https://en.wikipedia.org/wiki/Ronald_Fagin

También puedes consultar la biografía de Raymond F. Boyce:

https://en.wikipedia.org/wiki/Raymond_F._Boyce

3.5.1 EL PROCESO DE NORMALIZACIÓN. TIPOS DE DEPENDENCIAS FUNCIONALES

En primer lugar definiremos el concepto de **dependencia funcional** empleando la propia definición de Codd. De este modo, Codd explicaba que *"Dados dos atributos A y B de una relación R, se dice que B es funcionalmente dependiente de A, si para cada valor de A existe un valor de B, y solo uno, asociado con él"*.

Esto se denota como A → B.

A esta definición anterior podemos añadir que, tanto A como B, no tienen que ser un solo atributo, sino que puede ser un conjunto de dichos atributos. Así, al conjunto A del que depende funcionalmente el conjunto B se le denomina **determinante**, mientras que al conjunto B se le conoce como **implicado**.

Si continuamos con el Ejemplo 3.1, podemos afirmar que todos los atributos de la relación *Alumnos* dependen funcionalmente del valor de *DNI_Al*.

Continuando con las dependencias funcionales, podemos clasificarlas en diferentes tipos. Los más extendidos en la literatura son:

- **Dependencia Funcional Completa:** se produce cuando un atributo B, de una relación R, depende funcionalmente de otro atributo A, también de R, pero no tiene dependencia funcional de ningún subconjunto de A.

  Se denota como A ⇒ B.

EJEMPLO 3.2

Vamos a plantear dos ejemplos sobre la relación *Alumnos* del Ejemplo 3.1, uno que cumple la condición de dependencia completa y otro que no lo hace.

- El atributo *Edad_Al* tiene dependencia funcional completa con el atributo *DNI_Al* (*DNI_Al* ⇒ *Edad_Al*), ya que

  $$DNI\_Al \rightarrow Edad\_Al,$$

  Y, además, *DNI_Al* no puede dividirse en un subconjunto del que también dependa funcionalmente el atributo *Edad_Al*.

- El atributo *Edad_Al* no tiene dependencia funcional completa con el conjunto de atributos *{DNI_Al, Nombre_Al, Apellidos_Al}*, ya que, aunque

  $$\{DNI\_Al, Nombre\_Al, Apellidos\_Al\} \rightarrow Edad\_Al,$$

  Ese conjunto se puede dividir en el subconjunto *{DNI_Al}* que cumple que

  $$\{DNI\_Al\} \rightarrow Edad\_Al.$$

- **Dependencia Funcional Transitiva:** en una relación existe una dependencia funcional transitiva (A → C) si, dados los atributos o conjuntos de atributos A, B y C, sucede que A → B y, además, B → C.

EJEMPLO 3.3

Si a la relación *Profesores*, del Ejemplo 3.1 le añadimos un nuevo atributo *Sucursal_P* que almacena el nombre de la sucursal donde ese docente tiene su *NCuenta_P*, tendríamos que

Profesores (DNI_P, Nombre_P, Apellidos_P, NCuenta_P, Teléfono_P, Sucursal_P)

DNI_P → NCuenta_P y, además, NCuenta_P → Sucursal_P.

Luego se produciría la dependencia funcional transitiva: DNI_P → Sucursal_P.

- **Dependencia Funcional Multivaluada:** sean A, B y C tres subconjuntos distintos de atributos de una relación R, se dice que B tiene una dependencia funcional multivaluada con A, que A multidetermina a B, o que B depende multivaluadamente de A, si cada valor de A tiene asignado un conjunto bien definido de valores de B, y, además, ese conjunto de valores es independiente de cualquier valor del conjunto de atributos C.

Se denota como A →→ B.

EJEMPLO 3.4

Continuando con el Ejemplo 3.1, supongamos que a la relación *Asignaturas* le añadimos el atributo *LibroTexto_As*.

Asignaturas (Cód_As, Nombre_As, Créditos_As, Facultad_As, DNI_P, LibroTexto_As)

Supongamos ahora que para cada asignatura quisiéramos reflejar que existen no uno, sino varios docentes que la van a impartir (un docente para teoría y otro para las prácticas, por ejemplo) y, del mismo modo, queremos reflejar que en cada asignatura se emplearía más de un libro de texto de referencia. La representación de esto podría ser algo así:

Cód_As	Nombre_As	Créditos_As	Facultad_As	DNI_P	LibroTexto_As
001	Bases de Datos	6	Informática	11111111A 22222222B	Libro1 Libro2
002	Redes	9	Arquitectura	22222222B 33333333C	Libro3 Libro4

Nótese que no hay ninguna dependencia entre el Profesor que imparta la Asignatura y el Libro de Texto, sino que cada Asignatura será impartida por dos Profesores y utilizará dos Libros de referencia.

Ante esta situación, tenemos que Cód_As determina valores múltiples de DNI_P y de LibroTexto_As, que, a su vez, son independientes entre sí, por lo que tenemos las siguientes dependencias funcionales multivaludadas:

Cód_As→→DNI_P

Cód_As→→LibroTexto_As

- **Dependencia Funcional de Combinación:** se dice que una relación R posee dependencia funcional de combinación sobre sus proyecciones sí.

  $R = R_1 * R_2 * R_3 * \ldots * R_N$

EJEMPLO 3.5

Supongamos que queremos gestionar con mayor detalle los *Libros de Texto* y, para ello, queremos almacenar una relación R, como la que vemos a continuación.

Distribuidor	LibroTexto_As	Editorial
Lacasadellibro	LibroAsignatura1	Rama
Lacasadellibro	LibroAsignatura2	Rama
Lacasadellibro	LibroAsignatura1	Papaninfo
Amazon	LibroAsignatura1	Rama

Si subdividimos la relación R en R_1, R_2 y R_3, obteniendo tres relaciones binarias como las que vemos a continuación:

Distribuidor	LibroTexto_As
Lacasadellibro	LibroAsignatura1
Lacasadellibro	LibroAsignatura2
Amazon	LibroAsignatura1

LibroTexto_As	Editorial
LibroAsignatura1	Rama
LibroAsignatura2	Rama
LibroAsignatura1	Papaninfo

Distribuidor	Editorial
Lacasadellibro	Rama
Lacasadellibro	Papaninfo
Amazon	Rama

Podemos comprobar que al proyectar R_1, R_2 y R_3, obtenemos la relación original y, por tanto, existe una dependencia funcional de combinación.

En primer lugar combinaremos R_1 y R_2:

Distribuidor	LibroTexto_As	Editorial
Lacasadellibro	LibroAsignatura1	Rama
Lacasadellibro	LibroAsignatura1	Papaninfo
Lacasadellibro	LibroAsignatura2	Rama
Amazon	LibroAsignatura1	Rama
Amazon	LibroAsignatura1	Papaninfo

Y, seguidamente, combinaremos R_3 a la anterior relación y observamos que resulta la relación R inicial.

Distribuidor	LibroTexto_As	Editorial
Lacasadellibro	LibroAsignatura1	Rama
Lacasadellibro	LibroAsignatura1	Papaninfo
Lacasadellibro	LibroAsignatura2	Rama
Amazon	LibroAsignatura1	Rama
Amazon	LibroAsignatura1	Papaninfo

3.5.2 PRIMERA FORMA NORMAL (1FN)

Una relación está en 1FN si sus atributos no tienen dominios que, a su vez, sean conjuntos. O, dicho de otra manera, para que una relación esté en 1FN todos sus atributos deben tener un solo valor (ser atómicos).

EJEMPLO 3.6

De Normalización: 1FN. Supongamos que necesitamos implementar una base de datos para gestionar los pedidos de productos informáticos que lleva a cabo una tienda con sus clientes. Inicialmente, los atributos se han agrupado en una única relación, que hemos denominado *Pedidos*:

Pedidos (Número_Pedido, Fecha_Pedido, Código_Producto, Descripción_Producto, Precio_Producto, Cantidad_Producto, Precio_Total_Pedido, Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

En primer lugar, en este ejemplo comprobaremos si la relación está en 1FN.

Pedidos (Número_Pedido, Fecha_Pedido, Código_Producto, Descripción_Producto, Precio_Producto, Cantidad_Producto, Precio_Total_Pedido, Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

Podemos concluir en un *Pedido* tiene un solo valor para los atributos *Número_Pedido, Fecha_Pedido* y *Precio_Total_Pedido*.

Del mismo modo, un *Pedido* es servido a un solo *Cliente*, luego solo existirá un valor para los atributos *Código_Cliente, Nombre_Cliente* y *Teléfono_Cliente.*

Sin embargo, los atributos con doble subrayado NO están en 1FN, ya que un *Pedido* consta de varios *Productos*, con más de un valor para los atributos seleccionados.

Por tanto, podemos afirmar que la relación NO está en 1FN.

Para conseguir que una relación esté en 1FN debemos excluir de la relación los atributos que no estén en 1FN, los cuales pasarán a almacenarse en una o varias nuevas relaciones. En esas relaciones también se almacenará la clave primaria de la relación inicial.

EJEMPLO 3.7

De Normalización: 1FN. En este ejemplo vamos a modificar la base de datos anterior para que esté en 1FN.

Pedidos (Número_Pedido, Fecha_Pedido, Código_Producto, Descripción_Producto, Precio_Producto, Cantidad_Producto, Precio_Total_Pedido, Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

Las modificaciones serían las siguientes:

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

Productos (Número_Pedido, Código_Producto, Descripción_Producto, Precio_Producto, Cantidad_Producto)

Y ahora, las dos relaciones resultantes, como podemos comprobar, sí están en 1FN.

NOTA

En el ejemplo anterior, podemos observar que la nueva relación *Productos* tienen como clave primaria los atributos *Código_Producto* y *Número_Pedido*.
Aunque esto sea así en el ejemplo, es necesario aclarar que **la clave de la relación inicial no tiene que pertenecer obligatoriamente a la clave primaria de la nueva relación**.

3.5.3 SEGUNDA FORMA NORMAL (2FN)

Una relación está en 2FN si está en 1FN y, además, cada atributo que no sea clave, depende de forma funcional completa a su clave primaria.

EJEMPLO 3.8

De Normalización: 2FN. En este ejemplo vamos a analizar las relaciones anteriores para comprobar si están en 2FN.

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

Productos (Número_Pedido, Código_Producto, Descripción_Producto, Precio_Producto, Cantidad_Producto)

Si analizamos las dos relaciones tenemos que:

- La relación *Pedidos* tiene como clave primaria un solo atributo. Por tanto, todos los atributos de la relación guardan dependencia funcional completa con dicha clave primaria y, de este modo, podemos afirmar que la relación está en 2FN.
- En la relación *Productos*, pese a que algunos atributos si cumplen con la dependencia funcional completa con respecto a la clave primaria:

$$\{Número\_Pedido, Código\_Producto\} \Rightarrow \{Precio\_Producto, Cantidad\_Producto\}$$

  El atributo con doble subrayado no cumple la dependencia funcional completa, ya que depende funcionalmente solo de un subconjunto de la clave primaria:

$$Código\_Producto \rightarrow Descripción\_Producto$$

Podemos concluir, por tanto, que la relación *Productos* NO está en 2FN y, de este modo, la base de datos no está en 2FN.

Para conseguir que una relación esté en 2FN debemos excluir de la misma los atributos que no estén en 2FN, los cuales pasarán a almacenarse, junto al subconjunto de clave primaria del que dependan funcionalmente, en otra relación.

EJEMPLO 3.9

De Normalización: 2FN. A continuación, aplicaremos los cambios comentados para que la base de datos esté en 2FN.

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

Productos (Número_Pedido, Código_Producto, Descripción_Producto, Precio_Producto, Cantidad_Producto)

Las modificaciones serían las siguientes:

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

Productos (Número_Pedido, Código_Producto,Precio_Producto, Cantidad_Producto)

Datos_Productos (Código_Producto, Descripción_Producto)

Y ahora, las tres relaciones resultantes, como podemos comprobar, sí están en 2FN.

3.5.4 TERCERA FORMA NORMAL (3FN)

Una relación está en 3FN si está en 2FN y, además, no tiene dependencias funcionales transitivas entre la clave primaria y sus atributos que no participen en una clave alternativa. O bien, dicho de otra forma, cuando no ocurre que un atributo dependa funcionalmente de otro/s atributo/s que no sea/s clave primaria.

EJEMPLO 3.10

De Normalización: 3FN. En este ejemplo vamos a analizar las relaciones anteriores para comprobar si están en 3FN.

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

Productos (Número_Pedido, Código_Producto, Precio_Producto, Cantidad_Producto)

Datos_Productos (Código_Producto, Descripción_Producto)

En este caso nos centraremos en la relación *Pedidos*, en la que observamos que:

Número_Pedido → Código_Cliente (A → B)
Código_Cliente → *{Nombre_Cliente, Teléfono_Cliente}* (B → C)

Luego existe una dependencia funcional transitiva

Número_Pedido → *{Nombre_Cliente, Teléfono_Cliente}* (A → C)

Podemos concluir, por tanto, que la relación *Pedidos* NO está en 3FN y, de este modo, la base de datos no está en 3FN.

Para conseguir que una relación esté en 3FN debemos excluir de la misma los atributos que hemos denotado como *C* y añadirlos a una nueva relación, junto al atributo o atributos denotados como *B* (dicho atributo *B* no se eliminan de la relación original). El atributo *B* será clave primaria de la nueva relación.

EJEMPLO 3.11

De Normalización: 3FN. A continuación, aplicaremos los cambios comentados para que la base de datos esté en 3FN.

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

Productos (Número_Pedido,Código_Producto, Precio_Producto, Cantidad_Producto)

Datos_Productos (Código_Producto, Descripción_Producto)

Las modificaciones serían las siguientes:

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente)

Productos (Número_Pedido, Código_Producto, Precio_Producto, Cantidad_Producto)

Datos_Productos (Código_Producto, Descripción_Producto)

Clientes (Código_Cliente, Nombre_Cliente, Teléfono_Cliente)

Y ahora, las cuatros relaciones resultantes, como podemos comprobar, sí están en 3FN.

Forma Normal de Boyce-Codd

Como hemos anticipado anteriormente, las tres formas normales fueron descritas por Codd, en 1970. No obstante, estas planteaban algunos problemas, así que cuatro años más tarde, Boyce y Codd redefinen la 3FN (de ahí el nombre de FNBC).

Una relación está en FNBC si cada determinante es una clave candidata.

EJEMPLO 3.12

De Normalización: FNBC. Para ejemplificar esta forma normal supondremos que a la base de datos anterior le añadimos los atributos que vemos en color rojo.

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente)

Productos (Número_Pedido, Código_Producto, Precio_Producto, Cantidad_Producto)

Datos_Productos (Código_Producto, Descripción_Producto)

Clientes (Código_Cliente, Nombre_Cliente, Teléfono_Cliente, Dirección_Cliente, Población_Cliente, CódigoPostal_Cliente)

En esta situación, la relación *Clientes* seguiría sin estar en 3FN, ya que existiría una nueva dependencia funcional transitiva:

Código_Cliente → {Dirección_Cliente, Población_Cliente}
{Dirección_Cliente, Población_Cliente} → CódigoPostal_Cliente

Luego tendríamos que transformar la base de datos de la siguiente forma:

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente)

Productos (Número_Pedido, Código_Producto, Precio_Producto, Cantidad_Producto)

Datos_Productos (Código_Producto, Descripción_Producto)

Clientes (Código_Cliente, Nombre_Cliente, Teléfono_Cliente, Dirección_Cliente, Población_Cliente)

Dirección_Cliente (Dirección_Cliente, Población_Cliente, CódigoPostal_Cliente)

De esta forma, la base de datos quedaría en 3FN, pero no estaría en FNBC, como se puede ver en las siguientes dependencias funcionales:

{Dirección_Cliente, Población_Cliente} → CódigoPostal_Cliente

CódigoPostal_Cliente → Población_Cliente

Como se puede observar, tenemos dos determinantes, uno que es clave primaria y otro que no lo es, por lo que tendríamos que modificar la base de datos, quedando de la siguiente forma:

Pedidos (Número_Pedido, Fecha_Pedido, Precio_Total_Pedido, Código_Cliente)

Productos (Número_Pedido, Código_Producto, Precio_Producto, Cantidad_Producto)

Datos_Productos (Código_Producto, Descripción_Producto)

Clientes (Código_Cliente, Nombre_Cliente, Teléfono_Cliente, Dirección_Cliente, Población_Cliente)
Dirección_Cliente (Dirección_Cliente, CódigoPostal_Cliente)

CódigoPostal_Población (CódigoPostal_Cliente, Población_Cliente)

De esta forma, la base de datos ya estará en FNBC.

NOTA

Dos ideas muy importantes que debemos tener en cuenta:

- Toda relación que esté en FNBC ya está en 3FN (su contrario no es cierto).
- La FNBC puede dar lugar a pérdida de dependencias funcionales, por lo que, en algunos casos, es mejor no aplicarla.

3.5.5 OTRAS FORMAS NORMALES (4FN, 5FN)

Cuarta Forma Normal (4FN)

Se dice que una relación está en 4FN si se encuentra en 3FN y, además, las únicas dependencias funcionales multivaluadas son aquellas que dependen de claves candidatas.

Continuando con el Ejemplo 3.4, para solucionar este problema, habría que generar tantas nuevas relaciones como dependencias funcionales multivaluadas hubiese.

Quinta Forma Normal (5FN)

Una relación está en 5FN si se encuentra en 4FN y, además, toda dependencia de combinación es consecuencia de las claves candidatas.

Continuando con el Ejemplo 3.5, la forma de solucionarlo sería descomponer la relación R en N proyecciones independientes (tres proyecciones en el caso del ejemplo planteado).

3.5.6 DESNORMALIZACIÓN. RAZONES PARA LA DESNORMALIZACIÓN

Una pregunta que está ampliamente extendida en la literatura sería la de:

¿Es necesario llegar hasta la Quinta Forma Normal?

La respuesta unánime es que esa decisión debe ser tomada por el diseñador de la base de datos. Lo natural y más extendido suele ser llegar hasta la 3FN, ya que hay muchos autores que piensan que con la FNBC se puede producir pérdidas de dependencias funcionales.

Además, según avanzamos en el proceso, las relaciones se van "fragmentando" y, por tanto, se ralentiza el acceso a los datos.

3.6 EJERCICIOS RESUELTOS

NOTA

Este extracto de ejercicio pertenece a un ejercicio de las **Oposiciones a cuerpos docentes en Andalucía (Convocatoria de 2004), concretamente para la especialidad de *Sistemas y Aplicaciones Informáticas.***

1. **Partiendo del modelo Entidad-Relación del Eresuelto número 1 del capítulo 2, se pide pasar a tablas el citado modelo Entidad-Relación.**

   Propuesta de resolución:

   Áreas (Código_A, Nombre_A)
   Profesores (NIF_Pr, Nombre_Pr, Domicilio_Pr)
   Encargado (Código_A, NIF_Pr)
   Sedes (Código_S, Nombre_S, Dirección_S)
   Tienen (Código_A, Código_S)
   Cursos (Id_C, Nombre_C, Duración_C, NIF_Coordinador)
   Adscrito (Código_A, Id_C)
   Imparte (NIF_Pr, Id_C)
   Alumnos (NIF_Al, Nombre_Al, Domicilio_Al)
   Asisten (Id_C, NIF_Al)
   Ponentes (NIF_Po, Nombre_Po, Domicilio_Po)
   Participan (Id_C, NIF_Po)

2. **Partiendo del modelo Entidad-Relación del Ejercicio resuelto número 2 del capítulo 2, se proponen los siguiente ejercicios:**

   – **Pasar a tablas el citado modelo Entidad-Relación.**

   – **A partir de los atributos expresados en el modelo Entidad-Relación, construir el modelo relacional hasta la 3FN.**

   Propuesta de resolución:

   – Apartado A:

   Alojamiento (Código_A, Nombre_A, Descripción_A, Capacidad_A)
   Equipamiento (Código_E, Descripción_E)
   Ofrece (Código_A, Código_E, Cantidad)
   Régimen (Código_A, Tipo_R, Precio_R)
   Reserva (Código_R, NPersonas_R, Entrada_R, Salida_R, DNI_Cli, Código_A, Tipo_R)
   Clientes (DNI_Cli, Nombre_Cli)

– Apartado B:

Alojamiento (Código_A, Nombre_A, Descripción_A, Capacidad_A, Código_E, Descripción_E, Cantidad, Tipo_R, Precio_R, Código_R, NPersonas_R, Entrada_R, Salida_R, DNI_Cli, Nombre_Cli)

En primer lugar, en este ejemplo comprobaremos si la relación está en 1FN.

Alojamiento (Código_A, Nombre_A, Descripción_A, Capacidad_A, Código_E, Descripción_E, Cantidad, Tipo_R, Precio_R, Código_R, NPersonas_R, Entrada_R, Salida_R, DNI_Cli, Nombre_Cli)

Podemos concluir que un *Alojamiento* tiene un solo valor para los atributos *Código_A, Nombre_A*, *Descripción_A* y *Capacidad_A*.

Sin embargo, los atributos con doble subrayado NO están en 1FN, ya que un *Alojamiento* consta de varios tipos de *Equipamientos*, con más de un valor para los atributos seleccionados. A su vez, otros atributos multivaluados, pero que se refieren a otro tipo de información, son los que están subrayados con línea de puntos, ya que un *Alojamiento* tiene diferentes tipos de *Reservas*.

Por tanto, podemos afirmar que la relación NO está en 1FN.

Para que esté en 1FN tendríamos que extraer de la tabla original los atributos con algún tipo de subrayado y conformar nuevas tablas. Las modificaciones serían las siguientes:

Alojamiento (Código_A, Nombre_A, Descripción_A, Capacidad_A)
Equipamiento (Código_A, Código_E, Descripción_E, Cantidad)
Régimen (Código_A, Tipo_R, Precio_R, Código_R, NPersonas_R, Entrada_R, Salida_R, DNI_Cli, Nombre_Cli)

Debemos tener presente que tras los cambios obligados para poner la base de datos en 1FN, las tablas resultantes pueden no estar en 1FN, por lo que tenemos que comprobar las tablas modificadas tras las recurrentes modificaciones, hasta que efectivamente toda la base de datos esté en 1FN.

Concretamente, en este caso, si observamos la tabla *Régimen*, podemos ver que de un mismo tipo de *Régimen* se pueden realizar múltiples *Reservas* (cada una a un único *Cliente*). Por ello, los atributos que vuelven a estar con doble subrayado saldrán de esta tabla.

El resultado sería el que se muestra a continuación:

Alojamiento (Código_A, Nombre_A, Descripción_A, Capacidad_A)
Equipamiento (Código_A, Código_E, Descripción_E, Cantidad)
Régimen (Código_A, Tipo_R, Precio_R)
Reservas (Código_A, Tipo_R, Código_R, NPersonas_R, Entrada_R, Salida_R, DNI_Cli, Nombre_Cli)

Y ahora, las relaciones resultantes, como podemos comprobar, sí están en 1FN.

Seguiremos analizando si la base de datos está en 2FN. Si observamos las relaciones tenemos que:

– La relación *Alojamiento* tiene como clave primaria un solo atributo. Por tanto, todos los atributos de la relación guardan dependencia funcional completa con dicha clave primaria y, de este modo, podemos afirmar que la relación está en 2FN.

– En la relación *Equipamiento*, solo el atributo *Cantidad* cumple con la dependencia funcional completa con respecto a la clave primaria:

$$\{Código\_A, Código\_E\} \Rightarrow \{Cantidad\}$$

Sin embargo, el atributo con doble subrayado no cumple la dependencia funcional completa, ya que depende funcionalmente solo de un subconjunto de la clave primaria:

Código_E → Descripción_E

– En la relación *Régimen*, el atributo *Precio_R* si cumplen con la dependencia funcional completa con respecto a la clave primaria:

{Código_A, Tipo_R} ⇒ {Precio_R}

Por tanto, esta relación sí está en 2FN.

– Y, la relación *Reservas* tiene como clave primaria un solo atributo. Por tanto, todos los atributos de la relación guardan dependencia funcional completa con dicha clave primaria y, de este modo, podemos afirmar que la relación está en 2FN.

Podemos concluir, por tanto, que la relación *Equipamiento* NO está en 2FN y, de este modo, la base de datos no está en 2FN. Así, la base de datos tendrá que transformarse en la que a continuación se muestra:

Alojamiento (Código_A, Nombre_A, Descripción_A, Capacidad_A)
Equipamiento (Código_A, Código_E, Cantidad)
Datos_Equipamiento (Código_E, Descripción_E)
Régimen (Código_A, Tipo_R, Precio_R)
Reservas (Código_A, Tipo_R, Código_R, NPersonas_R, Entrada_R, Salida_R, DNI_Cli, Nombre_Cli)

Estas tablas sí están en 2FN.

Por último, analizaremos si existe alguna dependencia funcional transitiva para solucionarla y que la base de datos pueda estar en 3FN.

En este caso nos centraremos en la relación *Reservas*, en la que observamos que:

Código_R → DNI_Cli (A → B)

Código_Cli → {Nombre_Cli} (B → C)

Luego existe una dependencia funcional transitiva:

Código_R → {Nombre_Cli} (A → C)

Podemos concluir, por tanto, que la relación *Reserva* NO está en 3FN y, de este modo, la base de datos no está en 3FN.

Y, por último, tras los cambios obligados por esta contingencia, la base de datos ya está en 3FN.

Alojamiento (Código_A, Nombre_A, Descripción_A, Capacidad_A)
Equipamiento (Código_A, Código_E, Cantidad)
Datos_Equipamiento (Código_E, Descripción_E)
Régimen (Código_A, Tipo_R, Precio_R)
Reservas (Código_A, Tipo_R, Código_R, NPersonas_R, Entrada_R, Salida_R, DNI_Cli)
Clientes (DNI_Cli, Nombre_Cli)

3. **Partiendo del modelo Entidad-Relación del Ejercicio resuelto número 3 del capítulo 2, se proponen los siguiente ejercicios:**

   - **Pasar a tablas el citado modelo Entidad-Relación.**
   - **A partir de los atributos expresados en el modelo Entidad-Relación, construir el modelo relacional hasta la 3FN.**

   Propuesta de resolución:

   - Apartado A:

     Sucursales (Código_S, Domicilio_S, Teléfono_S)
     Empleados (DNI_E, Nombre_E, Apellidos_E, Teléfono_E, Código_S)
     Revistas (NRegistro_R, Título_R, Tipo_R, Periodicidad_R, Código_S)
     Secciones (NRegistro_R, Título_Sec, Extensión_Sec)
     Periodistas (DNI_P, Nombre_P, Apellidos_P, Especialidad_P)
     Escriben (DNI_P, NRegistro_R)
     Números (NRegistro_R, Fecha_N, NPáginas_N, NEjemplares_N)

   - Apartado B:

     Sucursales (Código_S, Domicilio_S, Teléfono_S, DNI_E, Nombre_E, Apellidos_E, Teléfono_E, NRegistro_R, Título_R, Tipo_R, Periodicidad_R, Título_Sec, Extensión_Sec, DNI_P, Nombre_P, Apellidos_P, Especialidad_P, Fecha_N, NPáginas_N, NEjemplares_N)

     En primer lugar, en este ejemplo comprobaremos si la relación está en 1FN.

     Sucursales (Código_S, Domicilio_S, Teléfono_S, DNI_E, Nombre_E, Apellidos_E, Teléfono_E, NRegistro_R, Título_R, Tipo_R, Periodicidad_R, Título_Sec, Extensión_Sec, DNI_P, Nombre_P, Apellidos_P, Especialidad_P, Fecha_N, NPáginas_N, NEjemplares_N)

     Podemos concluir que una *Sucursal* tiene un solo valor para los atributos *Código_S, Domicilio_S*, y *Teléfono_S*.

     Sin embargo, los atributos con doble subrayado NO están en 1FN, ya que en una *Sucursal* trabajan varios *Empleados*, con más de un valor para los atributos seleccionados. A su vez, otros atributos multivaluados, pero que se refieren a otro tipo de información, son los que están subrayados con línea de puntos, ya que una *Sucursal* publica diferentes *Revistas*.

     Por tanto, podemos afirmar que la relación NO está en 1FN.

     Para que esté en 1FN tendríamos que extraer de la tabla original los atributos con algún tipo de subrayado y conformar nuevas tablas. Las modificaciones serían las siguientes:

     - **Sucursales** (Código_S, Domicilio_S, Teléfono_S)
     - **Empleados** (Código_S, DNI_E, Nombre_E, Apellidos_E, Teléfono_E)
     - **Revistas** (Código_S, NRegistro_R, Título_R, Tipo_R, Periodicidad_R, Título_Sec, Extensión_Sec, DNI_P, Nombre_P, Apellidos_P, Especialidad_P, Fecha_N, NPáginas_N, NEjemplares_N)

   Debemos tener presente que tras los cambios obligados para poner la base de datos en 1FN, las tablas resultantes pueden no estar en 1FN, por lo que tenemos que comprobar las tablas modificadas tras las recurrentes modificaciones, hasta que efectivamente toda la base de datos esté en 1FN.

Concretamente, en este caso, si observamos la tabla *Revistas*, podemos ver que una misma *Revista:*

- Tiene diferentes secciones.
- Escriben muchos periodistas.
- Sacan diferentes números de cada una.

Por ello, los atributos que vuelven a estar con algún tipo de subrayado deberán salir de esta tabla.

El resultado sería el que se muestra a continuación:

Sucursales (Código_S, Domicilio_S, Teléfono_S)
Empleados (Código_S, DNI_E, Nombre_E, Apellidos_E, Teléfono_E)
Revistas (Código_S, NRegistro_R, Título_R, Tipo_R, Periodicidad_R)
Secciones (NRegistro_R, Título_Sec, Extensión_Sec)
Periodistas (NRegistro_R, DNI_P, Nombre_P, Apellidos_P, Especialidad_P)
Número (NRegistros_R, Fecha_N, NPáginas_N, NEjemplares_N)

Y ahora, las relaciones resultantes, como podemos comprobar, sí están en 1FN.

Seguiremos analizando si la base de datos está en 2FN. Si observamos las relaciones tenemos que:

- Las relaciones *Sucursales*, *Empleados* y *Revistas* tiene como clave primaria un solo atributo. Por tanto, todos los atributos de la relación guardan dependencia funcional completa con dicha clave primaria y, de este modo, podemos afirmar que la relación está en 2FN.
- En la relación *Periodistas* los atributos con doble subrayado no cumplen la dependencia funcional completa, ya que dependen funcionalmente solo de un subconjunto de la clave primaria:

$$DNI\_P \rightarrow \{Nombre\_P, Apellidos\_P, Especialidad\_P\}$$

- En las relaciones *Secciones* y *Número*, todos los atributos cumplen con la dependencia funcional completa con respecto a la clave primaria y, por tanto, estas relaciones sí están en 2FN.

Podemos concluir, por tanto, que la relación *Periodistas* NO está en 2FN y, de este modo, la base de datos no está en 2FN. Así, la base de datos tendrá que transformarse en la que a continuación se muestra:

Sucursales (Código_S, Domicilio_S, Teléfono_S)
Empleados (Código_S, DNI_E, Nombre_E, Apellidos_E, Teléfono_E)
Revistas (Código_S, NRegistro_R, Título_R, Tipo_R, Periodicidad_R)
Secciones (NRegistro_R, Título_Sec, Extensión_Sec)
Periodistas (NRegistro_R, DNI_P)
Datos_Periodistas (DNI_P, Nombre_P, Apellidos_P, Especialidad_P)
Número (NRegistros_R, Fecha_N, NPáginas_N, NEjemplares_N)

Estas tablas sí están en 2FN.

Por último, observamos que en las tablas anteriores no hay ninguna dependencia funcional transitiva, luego podemos concluir que también están en 3FN.

3.7 EJERCICIOS PROPUESTOS

NOTA

Este extracto de ejercicio pertenece a un ejercicio de las **Oposiciones a cuerpos docentes en Andalucía (Convocatoria de 2006), concretamente para la especialidad de *Sistemas y Aplicaciones Informáticas.***

1. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 1 del capítulo 2, se proponen los siguiente ejercicios:

- Pasar a tablas el citado modelo Entidad-Relación.
- A partir de los atributos expresados en el modelo Entidad-Relación, construir el modelo relacional hasta la 3FN.

NOTA

Este extracto de ejercicio pertenece a un ejercicio de las **Oposiciones a cuerpos docentes en Andalucía (Convocatoria de 2006), concretamente para la especialidad de *Sistemas y Aplicaciones Informáticas.***

2. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 2 del capítulo 2, se proponen los siguiente ejercicios:

- Pasar a tablas el citado modelo Entidad-Relación.
- A partir de los atributos expresados en el modelo Entidad-Relación, construir el modelo relacional hasta la 3FN.

3. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 3 del capítulo 2, se proponen los siguiente ejercicios:

- Pasar a tablas el citado modelo Entidad-Relación.
- A partir de los atributos expresados en el modelo Entidad-Relación, construir el modelo relacional hasta la 3FN.

4. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 4 del capítulo 2, se proponen los siguiente ejercicios:

- Pasar a tablas el citado modelo Entidad-Relación.
- A partir de los atributos expresados en el modelo Entidad-Relación, construir el modelo relacional hasta la 3FN.

5. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 5 del capítulo 2, se proponen los siguiente ejercicios:

- Pasar a tablas el citado modelo Entidad-Relación.
- A partir de los atributos expresados en el modelo Entidad-Relación, construir el modelo relacional hasta la 3FN.

3.8 TEST DE CONOCIMIENTOS

NOTA

Las consultas 1 y 2 pertenecen a un ejercicio de las **Oposiciones al Cuerpo de Técnicos Auxiliares de Informática de la Administración del Estado (Convocatoria de 2015)**.

1 Dentro del modelo Entidad-Relación se puede definir el dominio de un atributo como:
- **a)** El número mínimo de correspondencias en las que puede tomar parte cada ocurrencia de dicha entidad.
- **b)** Es una correspondencia o asociación entre dos o más entidades con los mismos atributos.
- **c)** El conjunto de todos los valores posibles que puede tomar el atributo.
- **d)** El número de tuplas o filas de un atributo.

2 Dentro del modelo relacional, señale la afirmación errónea:
- **a)** En una relación puede haber varias claves candidatas.
- **b)** Una clave primaria es, a su vez, clave candidata.
- **c)** Si se elimina un atributo de una clave candidata, ésta deja de serlo.
- **d)** Una relación puede no tener clave candidata.

3 Señala la afirmación incorrecta:
- **a)** Las tuplas en una relación no guardan un orden concreto y dos tuplas pueden tener el mismo valor de clave primaria.
- **b)** La unicidad de nombre, los dominios atómicos de sus atributos y las columnas con distinto nombre son propiedades de las relaciones en el modelo relacional.
- **c)** Para columnas con el mismo nombre en tablas distintas se utilizan notaciones especiales para diferenciarlas.
- **d)** Los atributos en una relación no guardan un orden concreto.

4 Indica la afirmación correcta:
- **a)** La cardinalidad indica el tamaño de una relación en base al número de columnas que contiene y el grado es el número de tuplas o filas que contiene una tabla.
- **b)** El grado indica el tamaño de una relación en base al número de columnas que contiene y la cardinalidad es el número de tuplas o filas que contiene una tabla.
- **c)** Grado y cardinalidad son términos sinónimos para relaciones cuadradas.
- **d)** Ninguna de las anteriores.

5 Podemos clasificar los tipos de tablas en:
- **a)** Temporales y vistas
- **b)** Base, vistas e instantáneas
- **c)** Persistentes e instantáneas
- **d)** Persistentes (base, vistas e instantáneas) y temporales

6 Una clave ajena…
- **a)** Siempre debe ser única en la tabla de origen.
- **b)** Siempre debe ser única en la tabla donde se añade para expresar la relación entre tablas.
- **c)** Nunca es única en ninguna tabla.
- **d)** Ninguna de las anteriores.

7 Señala la afirmación correcta:
- **a)** La unicidad y la obligatoriedad son restricciones inherentes a una base de datos relacional.
- **b)** Las restricciones explícitas no son determinadas por los usuarios.
- **c)** Las restricciones semánticas deben cumplir una regla de validación.
- **d)** Ninguna de las anteriores.

8 Señale la afirmación incorrecta:
- **a)** Verdadero OR Verdadero = Verdadero
- **b)** Null OR Falso = Null
- **c)** Falso AND Verdadero = Verdadero
- **d)** NOT Null = Null

9 Una dependencia funcional completa…
- **a)** Se produce cuando, dados los atributos A, B y C, sucede que A → B y, además, B → C.
- **b)** Se produce cuando, dados A, B y C, tres subconjuntos distintos de atributos, se dice para B con respecto a A, si para cada valor de A tiene asignado un conjunto bien definido de valores de B y, además, ese conjunto de valores es independiente de cualquier valor de otro conjunto de atributos C.
- **c)** Se produce cuando, dados dos atributos A y B, para B, si para cada valor de A existe un valor de B y solo uno, asociado con él.
- **d)** Se produce un atributo B depende funcionalmente de otro atributo A, pero no tiene dependencia funcional de ningún subconjunto de A.

10 Una dependencia funcional …
- **a)** Se produce cuando, dados los atributos A, B y C, sucede que A → B y, además, B → C.
- **b)** Se produce cuando, dados A, B y C, tres subconjunto distintos de atributos, se dice para B con respecto a A, si para cada valor de A tiene asignado un conjunto bien definido de valores de B y, además, ese conjunto de valores es independiente de cualquier valor de otro conjunto de atributos C.
- **c)** Se produce cuando, dados dos atributos A y B, para B, si para cada valor de A existe un valor de B y solo uno, asociado con él.
- **d)** Se produce un atributo B depende funcionalmente de otro atributo A, pero no tiene dependencia funcional de ningún subconjunto de A.

11 Una dependencia funcional transitiva…

a) Se produce cuando, dados los atributos A, B y C, sucede que A → B y, además, B → C.

b) Se produce cuando, dados A, B y C, tres subconjuntos distintos de atributos, se dice para B con respecto a A, si para cada valor de A tiene asignado un conjunto bien definido de valores de B y, además, ese conjunto de valores es independiente de cualquier valor de otro conjunto de atributos C.

c) Se produce cuando, dados dos atributos A y B, para B, si para cada valor de A existe un valor de B y solo uno, asociado con él.

d) Se produce un atributo B depende funcionalmente de otro atributo A, pero no tiene dependencia funcional de ningún subconjunto de A.

12 Una dependencia funcional multivaluada…

a) Se produce cuando, dados los atributos A, B y C, sucede que A → B y, además, B → C.

b) Se produce cuando, dados A, B y C, tres subconjunto distintos de atributos, se dice para B con respecto a A, si para cada valor de A tiene asignado un conjunto bien definido de valores de B y, además, ese conjunto de valores es independiente de cualquier valor de otro conjunto de atributos C.

c) Se produce cuando, dados dos atributos A y B, para B, si para cada valor de A existe un valor de B y solo uno, asociado con él.

d) Se produce un atributo B depende funcionalmente de otro atributo A, pero no tiene dependencia funcional de ningún subconjunto de A.

13 Indica la afirmación correcta:

a) Siempre es necesario llegar a la 5FN.

b) Lo mejor es llegar hasta la FNBC.

c) Lo mejor es llegar hasta la 3FN.

d) La opción que más fragmente la base de datos es la más correcta.

14 En las relaciones siguientes:

PEDIDOS (Núm_Ped, Precio_Total_Ped, DNI_Cli, Nombre_Cli, Apellidos_Cli)

PRODUCTOS (Núm_Ped, Cód_Art, Cantidad, Precio, Desc_Art)

a) Están en 1FN.

b) Están en 2FN.

c) Están en 3FN.

d) Están en FNBC.

15 En las relaciones siguientes:

PEDIDOS (Núm_Ped, Precio_Total_Ped, DNI_Cli, Nombre_Cli, Apellidos_Cli)

PRODUCTOS (Núm_Ped, Cód_Art, Cantidad, Precio, Desc_Art)

a) La relación *PEDIDOS* no está en 3FN y la relación *PRODUCTOS* no está en 2FN.
b) La relación *PEDIDOS* no está en 1FN y la relación *PRODUCTOS* no está en 2FN.
c) La relación *PEDIDOS* no está en 2FN y la relación *PRODUCTOS* no está en 3FN.
d) La relación *PEDIDOS* no está en 2FN y la relación *PRODUCTOS* no está en 1FN.

4 El ciclo de vida de un proyecto

4.1 EL CICLO DE VIDA DE UNA BASE DE DATOS

Una base de datos es un componente más, quizás uno de los más importantes, de un sistema de información o *software* determinado, por lo que su ciclo de vida estaría incluido en el ciclo de vida propio del sistema de información o aplicación informática.

En la bibliografía existe una amplia amalgama de ítems que compondrían lo que se denomina **ciclo de vida de una base de datos**, aunque esta propuesta bibliográfica los clasifica en los siguientes:

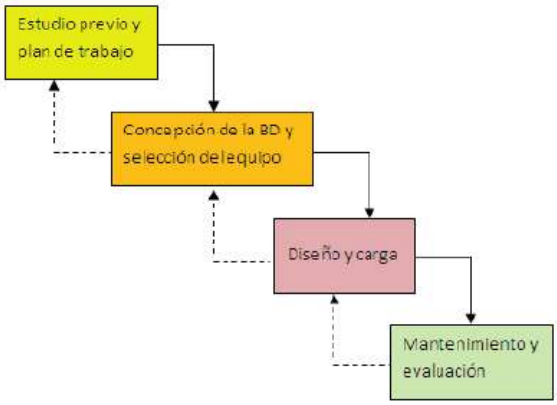


Figura 4.1. *Ciclo de vida de una base de datos*

4.1.1 ESTUDIO PREVIO Y PLAN DE TRABAJO. ACTIVIDADES

Esta tarea, también conocida en la literatura como **definición del sistema** o **análisis de requisitos de la base de datos**, consiste en llevar a cabo un análisis de los requerimientos del sistema, fijándonos especialmente en todos los requisitos asociados a los datos con los que va a trabajar nuestro sistema de información.

El objetivo de esta fase es el de recabar información sobre el uso que se le desea dar a la base de datos. Por su parte, las principales acciones a llevar a cabo son:

- Análisis de la situación de la compañía (estudio del entorno operacional actual).
- Identificación de problemas y restricciones, especialmente relacionados con la información (transacciones, frecuencias, flujos de datos, etc.).
- Definición de objetivos y determinación del alcance.

En esta fase también es momento de determinar aspectos tan importantes como, por ejemplo:

- La **viabilidad**, desde diferentes puntos de vista:
  - **Económico:** para lo cual se debe hacer una estimación del esfuerzo que llevará la implementación de la base de datos, que se traducirá en un presupuesto a nivel económico.

- **Legal:** consiste en analizar si la base de datos del sistema de información se puede implementar sin que interfieran limitaciones legales (como podría ser la LOPD, Ley Orgánica de Protección del Menor, u otra).
- **Técnico-operativa:** analizando si técnicamente es posible la implementación de los problemas y restricciones antes estudiados, cumpliendo con los requisitos operativos y de tiempos de respuesta del sistema de información global y con las necesidades empresariales.

- **Estimación de plazos y costes**, para lo cual se proponen, por un lado, modelos de estimación de coste y esfuerzo, entre los que destacan los Modelos de regresión, el Modelo de Bailey-Basili, el extendido Modelo COCOMO o el Modelo SLIM de Putnam.

  Por su parte, para la planificación temporal o estimación de plazos, existen diversas herramientas como las Redes de Pert o los Diagramas de Gantt.

4.1.2 CONCEPCIÓN DE LA BASE DE DATOS Y SELECCIÓN DEL EQUIPO FÍSICO Y LÓGICOS

El objetivo de la fase de **diseño conceptual de la base de datos** (con independencia del DBMS que se vaya a implantar finalmente) consistirá en llevar a cabo un esquema conceptual de la base de datos, que se puede materializar en un diagrama E-R, como estudiamos en el capítulo 2, un diagrama de clases UML, o similares, así como el diccionario de datos.

RECUERDA

Ahondaremos en estos conceptos, especialmente en el de **diccionario de datos** en el tema 5, que se denomina "Creación y Diseño de bases de datos".

Posteriormente, tendrá lugar la tarea de **selección del DBMS** que vayamos a utilizar en nuestro sistema de información.

A continuación analizaremos estas tareas con mayor detenimiento.

4.1.2.1 Conceptos generales acerca del análisis de aplicaciones

Siguiendo la publicación de Rober S. Pressman, podemos establecer los primeros trabajos sobre modelos de análisis de aplicaciones a finales de los 60, pero la primera aparición del enfoque de *análisis estructurado* fue como complemento de otro concepto, el de *diseño estructurado*, debido a que los autores necesitaban una notación gráfica para representar los datos y los procesos que los transforman.

El término *análisis estructurado* originalmente fue acuñado por Douglas Ross y posteriormente popularizado por DeMarco.

El modelo de análisis tiene, como principales objetivos, los tres siguientes:

- Describir lo que requiere el cliente.
- Establecer una base para la creación de un diseño de *software*.
- Definir un conjunto de requisitos que se pueda validar una vez que se construya el *software*.

Para lograr dichos objetivos, el modelo de análisis sigue la siguiente estructura:

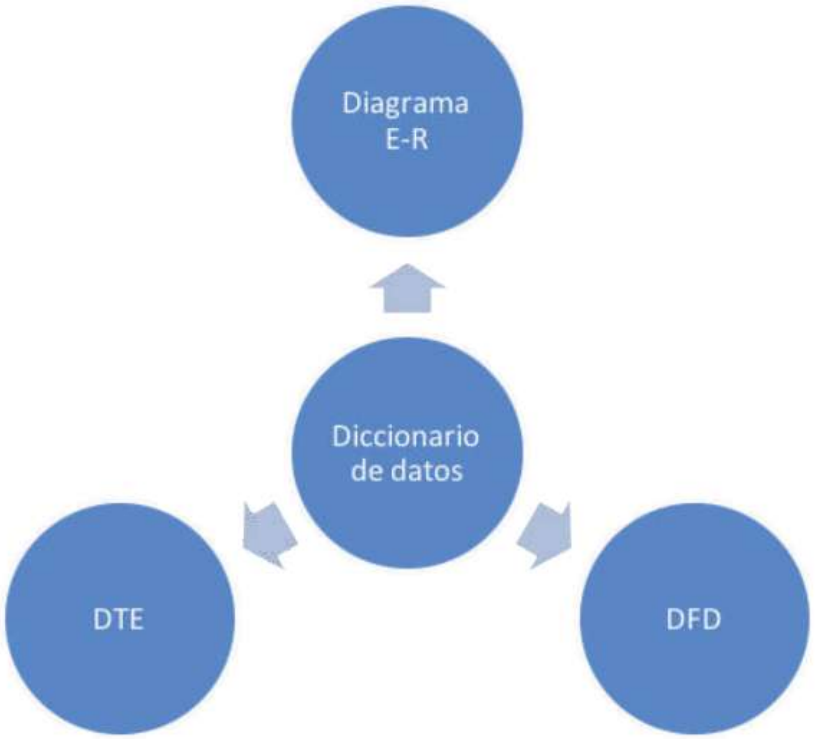


Figura 4.2. *Estructura del modelo de análisis*

En el centro del esquema, podemos observar el **diccionario de datos** que contiene, como veremos en el capítulo siguiente, las definiciones de todos los datos que emplea el sistema de información.

Y, en segundo lugar, tenemos tres clases de diagramas que complementan la información del citado diccionario de datos:

- El **diagrama Entidad-Relación (diagrama E-R)** representa los datos que contendrá la base de datos y las relaciones entre ellos.

  Los atributos del citado diagrama E-R estarán descritos en el diccionario de datos.

- El **diagrama de flujo de datos (DFD)** tiene principalmente dos tareas:
  - Indicar cómo se transforman los datos según se avanza en el sistema de información.
  - Representar las funciones (métodos) que utilizan dichos datos y los transforman.

- El **diagrama de transición de estados (DTE)** indica cómo se comporta el sistema como consecuencia de sucesos externos (estímulos del exterior), representando cada modo de comportamiento (denominado *estado*) y la manera en que se hacen las transiciones de un estado a otro/s. Su principal tarea consiste en establecer la base para el modelado de comportamiento.

4.1.2.2 Concepción de la base de datos

Como ya hemos comentado, el objetivo fundamental del **diseño conceptual de la base de datos** consiste en realizar un esquema teórico o conceptual, en forma de diagrama E-R, diagrama CASE o diagrama de clases UML, entre otros, así como el diccionario de datos con el comportamiento y principales características de la base de datos que queremos, posteriormente, implementar o implantar.

Para tener éxito en esta fase, tendremos que llevar a cabo las siguientes subtareas:

- Analizar la estructura, semántica, relaciones y restricciones asociadas a los datos que almacenará la base de datos.
- Obtener la descripción de lo que será el contenido de la base de datos.
- Analizar la forma en que se comunicarán los usuarios finales, analistas y diseñadores con la base de datos, para comprobar la validez del modelo obtenido y para implantación final del sistema.

4.1.2.3 Selección del equipo físico y lógicos necesarios

La elección del DBMS a emplear se lleva a cabo a través de la realización de dos etapas:

- En primer lugar, se lleva a cabo la **elección del modelo de datos**, el tipo de DBMS que es más adecuado (relacional, objeto-relacional, orientado a objetos, etc.).

RECUERDA

En la sección 1.3.5 "Modelos de datos. Clasificación" pudimos estudiar los principales tipos de DBMS existentes.

- A continuación, se elige el DBMS concreto (y su versión). Hoy en día el modelo relacional está tan extendido que su elección es una garantía de éxito. Entre los DBMS relacionales comerciales existentes, podemos destacar:
  - **Oracle Database:** se trata de un DBMS objeto-relacional desarrollado por Oracle Corporation. Quizás sea de los más extendidos en el mercado profesional, destacando por su soporte de transacciones, estabilidad, escalabilidad y soporte multiplataforma.

  - **MySQL:** es el DBMS de *software* libre más importante y mayor competidor por la hegemonía de los DMBS de Oracle. Se trata de una DBMS relacional, multiusuario y también multihilo y está desarrollado y soportado por MySQL AB.

- **Microsoft SQL Server:** es un DBMS producido por Microsoft y está caracterizado principalmente por su soporte de transacciones y de procedimientos almacenados. También incluye un entorno gráfico para el DDL y DML.

También se caracteriza por permitir administrar información de otros servidores de datos.

Además, este producto incluye una versión que, aunque reducida, se distribuye de forma gratuita. Esta se denomina MSDE (en su versión de 2008, que es la última, se denomina SQL Express Edition).

Según nos explica el profesor Berzal Galiano, debemos partir de la base de que un DBMS es un producto *software* con capacidad para definir, mantener y utilizar bases de datos. Por ello, el DBMS *"que decidamos emplear debe permitirnos, entre otras cosas, definir estructuras de almacenamiento adecuadas y acceder a los datos de forma eficiente y segura"*.

Teniendo esto en cuenta, Berzal Galiano establece una clasificación de los principales aspectos a tener en cuenta en la selección de un DBMS, agrupando estos en factores técnicos y factores no técnicos.

- Entre los **factores técnicos** podemos destacar:
  - La **portabilidad** del DBMS entre diferentes tipos de *hardware* y *software* (es decir, su independencia física y lógica).
  - La calidad y variedad de **herramientas** para el desarrollo de sistemas de información, así como los **tipos de lenguaje de consulta** que ofrece o las **interfaces de usuario y programador** que dispone.
  - Las opciones de **arquitectura cliente-servidor**.
  - Y otros factores, cada vez menos tenidos en cuenta debido a su estandarización, tales como:
    - No redundancia, consistencia e integridad.
    - Fiabilidad, es decir, respuesta ante posibles fallos de *hardware* o *software*, y seguridad.
    - Posibilidad de comunicarse con otros DBMS.
- Por otra parte, entre los **factores no técnicos** nos encontramos con los siguientes:
  - El **coste de adquisición** del DBMS.
  - El **coste de mantenimiento** del DBMS.
  - El **coste de adquisición del *hardware*** necesario para implantar el DBMS (requisitos de memoria, memoria RAM, núcleos de procesamiento, etc.).
  - El **coste de personal**, bien a nivel de formación o, incluso, de contratación de personal especializado.
  - **Adopción de nuevos protocolos de trabajo** que tendrán que asumir los integrantes de la empresa que adquiera el nuevo DBMS, también a nivel de comunicación con el proveedor y, del mismo modo, con respecto a clientes finales que puedan llegar a trabajar con el citado DBMS.

4.1.3 DISEÑO Y CARGA

Una vez diseñada a nivel conceptual la base de datos y tomado una decisión sobre el DBMS que la soportará, llega el momento de implantarla definitivamente, por lo que tendremos que retomar la fase de diseñado, anteriormente iniciada, para llevar a cabo un diseño tanto a nivel lógico, como a nivel físico, como a continuación analizaremos.

Este proceso será previo a la carga de la información en el nuevo DBMS, que son los datos definitivos con los que el sistema de información o *software* va a trabajar.

4.1.3.1 Conceptos generales acerca del diseño de aplicaciones

De acuerdo con la definición de Taylor, *"(…) diseño es el proceso de aplicar distintas técnicas y principios con el propósito de definir un dispositivo, proceso, o sistema, con los suficientes detalles como para permitir su realización física (…)".*

INFORMACIÓN

Esta definición ha sido extraida de su trabajo más relevante:
Taylor, E. S., *An Interim Report on Engineering Design.* Massachusetts Institute of Technology, 1959.

Esta fase se deberá llevar a cabo a partir de la información obtenida en la fase de análisis y, a su vez, se puede descomponer en diversos ámbitos de diseño, entre los que destacan:

- El **diseño de datos** consiste en el diseño lógico y físico de la base de datos que, a continuación, estudiaremos.
- El **diseño arquitectónico** define las relaciones entre los principales elementos estructurales del programa o sistema de información.
- El **diseño procedimental** transforma los elementos estructurales en una descripción procedimental del *software*.
- El **diseño de la interfaz de usuario** establece, a nivel de visión, los formularios e informes con los que el sistema de información mostrará los datos, con los que el usuario final podrá interactuar.

4.1.3.2 Diseño lógico

El objetivo de llevar a cabo el **diseño lógico** es el de transformar el esquema conceptual (anteriormente comentado) en un esquema lógico, en función del tipo de DBMS que hayamos escogido.

Si adaptamos esta definición al modelo relacional que hemos estado estudiando en el capítulo 3, el proceso consistirá en adaptar el modelo conceptual (el cual consta de atributos, entidades, relaciones, etc.), en elementos lógicos de la base de datos, concretamente del citado modelo relacional (campos, relaciones, tuplas, etc.). Este será el resultado que se obtendrá en esta fase, esto es, el modelo lógico de datos, en nuestro caso, el modelo relacional.

4.1.3.3 Diseño físico

Por su parte, el **diseño físico** de la base de datos consiste en seleccionar las estructuras de almacenamiento físico apropiadas con el objetivo de que el rendimiento de la base de datos sea óptimo, en función de los requisitos del sistema de información a implantar.

Como resultado de este proceso, obtendremos las sentencias de definición de datos (DDL) (creación de tablas, índices, etc.), escritas en el lenguaje de programación propio del DBMS (como, por ejemplo, SQL).

Esta fase se puede llevar a cabo de dos formas diferentes:

- **Empleando utilidades gráficas** que permiten realizar la transformación del esquema lógico al esquema físico de una forma automatizada y muy intuitiva.
- **Implementando directamente las sentencias DDL**, por parte del desarrollador de la base de datos. A lo largo de este documento será la forma con la que trabajaremos, aunque el uso de utilidades gráficas es una solución muy interesante.

INFORMACIÓN

A modo de propuesta de utilidad gráfica para pasar del esquema lógico al físico, planteamos el uso de la herramienta *DBDesigner.*

Se trata de una aplicación muy intuitiva con la que se puede diseñar la base de datos con una nomenclatura tipo esquema E-R de muy fácil manejo y una potente opción de *Exportar a SQL*, con la que se generará un archivo con las sentencias SQL antes comentadas.

4.1.3.4 Carga y optimización de la base de datos

En algunas ocasiones, la fase de carga de datos en el nuevo DBMS puede llegar a ser una de las tareas más tediosas y costosas en tiempo de todo el ciclo de vida de la base de datos, debido a la posibilidad de tener que trabajar con enormes volúmenes de datos.

Aunque aparentemente esta tarea sea repetitiva y poco valiosa, es muy importante prestar una importante atención a la misma, ya que es en la carga de datos donde se pueden detectar posibles errores de diseño, incumplimiento de restricciones esenciales para la base de datos y, también, posibles alternativas de optimización de la citada base de datos.

Por ello, la propuesta de este libro es realizar dicha carga de datos de forma planificada, diseñando al mismo tiempo mecanismos (fundamentalmente basados en consultas y otras operaciones) de prueba con el objetivo de medir, tanto si se cumplen las diferentes restricciones de integridad, como el rendimiento de la base de datos.

INFORMACIÓN

Os aconsejamos la lectura de este artículo sobre "Cómo optimizar bases de datos en MySQL":
http://blog.arsys.es/como-optimizar-bases-de-datos-mysql/

4.2 CONCEPTOS GENERALES DEL CONTROL DE CALIDAD

De acuerdo con el Diccionario de la Real Academia Española, podemos entender el concepto de calidad como *"conjunto de propiedades inherentes a una cosa, que permiten apreciarla como igual, mejor o peor que las restantes de su especie"*.

Si nos orientamos hacia una definición más relacionada con la informática, tenemos que Roger S. Pressman la define como la *"concordancia con los requisitos funcionales y de rendimiento explícitamente establecidos, con los estándares de desarrollo explícitamente documentados y con las características implícitas que se espera de todo* software *desarrollado profesionalmente"*.

Desde hace algunos años, se ha tomado conciencia de la importancia de gestionar la calidad de las bases de datos, tomándose como referente el extendido trabajo propuesto por el SCOUG (Southern California Online User Group) acerca de los principales indicadores de calidad para bases de datos, mas solo desde el punto de vista del usuario. Por su parte, Rodríguez Yunta, en su trabajo *Evaluación e indicadores de calidad en bases de datos*, hace una propuesta más completa sobre la evaluación de las bases de datos, añadiendo los puntos de vista de los productores y administradores de las citadas base de datos.

De este modo, Rodríguez Yunta propone los siguientes tipos de indicadores:

- Indicadores de calidad que dependen de las fuentes seleccionadas:
  - Grado de cobertura temática o alcance.
  - Grado de especialización temática.
  - Calidad y accesibilidad de los documentos originales.
- Indicadores que dependen de la actualización y presentación de los datos:
  - Grado de actualización.
  - Nivel de crecimiento.
  - Fiabilidad y precisión de los datos.
- Indicadores que dependen del diseño de la base de datos y criterios aplicados en el análisis documental:
  - Capacidad para realizar búsquedas exhaustivas.
  - Capacidad para realizar búsquedas precisas.

- Facilidad para juzgar la adecuación de los registros a la búsqueda deseada.
- Esfuerzo requerido en la recuperación.
- Consistencia del análisis documental.

- Indicadores que dependen de la forma de acceso de los usuarios:
  - Amigabilidad del lenguaje de recuperación.
  - Potencialidad del lenguaje de recuperación.
  - Atención al usuario.
  - Calidad de las salidas.
  - Accesibilidad.

- Indicadores de calidad de bases de datos en una búsqueda concreta:
  - Tasa de pertinencia, precisión o relevancia.
  - Tasa de acierto, de respuesta o de exhaustividad.
  - Tasa de actualización.

NOTA

Es muy recomendable la lectura de este artículo al completo donde explica detalladamente cada uno de los indicadores de calidad en bases de datos:

http://digital.csic.es/bitstream/10261/27342/1/594.pdf

Dicho trabajo cuenta con una Licencia Creative Commons 3.0, por lo que su lectura y uso es libre bajo los términos de dicha licencia.

Además, el artículo está publicado en la *Revista española de documentación científica*, la cual cuenta con un bibliografía muy amplia y variada sobre el tema en cuestión:
http://redc.revistas.csic.es

4.2.1 CONTROL DE CALIDAD DE LAS ESPECIFICACIONES FUNCIONALES

Las **especificaciones** o **requisitos** son un aspecto fundamental para llevar a cabo un correcto control de calidad, no solo en el diseño e implementación de bases de datos, sino, con carácter general, a nivel de desarrollo de *software*. De hecho, todo ello ha dado lugar al paradigma denominado como **ingeniería de requisitos**.

Estos requisitos o especificaciones determinan lo que hará el sistema o base de datos (es decir, cómo funcionará) y las restricciones sobre su operación e implementación. El proceso para definir los requisitos o especificaciones de una base de datos se lleva a cabo a través de la elicitación, análisis y especificación de los requisitos, en base a las necesidades de los usuarios.

A su vez, las especificaciones o requisitos se pueden clasificar en base a dos criterios:

- Requisitos de usuario y del sistema:
  - Los **requisitos de usuario** son declaraciones, en lenguaje natural o diferentes diagramas, de los servicios de la base de datos y de las restricciones bajo las que debe operar.

- Los **requisitos del sistema** conforman un documento estructurado que determina las descripciones detalladas de los servicios de la base de datos. Se trata de un escrito como contrato entre el cliente y el desarrollador, que debe contener una especificación completa y consistente del sistema.

- Requisitos funcionales y no funcionales:
  - Los **requisitos funcionales** definen los servicios que la base de datos debe proporcionar, cómo debe reaccionar a una entrada particular y cómo se debe comportar ante situaciones especiales.
  - Los **requisitos no funcionales** deben contemplar las restricciones que afectan a los servicios o funciones de la base de datos, tales como restricciones de tiempo, sobre el proceso de desarrollo, estándares, etc.

Mientras que los **requisitos funcionales del usuario** pueden ser frases muy generales sobre lo que la base de datos debería hacer (suelen expresarse como "objetivos" de la misma), los **requisitos funcionales del sistema** deben describir los servicios que hay que proporcionar con todo detalle.

Con los comentados requisitos o especificaciones se elaborará el denominado **documento de especificación de requisitos del *software*** (en nuestro caso **documento de especificación de requisitos de la base de datos**), el cual debe incluir, tanto los requisitos de usuario como los del sistema. Se debe tener en cuenta que dicho documento no es un elemento de diseño, ya que especifica **qué** es lo que se debe hacer, pero nunca el **cómo** se debe hacer.

Se trata de un documento de especial relevancia, el cual debe caracterizarse por no ser ambiguo, ser completo, fácil de verificar, consistente, y sencillo de modificar y de utilizar.

4.2.2 SEGUIMIENTO DE LOS REQUISITOS DE USUARIO

Haciendo nuevamente alusión a Roger S. Pressman, la ingeniería de requisitos es un proceso de descubrimiento, refinamiento, modelado y especificación. Se refinan en detalle los requisitos y el papel asignado al *software*, en general, y, en nuestro caso de estudio, a la base de datos.

Tanto el desarrollador como el cliente tienen un papel activo en la ingeniería de requisitos (un conjunto de actividades que son denominadas análisis). El cliente intenta replantear un sistema confuso, a nivel de descripción de datos, funciones y comportamiento, en detalles concretos. El desarrollador actúa como interrogador, como consultor, como persona que resuelve problemas y como negociador.

El análisis y la especificación de requisitos pueden parecer una tarea relativamente sencilla, pero suele ser una tarea altamente compleja y, como hablamos antes, muy importante para el correcto funcionamiento de la base de datos. El contenido de comunicación es muy denso. Abundan las ocasiones para malas interpretaciones o falta de información. Es muy probable que haya ambigüedad. El dilema al que se enfrenta el diseñador de la base de datos puede entenderse muy bien repitiendo la famosa frase de un cliente anónimo *"Sé que cree que entendió lo que piensa que dije, pero no estoy seguro de que se dé cuenta de que lo que escuchó no es lo que yo quise decir"*.

El análisis de requisitos se puede subdividir en cinco áreas de esfuerzo:

- Reconocimiento del problema
- Evaluación y síntesis
- Modelado
- Especificación
- Seguimiento

Todos los métodos de análisis se relacionan por un conjunto de principios operativos:

- Debe representarse y entenderse el dominio de la información de un problema.
- Deben definirse las funciones que debe realizar la base de datos.
- Debe representarse el comportamiento de la base de datos (como consecuencia de acontecimientos externos).
- Deben dividirse los modelos que representan información, función y comportamiento de manera que se descubran los detalles por capas (o jerárquicamente).
- El proceso de análisis debería ir desde la información esencial hasta el detalle de la implementación.

Además de los principios operativos mencionados anteriormente, se sugiere un conjunto de principios directrices para la ingeniería de requerimientos:

- Entender el problema antes de empezar a crear el modelo de análisis.
- Desarrollar prototipos que permitan al usuario entender cómo será la interacción hombre-base de datos.
- Registrar el orden y la razón de cada requerimiento.
- Usar múltiples planteamientos de requerimientos.
- Priorizar los requerimientos.
- Trabajar para eliminar la ambigüedad.

La función principal de un analista/diseñador de bases de datos es llevar a cabo las actividades necesarias para cumplir con las cinco áreas de esfuerzo descritas anteriormente. Para lo cual hace uso de las siguientes técnicas:

- Entrevistas
- Talleres
- Observación
- Encuestas
- Revisión documental

4.3 EJERCICIOS PROPUESTOS

- **1.** Con ayuda de Internet, haz una comparativa técnica de los siguientes DBMS: Oracle, MySQL y SQL Server.
- **2.** Leer el blog titulado "Cómo optimizar bases de datos en MySQL" (cuyo enlace está en este capítulo) y realiza un esquema con los aspectos más destacados en tu opinión.
- **3.** Elabora una hoja de seguimiento tipo para evaluar la evolución de un proyecto con bases de datos.
- **4.** Elabora una encuesta y una entrevista tipo para recoger los datos más relevantes en pos de poder llevar a cabo el análisis de una base de datos.

4.4 TEST DE CONOCIMIENTOS

1 Las principales fases del análisis de requisitos en un proyecto *software* con bases de datos son:
- **a)** Análisis de la situación de la compañía, estudio económico y definición de objetivos.
- **b)** Análisis, diseño, implementación y prueba de la base de datos.
- **c)** Estudio de la viabilidad económica, viabilidad legal y viabilidad técnico-operativa.
- **d)** Análisis de la situación de la compañía, identificación de los problemas y restricciones y definición de objetivos y del alcance.

2 Los objetivos del análisis estructurado son:
- **a)** Describir lo que se necesita, establecer la base para el diseño y definir requisitos para el seguimiento.
- **b)** Estudiar la viabilidad económica, la viabilidad legal y la viabilidad técnico-operativa, así como la estimación de plazos y coste.
- **c)** Describir los requerimientos y definir los requisitos para el seguimiento.
- **d)** Todas las respuestas son verdaderas.

3 Los datos de una base de datos:
- **a)** Están representados en el diagrama E-R.
- **b)** El diagrama DFD indica cómo evolucionan.
- **c)** El diagrama DTE los puede tener en cuenta como estímulos en sus transiciones.
- **d)** Todas las respuestas son verdaderas.

4 Señale la herramienta que no se emplea en el diseño conceptual de la base de datos:
- **a)** Diccionario de datos
- **b)** Diagrama Case
- **c)** Diagrama UML
- **d)** Encuesta

5 Según Berzal Galiano, los principales aspectos para la selección de un DBMS se clasifican en:
- **a)** Factores técnicos y factores no técnicos
- **b)** Factores técnicos, factores no técnicos y factores económicos
- **c)** Coste de adquisición, costes de mantenimiento, costes de personal y portabilidad
- **d)** Factores técnicos, costes de adquisición, costes de mantenimiento, costes de personal y portabilidad

6 Los factores no técnicos para la selección de un DBMS son:
- **a)** Coste de adquisición, costes de mantenimiento, costes de personal y portabilidad
- **b)** Coste de adquisición, costes de mantenimiento, costes de personal, portabilidad y adopción de nuevos protocolos de trabajo
- **c)** Coste de adquisición, costes de mantenimiento, costes de personal y adopción de nuevos protocolos de trabajo
- **d)** Coste de adquisición, costes de mantenimiento y costes de personal

7 Los siguientes indicadores son indicadores de calidad:
- **a)** Grado de cobertura temática, especialización temática y accesibilidad de los documentos originales
- **b)** Capacidad para realizar búsquedas exhaustivas y precisas, facilidad para juzgar la adecuación de los registros a la búsqueda deseada y esfuerzo en la recuperación.
- **c)** Grado de actualización, nivel de crecimiento y fiabilidad
- **d)** Tasa de pertinencia, tasa de exhaustividad y tasa de actualización

8 Los siguientes indicadores son indicadores de análisis documental:
- **a)** Grado de cobertura temática, especialización temática y accesibilidad de los documentos originales
- **b)** Capacidad para realizar búsquedas exhaustivas y precisas, facilidad para juzgar la adecuación de los registros a la búsqueda deseada y esfuerzo en la recuperación.
- **c)** Grado de actualización, nivel de crecimiento y fiabilidad
- **d)** Tasa de pertinencia, tasa de exhaustividad y tasa de actualización

9 Los siguientes indicadores son indicadores de presentación de los datos:
- **a)** Grado de cobertura temática, especialización temática y accesibilidad de los documentos originales
- **b)** Capacidad para realizar búsquedas exhaustivas y precisas, facilidad para juzgar la adecuación de los registros a la búsqueda deseada y esfuerzo en la recuperación.
- **c)** Grado de actualización, nivel de crecimiento y fiabilidad
- **d)** Tasa de pertinencia, tasa de exhaustividad y tasa de actualización

10 Los siguientes indicadores son indicadores de calidad de base de datos en búsquedas concretas:
- **a)** Grado de cobertura temática, especialización temática y accesibilidad de los documentos originales
- **b)** Capacidad para realizar búsquedas exhaustivas y precisas, facilidad para juzgar la adecuación de los registros a la búsqueda deseada y esfuerzo en la recuperación.
- **c)** Grado de actualización, nivel de crecimiento y fiabilidad
- **d)** Tasa de pertinencia, tasa de exhaustividad y tasa de actualización

11 ¿En cuántas áreas de esfuerzo se puede subdividir el análisis de requisitos?
- **a)** 3
- **b)** 6
- **c)** 5
- **d)** 4

12 La herramienta DBDesigner se utilizar en:
- **a)** El diseño de datos
- **b)** El diseño arquitectónico
- **c)** El diseño procedimental
- **d)** El diseño de la interfaz de usuario

13 El proceso de carga y optimización…
- **a)** Puede ser una fase larga y costosa debido a que se trabaja con enormes volúmenes de datos.
- **b)** Es una tarea valiosa ya que se pueden detectar errores de diseño.
- **c)** Existen utilidades gráficas que nos facilitan el trabajo.
- **d)** (a) y (b) son correctas.

14 El diagrama de flujo de datos realiza las siguientes tareas:
- **a)** Indicar cómo se transforman los datos según se avanza en el sistema de información y representar las funciones que utilizan dichos datos y los transforman.
- **b)** Indicar cómo se transforman los datos y cómo se comporta el sistema como consecuencia de sucesos externos.
- **c)** Indicar cómo se transforman, definen y se relacionan los datos según se avanza en el sistema de información y representar las funciones que utilizan dichos datos y los transforman.
- **d)** Indicar cómo se transforman, definen y se relacionan los datos y cómo se comporta el sistema como consecuencia de sucesos externos.

15 El modelo COCOMO sirve para
- **a)** El estudio de la viabilidad económica
- **b)** El estudio de la viabilidad técnico-operativa
- **c)** La estimación de plazos y costes
- **d)** Todas las afirmaciones son correctas.

5 Creación y diseño de bases de datos

5.1 ENFOQUES DE DISEÑO

Tal y como hemos comentado en el capítulo anterior, los pasos fundamentales que se deben llevar a cabo previamente a la implementación y carga de datos en un DBMS deben ser un exhaustivo análisis de requisitos donde se plasme, sobre un documento, todas las necesidades y características de la realidad susceptible de ser almacenada y tratada en la nueva base de datos.

Y será precisamente con esa información con la que se deberá realizar el diseño de nuestra base de datos, que, como también se ha comentado anteriormente, se deberá llevar a cabo en tres fases o niveles de abstracción diferentes: diseño conceptual, diseño lógico y diseño físico.

A lo largo de este apartado vamos a analizar las principales causas de errores en base de datos implementadas, originados en su totalidad por diseño incorrectos de estas, a la vez que analizaremos los dos principales enfoques para el correcto diseño de dichas base de datos, analizando sus ventajas y desventajas.

5.1.1 DISEÑOS INCORRECTOS. CAUSAS

RECUERDA

Para llevar a cabo la explicación de los errores más comunes y la forma de detectarlos, vamos a realizar una simulación de mala praxis de diseño sobre un ejemplo de base de datos que ya hemos analizado en diferentes ocasiones en este documento. Se trata del problema de almacenar los datos de un centro de formación que desea gestionar los datos de sus alumnos, junto con las asignaturas de las que están matriculados, que podemos ver en la Figura 2.5 (esto es, en el capítulo 2).

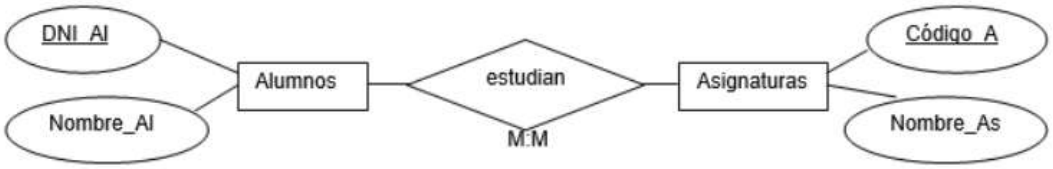


Figura 2.5. *Representación en el modelo E-R de dos entidades (con sus atributos) relacionadas*

Del mismo modo, antes de comenzar, aclararemos que hemos desarrollado esta simulación tomando la decisión de emplear un DBMS relacional, ya que es el más extendido, tanto comercialmente como en la literatura.

De este modo, al hablar de un modelo relacional, podemos concluir que diseñar una base de datos consistirá en diseñar qué tablas formarán parte de la misma, los atributos que contendrá cada tabla y las relaciones entre estas.

Aunque pudiera parecer una tarea no demasiado compleja el diseñar este tipo de bases de datos, debemos tener en cuenta que un mal diseño puede conducir a graves problemas de redundancia de datos, como vamos a ver a continuación, lo cual producirá a su vez, problemas de inconsistencia.

EJEMPLO 5.1

Una primera versión de diseño

Al ser una base de datos relativamente pequeña, podríamos estar tentados a almacenar toda la información en una sola tabla, ya que, de este modo, todos los datos estarían almacenados de una forma muy intuitiva y de fácil manejo.

DNI_Al	Nombre_Al	Código_As	Nombre_As
11111111A	Fran	001	Bases de Datos

La dificultad surgiría cuando, al empezar a trabajar con esa nueva base de datos, es decir, al introducir nuevos datos, actualizarlos o eliminar datos, nos podríamos encontrar con los siguientes problemas:

- **Problemas al insertar nuevos datos:** si quisiéramos dar de alta una nueva asignatura en el centro de formación, para ofertarla para un futuro, tendríamos que dejar vacíos (o marcados como *Null*) los valores correspondientes al alumnado que la cursa (ya que inicialmente no habría). Lo vemos en la fila 2 de la Figura 5.1.

DNI_Al	Nombre_Al	Código_As	Nombre_As
11111111ª	Fran	001	Bases de Datos
Null	Null	002	Redes
22222222B	Amalia	002	Redes
22222222B	Amalia	001	Bases de Datos
11111111ª	Fran	001	Bases de Datos

Figura 5.1. *Representación del diseño del Ejemplo 5.1 con datos*

Sin embargo, cuando comenzamos a dar de alta alumnos/as a la asignatura nueva, el DBMS no sabría si añadir el nombre del primer alumno a los dos campos vacíos o insertar una nueva fila o tupla a la tabla (lo cual a su vez provocaría una redundancia totalmente innecesaria) (fila 3 de la Figura 5.1).

Del mismo modo tendríamos problemas de redundancia de información cuando queramos duplicar un alumno en la tabla (porque se quiera matricular de varias asignaturas) o queramos duplicar una asignatura en la tabla (porque se matriculen varios alumnos) (fila 4 de la Figura 5.1). En cualquiera de los dos casos tendríamos que hacer una búsqueda secuencias para comprobar que el par *Alumno-Asignatura* no esté previamente insertado, porque estaríamos provocando duplicidades (fila 5 de la Figura 5.1).

Todo este proceso, como podemos imaginar, provocaría una pérdida en tiempo de respuesta en la base de datos.

- **Problemas al modificar datos:** partiendo de los datos de la Figura 5.2, tendríamos el caso de que los datos de *Amalia* estarían repetidos, ya que está matriculada de dos asignaturas.

DNI_Al	Nombre_Al	Código_As	Nombre_As
11111111ª	Fran	001	Bases de Datos
22222222B	~~Amalia~~ Amalia María	002	Redes
22222222B	Amalia	001	Bases de Datos

Figura 5.2. *Representación del diseño del Ejemplo 5.1 con datos (II)*

El problema que provocaría esta redundancia de información es que puede ser modificado solo en un fila, provocando la inconsistencia en la base de datos (por ejemplo, si el profesor de *Redes* modifica el nombre de alumna (fila 2 de la Figura 5.2).

- **Problemas al eliminar datos:** partiendo de los datos de la Figura 5.2, si quisiéramos eliminar a *Amalia* (o *Amalia María*) de la asignatura de *Redes*, ya que ha causado baja en dicha asignatura, automáticamente se eliminaría la fila al completo, por lo que dejaría de estar registrada la citada asignatura ya que *Amalia* era su única alumna activa.

EJEMPLO 5.2

Una segunda versión de diseño

Lo ideal sería utilizar una tabla para almacenar los datos de los *Alumnos*, otra para gestionar los datos de las distintas *Asignaturas* y, por último, una tercera tabla para almacenar la relación que se produce cuando *Un Alumno Estudia Una Asignatura*.

Tabla "Alumnos"

DNI_Al	Nombre_Al
11111111A	Fran
22222222B	Amalia

Tabla "Asignaturas"

Código_As	Nombre_As
001	Bases de Datos
002	Redes

Tabla "Estudian"

Código_As	DNI_Al
001	11111111A
001	22222222B
002	22222222B

Podemos analizar los posibles problemas expuestos anteriormente y nos daremos cuenta de que, en esta nueva versión, quedan solucionados.

Si analizamos las **causas** que nos llevaron a la primera versión del diseño, nos encontramos que no aplicamos en ningún caso la metodología explicada para la fase de diseño, o bien nos precipitamos, quizás por falta de experiencia o por falta de conocimientos acerca de la realidad que pretendíamos plasmar en una base de datos.

5.1.2 ENFOQUE DE ANÁLISIS. VENTAJAS Y DESVENTAJAS

El **enfoque de análisis** parte de entender la realidad a representar en una base de datos como una única tupla con múltiples y heterogéneos campos, a la cual, en progresivas fases, se le aplicará las diferentes reglas de normalización que estudiamos en la sección 3.5 "Teoría de la Normalización".

Por último, a partir de ese diseño lógico, se llevaría a cabo el diseño físico.

La principal **ventaja** del enfoque de análisis es la objetividad en el citado proceso de diseño y la capacidad de estructuración y automatización de dicho proceso de diseño.

Por el contrario, las **desventajas** del diseño analítico son diversas:

- Es más difícil hacer partícipe del diseño al usuario ya que es una metodología de diseño demasiado abstracta para él.
- En ocasiones la representación de todas las relaciones puede llegar a ser muy compleja.
- En ocasiones es difícil decidir en qué forma normal finalizar el proceso, llegando a provocar problemas de fragmentación y disminución de la eficiencia en tiempo de respuesta de la base de datos si la decisión no es acertada.
- Es más difícil especificar restricciones, herencias, diversas relaciones entre dos mismas tuplas, etc.

5.1.3 ENFOQUE DE SÍNTESIS. VENTAJAS Y DESVENTAJAS

Aunque en este libro se van a llevar a cabo actividades donde aplicar ambas metodologías de diseño, este enfoque sería quizás el más recomendado (según nuestra experiencia) para afrontar el diseño de bases de datos.

Dicho enfoque se basa en la metodología de análisis de requisitos, diseño conceptual, diseño lógico y diseño físico que hemos estado reiterando en estos últimos capítulos y, como ya se ha repetido, conlleva los siguientes pasos:

1. Tras el preceptivo análisis de requisitos, se llevará a cabo un diseño conceptual que se materializará, preferentemente, en un diagrama E-R (aspecto este explicado a lo largo del capítulo 2, denominado "Modelos conceptuales de bases de datos").
2. Transformar el diseño conceptual al diseño lógico, pasando el diagrama E-R al modelo relacional, tal y como se ha explicado en el apartado 3.2 "Estructura de modelo relacional".
3. Aplicar la teoría de la normalización a las tablas resultantes, para encontrar y corregir alguna tabla que no cumpla las reglas de la normalización. Este proceso suele concluir sin ninguna incidencia o con casos puntuales fácilmente subsanables.
4. Y, por último, a partir del diseño lógico, como en el enfoque anterior, se llevaría a cabo el diseño físico.

Las principales **ventajas** de este enfoque de diseño se sintetizan en:

- Es muy fácil hacer partícipe del diseño al usuario ya que el diagrama E-R es muy intuitivo y comprensible a niveles superficiales.
- La representación de las relaciones es muy ajustada a la realidad que se requiere y, en general, se obtienen relaciones que representan la realidad mejor que en la normalización.

Quizás como principal **desventaja** podríamos señalar que el diseñador debe tener un dominio bastante profundo sobre las técnicas de modelado Entidad-Relación, ya que, caso de no ser así, puede no llegar a diseños conceptuales completos o incluso correctos.

5.2 METODOLOGÍA DE DISEÑO

5.2.1 CONCEPTO

Como hemos estado hablando a lo largo de este capítulo, el diseño de las tablas que componen una base de datos relacional no es sencillo y, al mismo tiempo, es muy importante, debido a sus costosas penalizaciones ante errores.

Además, es necesario ser consciente de que las bases de datos son "entes vivos" y, por tanto, susceptibles al cambio. Por lo tanto, lo habitual es llevar a cabo el diseño de la base de datos, de forma gradual, siguiendo las fases que hemos estado comentando, las cuales son:

- Análisis de requisitos (aunque no forma parte del diseño es muy relevante en dicha tarea)
- Diseño conceptual
- Diseño lógico
- Diseño físico

A continuación, volveremos a referirnos a estas tres fases de diseño y hablaremos de la información de la que partimos, antes de iniciar el diseño, y qué información debemos generar, a modo de información de salida de este importante proceso.

5.2.2 DISEÑOS CONCEPTUAL, LÓGICO Y FÍSICO

Previamente a estas fases de diseño, en el **análisis de requisitos** o **estudio previo**, debemos determinar qué información se debe almacenar en la base de datos, sin prestar atención a cómo se van a almacenar dichos datos. Del mismo modo, se debe determinar en esta fase previa el uso que se le va a dar a esa información, esto es, su utilidad.

Toda la información que se necesita obtener se conseguirá mediante entrevistas y reuniones con los usuarios, principalmente, aunque existen otras técnicas que ya se han citado anteriormente. Del mismo modo, en ocasiones, también será necesario emplear el asesoramiento externo de especialistas en el tema a tratar.

Una vez determinados los requisitos de la base de datos, comenzaremos con la fase del **diseño conceptual**, en la que elaboraremos un esquema conceptual (en nuestra propuesta formativa planteamos el uso de un diagrama E-R) que incluya una descripción de la información que va a contener la base de datos, relacionando tanto los datos

(atributos), las entidades y las relaciones entre las mencionadas entidades. Esta herramienta propuesta (diagrama E-R) no es la única alternativa para esta fase de diseño, mas es una opción gráfica, sencilla de comprender por el usuario "inexperto" y, por tanto, en la que se puede hacer partícipe al usuario final en el diseño de la base de datos. Así pues, para mantener dicha sencillez y claridad, no se deberá incluir, en este esquema, detalles de implementación (definición de las tablas, declaración de atributos, etc.), ni terminología informática, para que los usuarios finales puedan colaborar en su elaboración.

Será en la fase de **diseño lógico** en la que se generará un esquema lógico más cercano a la implementación, el cual contendrá una descripción, a un nivel menor de abstracción, de las estructuras que permitirán almacenar la información. También, en este momento, ya se habrá seleccionado el DBMS, como vimos en el capítulo 4, por lo que el diseño lógico ya se orientará al DBMS, en nuestro caso, al modelo relacional (tuplas, relaciones, campos, etc.).

Por último, en la fase final se procederá a realizar el **diseño físico** de la base de datos, momento en el cual tendremos que adaptar el diseño lógico a la organización interna del DBMS escogido. Básicamente, la tarea consistirá en definir las estructuras de almacenamiento interno, el formato de los archivos del DBMS y los métodos de acceso a la información.

5.2.3 ENTRADAS Y SALIDAS DEL PROCESO

En el proceso de diseño de una base de datos, debemos utilizar una serie de datos o información, a la que podemos denominar **información de entrada**, y, del mismo modo, generaremos una **información de salida**, la cual servirá para la siguiente fase de implementación de la base de datos.

Comenzando con la información de entrada, esta será:

- **Requisitos de la base de datos y objetivos de la misma.** Dicha información se obtendrá a partir de entrevistas con los usuarios finales, análisis de la documentación susceptible de automatizar mediante la base de datos, objetivos de los usuarios, empresa, entidad… que la gestionará para con esta.
- **Requisitos de procesos**, es decir, las características que deben cumplir las aplicaciones que interactúen con la base de datos (por ejemplo, necesidad de base de datos distribuida, demandas de tiempos de respuesta, etc.).
- **Características del *hardware* y *software*** donde se implantará la base de datos o disponibilidad económica de adquisición y/o ampliación.
- **Especificaciones del DBMS**, es decir, sus características, lenguajes…

Por su parte, la información de salida que generará esta fase de diseño será:

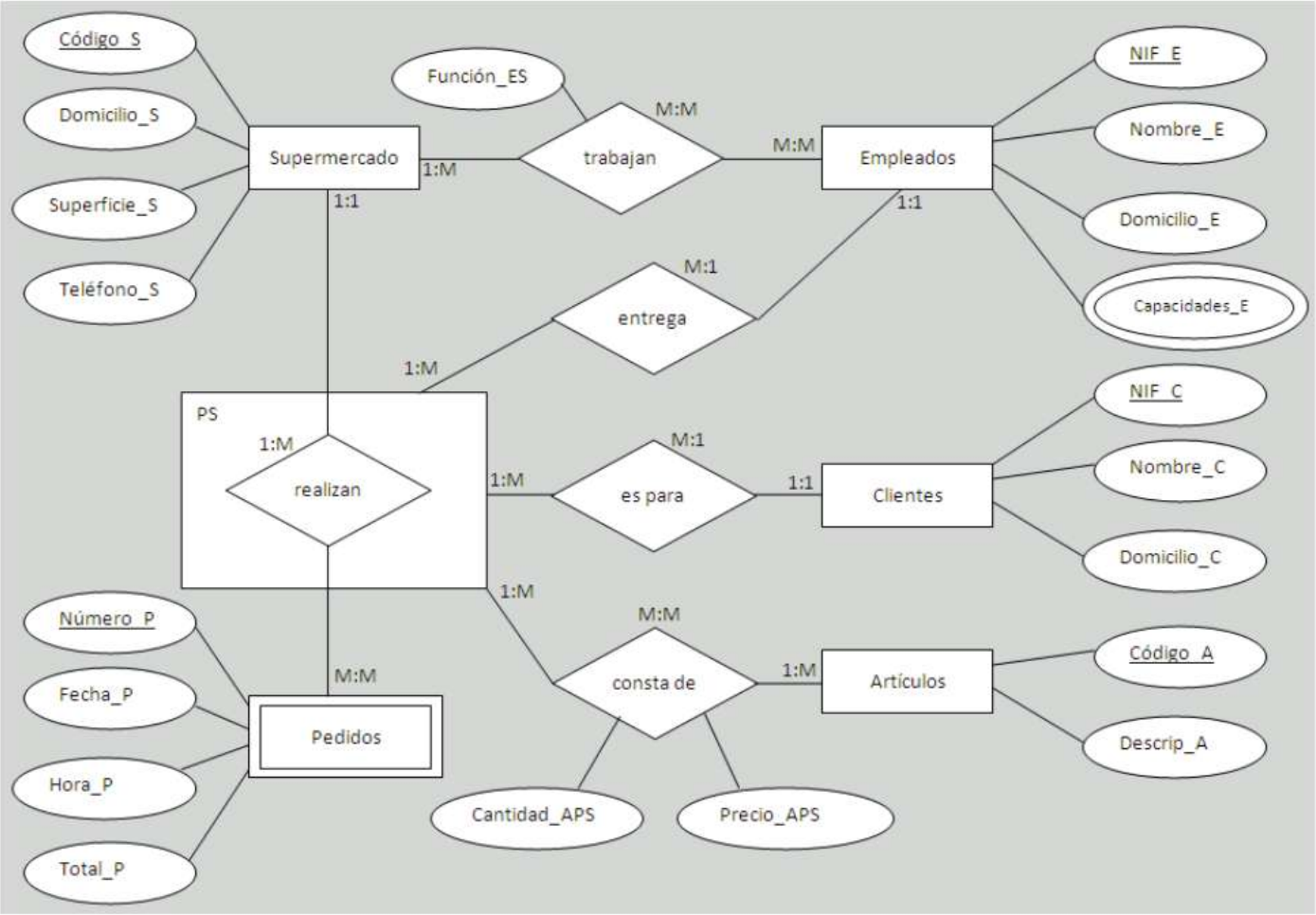
- **Estructura lógica de datos,** a través del esquema conceptual y lógico del que hemos estado hablando en el apartado anterior y en el capítulo 4.
- **Estructura de almacenamiento físico**, generada por la última fase de diseño (diseño físico de la base de datos), que contendrá las sentencias de definición de datos (DDL) (creación de tablas, índices, etc.), escritas en el lenguaje de programación propio del DBMS (como, por ejemplo, SQL).
- **Normativa de explotación**, en términos de reglas de confidencialidad, seguridad, integridad, etc., a tener presente en la implementación y posterior explotación de la base de datos.

5.3 ESTUDIO DEL DISEÑO LÓGICO DE UNA BASE DE DATOS RELACIONAL

Dado que ya hemos estudiado cómo pasar del modelo conceptual, concretamente de un diagrama E-R, al modelo lógico, para bases de datos relacionales (concretamente en el apartado 3.2 "Estructura del modelo relacional"), para afianzar estos conocimientos, en esta sección vamos a llevar a cabo un ejercicio guiado en el que aplicaremos esos conocimientos para desarrollar un ejemplo práctico de cierta envergadura.

EJERCICIO GUIADO 5.1

Dado el diagrama Entidad-Relación del Ejercicio guiado 2.1, a continuación se generará el modelo lógico (paso a tablas).



En primer lugar, pasaremos a tablas las entidades fuertes:

Supermercado (Código_S, Domicilio_S, Superficie_S, Teléfono_S)

Empleados (NIF_E, Nombre_E, Domicilio_E)

Clientes (NIF_C, Nombre_C, Domicilio_C)

Artículos (Código_A, Descrip_A)

En segundo lugar, nos centraremos en la entidad débil *Pedidos*.

Pedidos (Código_S, Número_P, Fecha_P, Hora_P, Total_P)

Esta tabla posee ciertas peculiaridades, propias de su tipología, como es la conformación de su clave primaria, compuesta de su discriminante y de la clave primaria de la entidad con la que está relacionada, en este caso, *Supermercado*.

Tenemos otra entidad débil en el atributo multivaluado *Capacidades_E:*

Capacidades_E (NIF_E, Capacidad_E)

Seguidamente, pasaremos a tablas las relaciones *M:M*, que son las únicas que forman tabla propia:

Trabajan (Código_S, NIF_E, Función_ES)

Consta_de (Código_S, Número_P, Código_A, Cantidad_APS, Precio_APS)

Del mismo modo, observamos la existencia de una agregación, que hemos denominado como PS, la cual, surge a partir de la relación realizan, que tiene cardinalidad 1:M y, por tanto, no forma tabla. De esta forma, dicha agregación tampoco pasará a tablas.

Pese a esto, aunque la agregación PS no pase a tablas, a efectos de otras relaciones, su clave primaria sería Código_S y Número_P, como observamos en la relación Consta_de.

Por último, para finalizar el paso a tablas, debemos analizar las relaciones de cardinalidad 1:M. Recordaremos que este tipo de relaciones no pasan a formar tabla sino que, para representar dicha relación, la clave de la entidad con cardinalidad "1" duplica su clave primaria y se la cede a la entidad "M".

No obstante, este ejemplo guiado plantea un caso particular en las relaciones *Entrega* y *Es para*, debido que las claves de las entidades *Empleados* y *Clientes*, según lo expuesto anteriormente, deberían duplicarse como atributos de *PS*. Sin embargo, al no pasar a tabla dicha agregación, los atributos referidos irán a la entidad *Pedidos*, quedando de la siguiente forma:

Pedidos (Código_S, Número_P, Fecha_P, Hora_P, Total_P, NIF_E, NIF_C)

Finalmente, el resultado del paso de tablas (diseño lógico) es el siguiente:

Supermercado (Código_S, Domicilio_S, Superficie_S, Teléfono_S)

Empleados (NIF_E, Nombre_E, Domicilio_E)

Capacidades_E (NIF_E, Capacidad_E)

Clientes (NIF_C, Nombre_C, Domicilio_C)

Artículos (Código_A, Descrip_A)

Pedidos (Código_S, Número_P, Fecha_P, Hora_P, Total_P, NIF_E, NIF_C)

Trabajan (Código_S, NIF_E, Función_ES)

Consta_de (Código_S, Número_P, Código_A, Cantidad_APS, Precio_APS)

5.4 EL DICCIONARIO DE DATOS: CONCEPTO Y ESTRUCTURA

La mayor parte de los DBMS del mercado poseen una base de datos del sistema, que contiene metadatos (esto es, datos referentes a los datos incluidos en la base de datos). Esto se denomina **diccionario de datos**, es decir, un fichero especial compuesto por todas las tablas pertenecientes a la base de datos, siendo el DBMS responsable de su definición y mantenimiento.

Las anotaciones contenidas en dicho diccionario de datos pueden ser:

- **Información de los elementos en cuestión:**
  - Nombre de los elementos
  - Tipos de elementos que almacena (real, entero, fecha, etc.)
  - Formato de los elementos
  - Tamaños de los elementos
  - Alias o nombre alternativo
  - Dónde se usa el elemento, indicando los procesos concretos
  - Descripción detallada del elemento
  - Información adicional (restricciones, limitaciones, condiciones, usuarios que pueden acceder a los elementos, etc.)
- **Información sobre los esquemas** correspondientes a los tres niveles de abstracción, así como lo relacionado a mapeos o correspondencias entre los citados esquemas.

Además de ello, se registra otro tipo de información:

- **Estadísticas de uso,** tales como la frecuencia de las transacciones y el número de acceso realizados a los objetos de la base de datos.
- **Espacio ocupado por la base de datos.**

Algunos de los beneficios que reporta el empleo de diccionarios de datos los podemos sintetizar en los siguientes:

- La información sobre los datos se puede almacenar de un modo centralizado. Ello ayuda a mantener el control sobre los mismos, como un recurso que son.

- El significado de los datos se puede definir, lo que ayudará a los usuarios a entender el propósito de los mismos.
- La comunicación se simplifica, ya que se almacena el significado exacto. El diccionario de datos también puede identificar al usuario o usuarios que poseen los datos o que acceden a ellos.
- Las redundancias y las inconsistencias se pueden identificar más fácilmente, ya que los datos están centralizados.
- Se puede tener un historial de los cambios realizados sobre la base de datos.
- El impacto que puede producir un cambio se puede determinar antes de que sea implementado, ya que el diccionario de datos mantiene información sobre cada tipo de dato, todas sus relaciones y todos sus usuarios.
- Se puede garantizar la seguridad y la integridad.
- Se puede proporcionar información para auditorías.

Para la descripción del contenido de los elementos incluidos en el diccionario de datos se emplea la siguiente sintaxis:

Elemento	Significado
=	Designa el contenido
+	Un elemento compuesto por la concatenación de varios simples
[|]	Un elemento compuesto de varias alternativas
$\{\}_{n\,m}$	Un conjunto compuesto por elementos simples repetidos un mínimo de "n" veces un máximo de "m" veces
( )	Elemento opcional

Figura 5.3. *Sintaxis utilizada en el diccionario de datos*

EJEMPLO 5.3

Veamos un ejemplo de definición de elementos en un diccionario de datos.

Nombre	Tipo	Formato	Unidad	Descripción
DNI_Al	Cadena (9)	$\{Digito\}_{9\,9}$+{Letra}		DNI del alumno
Nombre_Al	Cadena (50)	$\{Letra\}_{1\,50}$		Nombre del alumno
Peso_Al	Número real	$\{Digito\}_{1\,3}$+"," +$\{Digito\}_{2\,2}$	Kg.	
Altura_Al	Número real	$\{Digito\}_{1\,3}$+"," +$\{Digito\}_{2\,2}$	cm.	

5.5 ESTUDIO DEL DISEÑO DE BBDD Y DE LOS REQUISITOS DE USUARIO

RECUERDA

La importancia de llevar a cabo un análisis de requisitos de usuario como paso previo al diseño de la base de datos lo hemos comentado ampliamente en el apartado 5.2.2 "Diseños conceptual, lógico y físico".

5.6 EJERCICIOS PROPUESTOS

1. Una cadena de tiendas de muebles de cocina desea crear una base de datos para su gestión. Cada tienda tiene su CIF, dirección, fax y varios teléfonos. La tienda vende cocinas. Cada cocina tiene un número (único dentro de una tienda), una fecha de encargo y un precio total.

Una cocina es encargada por un cliente, del que tenemos su NIF, nombre, dirección y teléfono. La cocina corresponde a una composición, que tiene un modelo, un precio y una descripción. Cada composición incluye muchos elementos, con su código de elemento, descripción, color, precio y cantidad de cada elemento por composición.

La cocina puede tener elementos extras adicionales a la composición elegida.

Cada cocina es montada por uno o varios montadores, que no trabajan en las tiendas. Para cada montador, tenemos su NIF, nombre y domicilio. Necesitamos guardar la cantidad que cobra cada montador por el montaje de cada cocina.

Se pide lo siguiente:

- Hacer el diagrama Entidad-Relación.
- Pasar a tablas el citado diagrama Entidad-Relación.
- Unir todos los campos en una relación universal y normalizar hasta 3FN (explicando el proceso).

2. Una institución ferial quiere crear una base de datos para la gestión de las ferias de muestras que organiza.

Cada feria tiene un nombre, unas siglas (únicas) y unos sectores. Cada feria tiene varias ediciones, con el año, la fecha de inicio, la fecha de final y el horario.

En cada edición de cada feria, hay expositores que montan *stands*. De cada expositor tenemos el nombre, el CIF, el sitio web y su actividad. Cada expositor tiene un único *stand* en cada edición en la que participa. Cada *stand* tiene un número, una superficie y un precio, que pueden variar en cada edición.

También se guarda la información de los visitantes de las ediciones de las ferias. De cada visitante, se anota el NIF, nombre y domicilio. Además, cada *stand* va a tener un terminal en el que los visitantes van a pasar una tarjeta identificadora que registra su visita, anotando la fecha y la hora de la visita, y puede anotarse una observación. Cada visitante solo puede pasar una vez la tarjeta en un *stand*.

Se pide lo siguiente:

- Hacer el diagrama Entidad-Relación.
- Pasar a tablas el citado diagrama Entidad-Relación.
- Unir todos los campos en una relación universal y normalizar hasta 3FN (explicando el proceso).

3. Un parque de atracciones está interesado en crear una base de datos sobre el uso de sus atracciones y los operarios que trabajan en las mismas. Para ello, se cuenta con los datos siguientes.

- Para cada atracción, se guarda un código (que es único para cada atracción del parque), un nombre de la atracción, el año de inauguración, un clasificar que indica el tramo de edad al que va dirigida la atracción y un indicador de la peligrosidad de la atracción.
- Para cada atracción se desean almacenar los datos relativos a cada una de las vueltas que da, indicando, para cada vuelta, la fecha, hora y número de personas que han subido a la atracción.
- Para cada atracción, se desean registrar las posibles incidencias que se produzcan. Para cada incidencia se almacenará un código de la incidencia (que será único), la fecha en la que se ha producido la incidencia y una descripción de la incidencia.
- En las atracciones trabajan operarios. De cada operario se guarda su DNI, nombre, dirección, teléfono, fecha de nacimiento y las distintas funciones que es capaz de desempeñar en las diferentes atracciones. Dichas funciones son propias del operario, independientemente de la atracción en la que trabaje.
- Estos operarios trabajan en turnos y atracciones variables de forma que van rotando en la hora y en la atracción en la que trabajan. Cada vez que un operario realice una jornada de trabajo anotaremos la fecha, la hora de inicio, la hora de fin y la atracción en la que ha trabajado.

Se pide lo siguiente:

- Hacer el diagrama Entidad-Relación.
- Pasar a tablas el citado diagrama Entidad-Relación.
- Unir todos los campos en una relación universal y normalizar hasta 3FN (explicando el proceso).

5.7 TEST DE CONOCIMIENTOS

1 Señala la afirmación incorrecta:

a) El enfoque de análisis entiende la realidad como una única tupla con múltiples y homogéneos campos, a la que se aplicarán las reglas de normalización.
b) El enfoque de análisis se caracteriza por su objetividad, capacidad de estructuración y automatización del proceso.
c) El inconveniente del enfoque de análisis es su abstracción y la no participación del usuario en él.
d) El enfoque de análisis puede provocar fragmentación y disminución de la eficiencia.

2 Señala cuál no es una desventaja del uso del enfoque de análisis:
- **a)** Alto nivel de abstracción.
- **b)** Dependencia total del usuario.
- **c)** Las relaciones pueden llegar a ser muy complejas.
- **d)** Es difícil especificar restricciones, herencias, más de una relación entre dos tuplas, etc.

3 Los pasos a seguir para llevar a cabo el enfoque de síntesis son:
- **a)** Diseño conceptual, diseño lógico y diseño físico
- **b)** Diseño conceptual, diseño lógico, normalización y diseño físico
- **c)** Diseño conceptual, diseño lógico y normalización
- **d)** Diseño conceptual, diagrama E-R, normalización y diseño lógico

4 Señala cuál no es una ventaja del uso del enfoque de síntesis:
- **a)** Es muy fácil hacer partícipe al usuario.
- **b)** Es muy intuitivo y comprensible.
- **c)** El diseñador debe tener un dominio bastante profundo del modelado E-R.
- **d)** Es una representación muy ajustada a la realidad.

5 La información de entrada y salida del proceso de diseño consta de:
- **a)** Entrada: estructura lógica de datos, estructura de almacenamiento físico y normativa de explotación. Salida: requisitos de la base de datos, requisitos de procesos y especificaciones del DBMS.
- **b)** Salida: estructura lógica de datos, estructura de almacenamiento físico y normativa de explotación. Entrada: requisitos de la base de datos, requisitos de procesos y especificaciones del DBMS.
- **c)** Entrada: requisitos de la base de datos, requisitos de procesos, especificaciones del DBMS y características del *hardware* y *software*. Salida: estructura lógica de datos, estructura de almacenamiento físico y normativa de explotación.
- **d)** Ninguna de las anteriores.

6 ¿Cuál de los siguientes datos no están contenidos en un diccionario de datos?:
- **a)** Tamaño, formato, tipo y nombre de los elementos
- **b)** Información sobre los esquemas
- **c)** Estadísticas de uso
- **d)** Usuario que usan la base de datos.

7 $[\{Dígito\}_{9\,9} + \{Letra\}_{1\,1} \mid \{Letra\}_{1\,1} + \{Dígito\}_{8\,8} + \{Letra\}_{1\,1}]$ sirve para:
- **a)** Definir un tipo de identificador de una persona en España.
- **b)** Identificar un DNI de una persona.
- **c)** (a) y (b) son correctas.
- **d)** Ninguna de las anteriores.

8 Para simbolizar que un elemento es opcional:
a) Se coloca dicho elemento entre llaves.
b) Se le precede del carácter '|'.
c) Se coloca dicho elemento entre paréntesis.
d) Se coloca dicho elemento entre corchetes.

9 Para definir una cadena de caracteres alfanuméricos, pero que empiece por un carácter y tenga como máximo 10 dígitos usaremos la expresión:
a) $\{Letra\}_{10}$
b) $\{Letra\}_{1\ 10} + \{Dígito\}_{1\ 10}$
c) $\{Letra\}_{1\ 1} + \{Letra + Dígito\}_{1\ 9}$
d) Ninguna de las anteriores

10 Señala una afirmación que no se corresponda con una ventaja del empleo de los diccionarios de datos:
a) La información de datos está centralizada.
b) Ayuda a los usuarios a conocer el propósito de los datos.
c) No se dan redundancias e inconsistencias con su uso.
d) Se puede hacer un historial de los cambios de la base de datos.

11 Señala una afirmación que no se corresponda con una ventaja del empleo de los diccionarios de datos:
a) Se garantiza la seguridad y la integridad de la base de datos.
b) Proporciona información para auditorías.
c) Los metadatos están centralizados.
d) El impacto de los cambios se detecta rápidamente (en cuanto se produzcan).

12 Indica la afirmación incorrecta respecto a la estructura de almacenamiento físico:
a) Es un elemento que se genera en la fase de diseño de la base de datos.
b) Contiene las sentencias de definición de datos, es decir, el DCL.
c) Genera un código en el lenguaje de programación del DBMS.
d) Puede contener creaciones de índices en SQL.

13 Indica la afirmación incorrecta respecto a la normativa de explotación:
a) Genera las reglas de control de uso y privilegios a tener en cuenta en la implementación y explotación de la base de datos.
b) Genera las reglas de control de confidencialidad a tener en cuenta en la implementación y explotación de la base de datos.
c) Genera las reglas de control de seguridad a tener en cuenta en la implementación y explotación de la base de datos.
d) Genera las reglas de control de integridad a tener en cuenta en la implementación y explotación de la base de datos.

14 ¿En qué momento se puede detectar un diseño incorrecto?:
a) Al insertar nuevos datos.
b) Al modificar datos.
c) Al eliminar datos.
d) Todas las respuestas son ciertas.

15 Las fases del diseño de una base de datos son:
a) Análisis de requisitos, diseño conceptual, diseño lógico y diseño físico
b) Diseño conceptual, diseño lógico y diseño físico
c) Diseño lógico, diseño conceptual y diseño físico
d) Análisis de requisitos, diseño lógico, diseño conceptual y diseño físico

6 Lenguajes relacionales

6.1 TIPOS DE LENGUAJES RELACIONALES

Los DBMS han supuesto, sin duda, una importante contribución en el mundo de la informática. De estos, es el modelo relacional el que destaca, por lo que tiene sentido estudiar este tipo de modelo para la definición y manipulación de datos.

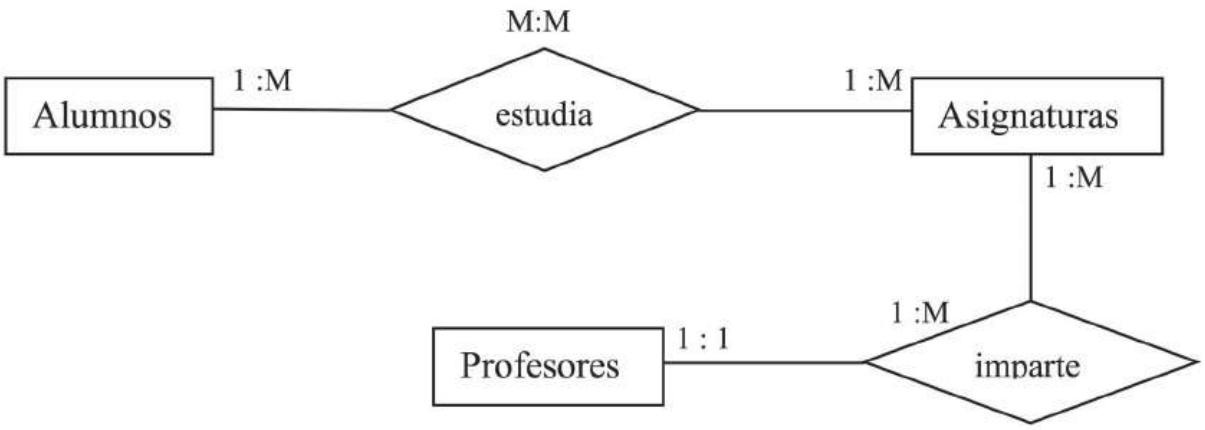
El citado modelo relacional cuenta con tres lenguajes formales:

- Álgebra relacional
- Cálculo relacional de tuplas
- Cálculo relacional de dominios

Seguidamente, expondremos una amplia explicación relacionada con estos lenguajes formales, a la vez que se presentará un análisis del lenguaje que, sobre este tipo de sistemas, ocupa la hegemonía en el mercado, como es SQL.

NOTA

Para todos los ejemplos de consultas que se van a ir mostrando a lo largo de este capítulo, emplearemos la base de datos presentada en el Ejemplo 3.1, con la incorporación de algunos atributos adicionales:



- **Alumnos** (DNI_Al, Nombre_Al, Apellido1_Al, Apellido2_Al, Edad_Al, Teléfono_Al, Dirección_Al, Ciudad_Al)
- **Estudian** (DNI_Al, Cód_As, Nota_Al_As, Convocatoria_Al_As)
- **Asignaturas** (Cód_As, Nombre_As, Créditos_As, Facultad_As, DNI_P)
- **Profesores** (DNI_P, Nombre_P, Apellido1_P, Apellido2_P, NCuenta_P, Teléfono_P, Especialidad_P, Dirección_P, Ciudad_P)

6.2 OPERACIONES EN EL MODELO RELACIONAL

Los sistemas gestores de bases de datos (DBMS) relacionales incluyen dos tipos de lenguajes:

- **Lenguaje para la definición de datos (DDL):** con el que definimos en el DBMS la estructura y los componentes de la base de datos (tablas, atributos, restricciones, etc.).

  Del mismo modo, también permite incluir restricciones de seguridad y definir vistas.

- **Lenguaje para la manipulación de datos (DML):** que puede ser utilizado para realizar consultas y modificaciones en la base de datos.

  Por su parte, podemos distinguir tres tipos de operaciones de modificación para bases de datos:

  – **Inserción:** consiste en añadir una o más tuplas a una determinada tabla de la base de datos.
  – **Borrado:** consisten en eliminar una o más tuplas a una determinada tabla de la base de datos.
  – **Actualización:** consiste en modificar el contenido de uno o varios atributos a una o más tuplas de una determinada tabla de la base de datos.

6.3 ÁLGEBRA RELACIONAL

El **álgebra relacional** constituye un lenguaje procedimental que cuenta con un conjunto de operaciones básica para llevar a cabo consultas sobre una base de datos. En este lenguaje se basa el lenguaje de SQL para consultas.

RECUERDA

La diferencia entre lenguajes procedimentales y lenguajes no procedimentales la estudiamos en la sección 1.3.8 "Lenguaje de manejo de bases de datos. Tipos".

6.3.1 CLASIFICACIÓN DE OPERADORES

Existen en la literatura diversas formas de clasificar los operadores del álgebra relacional, en base de diferentes características, por lo que comentaremos, en este apartado, las formas de clasificación más extendidas:

- Una primera forma de clasificación podría ser en base al **número de relaciones implicadas en la operación**. De este modo, tenemos:

  – **Operaciones unarias:** cuando en la operación está implicada una única relación. Las operaciones representativas de este tipo son la selección (σ) y la proyección (π). Otro operador binario menos extendido es el operador de renombrado (ρ).

- **Operaciones binarias:** cuando en la operación están implicadas dos relaciones. Las operaciones binarias son las relacionadas con la teoría de conjuntos (unión (U), intersección (∩) y diferencia (-)) y el producto cartesiano (X).

- Y otra alternativa de clasificación podría ser en base a si el operador se corresponde o no a los de la teoría de conjuntos, teniendo, a su vez:
  - **Operaciones conjuntistas**, es decir, los relacionados a la teoría de conjuntos, entre los que se encuentran unión (U), intersección (∩), diferencia (-) y el producto cartesiano (X).
  - **Operaciones específicamente relacionales**, entre los que tenemos las operaciones de la selección (σ) y la proyección (π).

También existen otras formas de clasificación menos usuales como la de **operaciones primitivas** y **operaciones derivadas**, entre otras.

6.3.2 DENOMINACIÓN DE ATRIBUTOS

Una relación nominada se define explícitamente sobre un esquema relacional y se expresa indicando el nombre de dicha relación y, para cada atributo, su nombre y el nombre de su dominio:

$$R(A_1{:}D_1, A_2{:}D_2, \ldots, A_n{:}D_n) \text{ o, también } R(\{A_i{:}D_j\})$$

De este modo, a la relación R se le asocian los siguientes nombres calificados de atributos:

$$R.A_1, R.A_2, \ldots, R.A_n$$

Por tanto, para **denominar cada atributo**, debemos usar su nombre calificado (anteponiendo el nombre de la relación o a su alias) o bien su nombre sin calificar (en caso de que no existan ambigüedades).

6.3.3 RELACIONES DERIVADAS

Se dice que una **relación es derivada** si se define mediante una expresión del álgebra relacional. Es decir, el resultado de aplicar un operador o programa algebraico a una base de datos es una relación derivada; por tanto, una consulta también es una relación derivada.

6.3.4 OPERACIONES PRIMITIVAS: SELECCIÓN, PROYECCIÓN, PRODUCTO CARTESIANO, UNIÓN Y DIFERENCIA

En esta sección vamos a estudiar las operaciones primitivas haciendo un pequeño resumen de su función, expresando su sintaxis y, por último, poniendo un ejemplo de consulta en el que se utilice dicha operación.

Selección

Esta operación consiste en seleccionar las tuplas (filas) de una relación (tabla) que satisfagan una determinada condición o condiciones.

Sintaxis: $\sigma_{\text{condición}}(\text{relación})$

NOTA

En la condición podemos emplear los operadores relacionales (>, <, >=, <=, =, <>) y los conectores lógicos (and (∧), or (∨), not (¬)) para concatenar varias condiciones.

EJEMPLO 6.1

Devolver los profesores que tengan como uno de sus apellidos "Pérez" y sean de la especialidad de "Historia".

$\sigma_{\text{Especialidad\_P="Historia"} \wedge (\text{Apellido1\_P="Pérez"} \vee \text{Apellido2\_P="Pérez"})}(\text{Profesores})$

Proyección

Esta operación unaria permite tomar un conjunto de atributos (columnas) de una relación (tabla).

Sintaxis: $\pi_{\text{atributos}}(\text{relación})$

NOTA

Para especificar varios atributos, los separamos por el operador *coma*: *","*.

EJEMPLO 6.2

Realizar un listado con los nombres y apellidos de los alumnos, junto con su número de teléfono.

$\pi_{\text{Nombre\_Al, Apellido1\_Al, Apellido2\_Al, Teléfono\_Al}}(\text{Alumnos})+$

NOTA

Como hemos comentado anteriormente, **estos operadores se pueden combinar**, para llevar a cabo consultas más específicas.

EJEMPLO 6.3

Devolver un listado con los nombres, apellidos y direcciones de los profesores que impartan *Historia*.

$\pi_{\text{Nombre\_P, Apellido1\_P, Apellido2\_P, Dirección\_P}}$ ($\sigma_{\text{Especialidad\_P= "Historia"}}$ (Profesores))

Producto cartesiano

Esta operación binaria (sobre dos relaciones) obtiene una nueva relación, que contiene elementos combinados de ambas relaciones. El número de tuplas de la nueva relación es igual a *nxm*, siendo *n* y *m* las tuplas de las relaciones originales.

Sintaxis: relación_1 X relación_2

EJEMPLO 6.4

Si realizamos la operación producto cartesianos a dos de las tablas con las que estamos trabajando en este capítulo, concretamente a *Asignaturas* y *Profesores* en un momento en el que poseen la siguiente información:

Tabla "Asignaturas"

Código_As	Nombre_As	...	DNI_P
001	Bases de Datos	...	11111111ª
002	Redes	...	22222222B

Tabla "Profesores"

DNI_P	Nombre_P	...	Dirección_P
11111111ª	Francisco Javier	...	Calle Almería, 1
22222222B	Amalia	...	Calle Granada, 2

El resultado del producto cartesiano sería:

Tabla "Asignaturas X Profesores"

Código_As	Nombre_As	...	DNI_P	DNI_P	Nombre_P	...	Dirección_P
001	Bases de Datos	...	11111111A	11111111A	Francisco Javier	...	Calle Almería, 1
001	Bases de Datos	...	11111111A	22222222B	Amalia	...	Calle Granada, 2
002	Redes	...	22222222B	11111111A	Francisco Javier	...	Calle Almería, 1
002	Redes	...	22222222B	22222222B	Amalia	...	Calle Granada, 2

NOTA

En el caso de que solo deseáramos mostrar las asignaturas impartidas por el profesor *Francisco Javier*, tendríamos que combinar los siguientes operadores:

$$\sigma_{\text{Nombre\_P="Francisco Javier"}} (\text{Asignaturas X Profesores})$$

Siendo el resultado de la consulta:

Código_As	Nombre_As	...	DNI_P	NI_P	Nombre_P	...	Dirección_P
001	Bases de Datos	…	11111111A	11111111A	Francisco Javier	…	Calle Almería, 1
002	Redes	…	22222222B	11111111A	Francisco Javier	…	Calle Almería, 1

Sin embargo, podemos comprobar que en las columnas *DNI_P* hay DNI que no son iguales, por lo que el resultado ofrece una incongruencia al mostrar información de una asignatura que imparte la profesora *Amalia*, junto a los datos del profesor *Francisco Javier.*

Para solventar esta ambigüedad, tendremos que añadir una nueva condición al operador selección, resultando la siguiente consulta:

$\sigma_{\text{Asignaturas.DNI\_P = Profesores.DNI\_P} \wedge \text{ Nombre\_P = "Francisco Javier"}}$ (Asignaturas X Profesores)

Unión

Esta operación muestra el resultado de la unión de los elementos de dos relaciones.

Para ello, las dos relaciones deben ser compatibles respecto la unión, o unión-compatibles.

NOTA

Para que dos relaciones sean **compatibles respecto la unión** o **unión-compatibles** deben:

- Tener el mismo número de atributos.
- Los dominios de esos atributos deben ser compatibles dos a dos.

Sintaxis: $\text{relación}_1 \text{ U } \text{relación}_2$

EJEMPLO 6.5

Devolver un listado con los nombres y teléfonos de todas las personas de la base de datos, es decir, tanto *Alumnos* como *Profesores*.

$(\pi_{Nombre\_P,\ Apellido1\_P,\ Apellido2\_P,\ Teléfono\_P}$ (Profesores))

U

$(\pi_{Nombre\_Al,\ Apellido1\_Al,\ Apellido2\_Al,\ Teléfono\_Al}$ (Alumnos))

Diferencia

Esta operación nos devuelve las tuplas que están presentes en la primera relación y que no están presentes en la segunda.

Para ello, las dos relaciones deben ser compatibles respecto la unión, o unión-compatibles.

Sintaxis: $relación_1$ - $relación_2$

EJEMPLO 6.6

Mostrar un listado de todos los profesores de la especialidad *Historia* pero que no sean de *Almería*.

$(\sigma_{Especialidad\_P = \text{“Historia”}}$ (Profesores)) - $(\sigma_{Ciudad\_P = \text{“Almería”}}$ (Profesores))

6.3.5 OTRAS OPERACIONES: INTERSECCIÓN, *JOIN*, DIVISIÓN, ETC.

Intersección

Esta operación binaria nos devuelve las tuplas que aparecen en ambas relaciones sobre las que se aplica.

Para ello, las dos relaciones deben ser compatibles respecto la unión, o unión-compatibles.

Sintaxis: $relación_1 \cap relación_2$

EJEMPLO 6.7

Mostrar un listado de todos los profesores de la especialidad *Historia* pero que sean de *Almería*.

$(\sigma_{Especialidad\_P = \text{“Historia”}}$ (Profesores)) $\cap$ $(\sigma_{Ciudad\_P = \text{“Almería”}}$ (Profesores))

Producto natural, combinación, *join* o reunión

Esta operación obtiene, a partir de dos relaciones, una nueva relación formada por todas las tuplas que resultan de concatenar (producto cartesiano) tuplas de la primera relación con tuplas de la segunda pero, además, que cumplen una condición de combinación especificada.

Sintaxis: $\text{relación}_1 |\text{X}|_{\text{condición de combinación}} \text{relación}_2$

O, también,

$(\text{relación}_1 * \text{relación}_2)_{\text{condición de combinación}}$

EJEMPLO 6.8

La diferencia entre este operador y el ejemplo del producto cartesiano del Ejemplo 6.4 sería la inclusión de la condición, por la cual se obtendría el siguiente resultado:

Tabla "Asignaturas"

Código_As	Nombre_As	...	DNI_P
001	Bases de Datos	...	11111111ª
002	Redes	...	22222222B

Tabla "Profesores"

DNI_P	Nombre_P	...	Dirección_P
11111111ª	Francisco Javier	...	Calle Almería, 1
22222222B	Amalia	...	Calle Granada, 2

En este caso, el resultado del producto natural sería:

Tabla "Asignaturas * Profesores"

Código_As	Nombre_As	...	DNI_P	DNI_P	Nombre_P	...	Dirección_P
001	Bases de Datos	...	11111111A	11111111ª	Francisco Javier	...	Calle Almería, 1
001	Bases de Datos	...	11111111A	22222222B	Amalia	...	Calle Granada, 2
002	Redes	...	22222222B	11111111ª	Francisco Javier	...	Calle Almería, 1
002	Redes	...	22222222B	22222222B	Amalia	...	Calle Granada, 2

Y, además, se eliminaría la columna repetida, es decir:

Código_As	Nombre_As	...	DNI_P	Nombre_P	...	Dirección_P
001	Bases de Datos	...	11111111A	Francisco Javier	...	Calle Almería, 1
002	Redes	...	22222222B	Amalia	...	Calle Granada, 2

División

Sean R y S dos relaciones con esquemas relacionales:

$R.A_1, ..., R.A_n, R.B_1, ..., R.B_m$

$S.B_1, ..., S.B_m$ respectivamente.

La operación **división** genera como resultado otra relación cuyo esquema es:

$Q.A_1, ..., Q.A_n$

Y su contenido son las tuplas tomadas a partir de las de r(R) tales que su valor está asociado en r(R) con **todos** los valores que están en s(S).

NOTA

Explicando este concepto de otra forma más visual, si tenemos las relaciones R y S, el resultado de la división Q = R/S para los valores:

R.A	R.B
a_1	b_1
a_1	b_2
a_2	b_1
a_2	b_2
a_2	b_3
a_2	b_4
a_3	b_1
a_3	b_3

S.B
b_1
b_2
b_3

sería Q(A) = a2

Sintaxis: relación$_1$/ relación$_2$

Renombrado

La operación **renombrado** cambia el nombre de una relación, usándose en operaciones que tenemos que referirnos a esa misma relación.

Sintaxis: ρ_{nuevo_nombre}(relación)

EJEMPLO 6.9

Mostrar los profesores que tengan la misma especialidad que *Amalia*.

$\sigma_{Profesores.Especialidad\_P=P.Especialidad\_P}(Profesores\ X\ \sigma_{Nombre=\text{“Amalia”}}(\rho_P\ (Profesores)))$

EJERCICIO GUIADO 6.1

Dado el diagrama Entidad-Relación del Ejercicio guiado 2.1 pasado a tablas en el Ejercicio guiado 5.1, tenemos el siguiente diseño lógico:

Supermercado (Código_S, Domicilio_S, Superficie_S, Teléfono_S)
Empleados (NIF_E, Nombre_E, Domicilio_E)
Capacidades_E (NIF_E, Capacidad_E)
Clientes (NIF_C, Nombre_C, Domicilio_C)
Artículos (Código_A, Descrip_A)
Pedidos (Código_S, Número_P, Fecha_P, Hora_P, Total_P, NIF_E, NIF_C)
Trabajan (Código_S, NIF_E, Función_ES)
Consta_de (Código_S, Número_P, Código_A, Cantidad_APS, Precio_APS)

A esta base de datos le podemos realizar las siguientes consultas:

1. **Nombres de clientes que han comprado *langosta*:**

   $\pi_{Nombre\_C}(((((((Clientes * Pedidos)_{Clientes.NIF\_C=Pedidos.NIF\_C}) *$

   $Consta\_de)_{Pedidos.Código\_S=Consta\_de.Código\_S \wedge Pedidos.Número\_P=Consta\_de.\ Número\_P}) *$

   $(\sigma_{Descrip\_A=\text{“langosta”}}\ (Artículos)))_{Consta\_de.Código\_A=Artículos.Código\_A}))$

2. **Nombre y domicilio de los empleados que no han entregado ningún pedido:**

   $(\pi_{Nombre\_E,\ Domicilio\_E}\ (Empleados)) -$

   $(\pi_{Nombre\_E,\ Domicilio\_E}((Empleados * Pedidos)_{Empleados.NIF\_E=Pedidos.NIF\_E}))$

3. **Nombres y domicilios de empleados que, a su vez, son clientes del supermercado (han hecho algún pedido en alguna ocasión):**

   $(\pi_{Nombre\_E,\ Domicilio\_E}(Empleados)) \cap$

   $(\pi_{Nombre\_C,\ Domicilio\_C}\ ((Clientes * Pedidos)_{Clientes.NIF\_C=Pedidos.NIF\_C}))$

4. **Fechas de pedidos que incluyen *tomates* y que han sido ordenados por *Amalia Gallegos*:**

   $\pi_{Fecha\_P}(((((((\sigma_{Nombre\_C=\text{“Amalia Gallegos”}}\ (Clientes)) * Pedidos)_{Clientes.NIF\_C=Pedidos.NIF\_C})$

   $* Consta\_de)_{Pedidos.Código\_S=Consta\_de.Código\_S \wedge Pedidos.Número\_P=Consta\_de.\ Número\_P}) *$

   $(\sigma_{Descrip\_A=\text{“tomates”}}\ (Artículos)))_{Consta\_de.Código\_A=Artículos.Código\_A}))$

5. **Función con la que trabaja el empleado *Francisco Javier Martínez* en los supermercados en lo que ha trabajado que tengan una superficie superior a 1000 m^2:**

$\pi_{Código\_S, Función\_ES}((((((\sigma_{Nombre\_E= \text{“Francisco Javier Martínez”}} (Empleados)) * Trabajan)_{Empleados.NIF\_E=Trabajan.NIF\_E}) * (\sigma_{Superficie\_S>1000}(Supermercado)))_{Supermercado.Código\_S=Trabajan.Código\_S}))$

6.4 CÁLCULO RELACIONAL

El **cálculo relacional** está basado en el cálculo de predicados de la lógica matemática. Proporciona una notación que permite formular la definición de la relación donde están los datos que responden a la consulta en términos de las relaciones almacenadas. Esta definición no describe un procedimiento, por lo tanto, se dice que los lenguajes basados en el cálculo relacional son **lenguajes** no procedimentales.

En este capítulo vamos a estudiar las dos clases de cálculo relacional más extendidas en la literatura, concretamente:

- Cálculo relacional orientado a dominios
- Cálculo relacional orientado a tuplas

Al igual que el álgebra relacional, el cálculo relacional tampoco es amigable para el usuario, por esta razón se han desarrollado lenguajes comerciales orientados al usuario, y que se basan en alguno de los lenguajes formales. Más adelante los estudiaremos.

6.4.1 CÁLCULO RELACIONAL ORIENTADO A DOMINIOS

El **cálculo relacional orientado a dominios** es un lenguaje que emplea variables de dominios, las cuales toman sus valores en el dominio de un atributo, en lugar de variables de tupla, como la siguiente clase de cálculo relacional a estudiar.

La *sintaxis* de este tipo de consultas se basa en la siguiente expresión:

$\{<x_1, \ldots, x_n> / P(x_1, \ldots, x_n)\}$

Donde:

- $x_1, \ldots, x_n$ son variables de dominio.
- $P(x_1, \ldots, x_n)$ es una fórmula.

Al igual que en la orientación a tuplas, una fórmula, a su vez, se compone de átomos, los cuales toman una de las siguientes formas:

- $<x_1, \ldots, x_n> \in r$, siendo *r* una relación de *n* atributos
- x *operación* y
- x *operación constante*

Teniendo que:

- *x* e *y* son variables de dominio.
- Las *operaciones* pueden ser >, >=, <, <=, =, ≠.

Por otro lado, tenemos que una fórmula debe seguir las siguientes reglas:

- Un átomo es una fórmula.
- Si P es una fórmula, también lo es ¬P y (P).
- Si P_1 y P_2 son fórmulas, también lo son:
  - $P_1 \wedge P_2$
  - P_1 v P_2
  - $P_1 \Rightarrow P_2$
- Si P(x) es una fórmula, siendo *x* una variable de dominio, también lo es:
  - ∃x (P(x))
  - ∀x (P(x))

EJEMPLO 6.10

Veremos, a través de varias consultas de ejemplos, algunas de las principales operaciones que hemos estudiado en álgebra relacional.

Consulta del Ejemplo 6.1 (selección): devolver los profesores que tengan como uno de sus apellidos Pérez y sean de la especialidad de Historia.

{<p, q, r, s, t, u, v, w, x> / < p, q, r, s, t, u, v, w, x >∈ Profesores ∧ v= “Historia” ∧
(r = “Pérez” ∨ s = “Pérez”)}

Consulta del Ejemplo 6.2 (proyección): realizar un listado con los nombres y apellidos de alumnos, junto con su número de teléfono.

{<q, r, s, u> / ∃p,t,v,w (<p, q, r, s, t, u, v, w>∈Alumnos)}

Consulta del Ejemplo 6.3 (selección y proyección): devolver un listado con los nombres, apellidos y direcciones de los profesores que impartan *Historia*.

{<q, r, s, w> / ∃ p,t,u,v,x (< p, q, r, s, t, u, v, w, x >∈ Profesores ∧ v= “Historia”)}

Consulta del Ejemplo 6.5 (unión): devolver un listado con los nombres y teléfonos de todas las personas de la base de datos, es decir, tanto *Alumnos* como *Profesores*.

{<q, r, s, u> / (∃ p,t,v,w,x (< p, q, r, s, t, u, v, w, x >∈ Profesores)) U (∃ pa,ta,va,wa (<pa,

q, r, s, ta, u, va, wa>∈ Alumnos))}

Consulta del Ejemplo 6.6 (diferencia): mostrar un listado de todos los profesores de la especialidad *Historia* pero que no sean de *Almería*.

{< p, q, r, s, t, u, v, w, x > / ((<p, q, r, s, t, u, v, w, x>∈ Profesores ∧ v= "Historia")∧
¬(<p, q, r, s, t, u, v, w, x>∈ Profesores ∧ x= "Almería"))}

Consulta del Ejemplo 6.7 (intersección): mostrar un listado de todos los profesores de la especialidad *Historia* pero que sean de *Almería*.

{< p, q, r, s, t, u, v, w, x > / ((<p, q, r, s, t, u, v, w, x>∈ Profesores ∧ v= "Historia") ∧
(<p, q, r, s, t, u, v, w, x>∈ Profesores ∧x= "Almería"))}

Consulta nueva (producto natural): consultar el nombre y apellidos de los profesores que imparten la asignatura de *bases de datos*.

{<q, r, s > / ∃ p(<p, q, r, s, t, u, v, w, x>∈ Profesores ∧ ∃pa, qa, ra, sa(<pa, qa, ra, sa, p>∈
Asignaturas ∧ qa= "bases de datos"))}

EJERCICIO GUIADO 6.2

Si realizamos en cálculo relacional orientado a dominios las consultas expuestas en el Ejercicio guiado 6.1, tenemos que:

1. **Nombres de clientes que han comprado *langosta*:**

   $\{<t_2> / \exists t_1,t_3 (<t_1,t_2,t_3> \in$ Clientes $\wedge \exists p_1, p_2, p_3, p_4, p_5, p_6 (<p_1, p_2, p_3, p_4, p_5,p_6, t_1>$
   $\in$ Pedidos) $\wedge \exists s_1, s_2, s_3 (<p_1, p_2, s_1, s_2, s_3> \in$ Consta_de) $\wedge \exists q_1 (<s_1, q_1> \in$
   Artículos $\wedge\ q_1 =$ "langosta"))$\}$

2. **Nombre y domicilio de los empleados que no han entregado ningún pedido:**

   $\{<t_2,t_3> / \exists t_1 (<t_1,t_2,t_3> \in$ Empleados $\wedge$
   $\neg\exists p_1, p_2, p_3, p_4, p_5, p_6 (<p_1, p_2, p_3, p_4, p_5, t_1, p_6> \in$ Pedidos))$\}$

3. **Nombres y domicilios de empleados que, a su vez, son clientes del supermercado (han hecho algún pedido en alguna ocasión):**

   {<t2,t3> / (∃ t1 (<t1,t2,t3> ∈ Empleados)) U (∃ q2, q3 (<q1, q2, q3 >∈ Clientes ∧
   ∃p1, p2, p3, p4, p5, p6 (<p1, p2, p3, p4, p5, p6, q1> ∈ Pedidos))}

4. **Fechas de pedidos que incluyen *tomates* y que han sido ordenados por *Amalia Gallegos*:**

   $\{<t_3> / \exists t_1, t_2, t_4, t_5, t_6, t_7 (<t_1, t_2, t_3, t_4, t_5, t_6, t_7> \in$ Pedidos $\wedge \exists p_1, p_2 (<t_7, p_1, p_2> \in$
   Clientes $\wedge\ p_1 =$ "Amalia Gallegos") $\wedge \exists s_1, s_2, s_3 (<t_1, t_2, s_1, s_2, s_3> \in$ Consta_de) $\wedge$
   $\exists q_1 (<s_1, q_1> \in$ Artículos $\wedge\ q_1 =$ "tomates"))$\}$

5. **Función con la que trabaja el empleado *Francisco Javier Martínez* en los supermercados en lo que ha trabajado que tengan una superficie superior a 1000 m^2:**

   $\{<t_3> / \exists t_1, t_2 (<t_1, t_2, t_3> \in$ Trabajan $\wedge \exists p_1,p_2 (<t_2, p_1, p_2> \in$ Empleados $\wedge$
   $p_1 =$ "Francisco Javier Martínez") $\wedge \exists s_1, s_2, s_3 (<t_1, s_1, s_2, s_3> \in$ Supermercado $\wedge$
   $s_2 > 1000))\}$

6.4.2 CÁLCULO RELACIONAL ORIENTADO A TUPLAS

La principal diferencia del **cálculo relacional orientado a tuplas** frente al anteriormente comentado se centra fundamentalmente en su sintaxis, que se basa en tuplas y no en dominios.

Concretamente, la *sintaxis* utilizada sigue el patrón siguiente:

{ t / P(t) }

Donde:

- *t* es una variable libre.
- *P* es una fórmula o predicado en la que aparecen variables libres y variables de tupla.

El resultado es el conjunto de tuplas *t* que verifican el predicado *P(t)*.

A su vez, para referirnos a un atributo *A* de una relación *t*, usaremos la siguiente *sintaxis*: *t[A]*.

También, al igual que en el cálculo relacional de dominios, se puede utilizar el cuantificador existencial y el cuantificador universal.

Una fórmula se compone de átomos, que, a su vez, toman una de las siguientes formas:

- $s \in r$
- s[atributo] *operación* r[atributo]
- s[atributo] *operación constante*

Teniendo que:

- *r* y *s* son relaciones.
- Las *operaciones* pueden ser >, >=, <, <=, =, ≠.

Por otro lado, tenemos que una fórmula se construye mediante átomos, siguiendo las siguientes reglas:

- Un átomo es una fórmula.
- Si P es una fórmula, también lo es ¬P y (P).
- Si P_1 y P_2 son fórmulas, también lo son:
  - $P_1 \wedge P_2$
  - $P_1 \vee P_2$
  - $P_1 \Rightarrow P_2$
- Si P(s) es una fórmula, también lo es:
  - $\exists s \in r\ (P(s))$
  - $\forall s \in r(P(s))$

EJEMPLO 6.11

Veremos, a través de varias consultas de ejemplos, algunas de las principales operaciones que hemos estudiado en álgebra relacional.

Consulta del Ejemplo 6.1 (selección): devolver los profesores que tengan como uno de sus apellidos Pérez y sean de la especialidad de Historia.

{t / t ∈ Profesores ∧ t[Especialidad_P] = "Historia" ∧ (t[Apellido1_P] = "Pérez" ∨
t[Apellido2_P] = "Pérez")}

Consulta del Ejemplo 6.2 (proyección): realizar un listado con los nombres y apellidos de alumnos, junto con su número de teléfono.

{t / ∃s ∈ Alumnos ∧ t[Nombre_Al] = s[Nombre_Al] ∧ t[Apellido1_Al] = s[Apellido1_Al] ∧
t[Apellido2_Al] = s[Apellido2_Al] ∧ t[Teléfono_Al] = s[Teléfono _Al]}

Consulta del Ejemplo 6.3 (selección y proyección): devolver un listado con los nombres, apellidos y direcciones de los profesores que impartan *Historia*.

{t / ∃s ∈ Profesores ∧ t[Nombre_P] = s[Nombre_ P] ∧ t[Apellido1_P] = s[Apellido1_P] ∧
t[Apellido2_P] = s[Apellido2_P] ∧ t[Dirección_P] = s[Dirección_P] ∧ s[Especialidad_P] = "Historia"}

Consulta del Ejemplo 6.5 (unión): devolver un listado con los nombres y teléfonos de todas las personas de la base de datos, es decir, tanto *Alumnos* como *Profesores*.

{t / (∃s ∈ Alumnos ∧ t[Nombre_Al] = s[Nombre_Al] ∧ t[Apellido1_Al] = s[Apellido1_Al] ∧
t[Apellido2_Al] = s[Apellido2_Al] ∧ t[Teléfono_Al] = s[Teléfono _Al]) U (∃u ∈ Profesores ∧
t[Nombre_P] = u[Nombre_P] ∧ t[Apellido1_P] = u[Apellido1_P] ∧ t[Apellido2_P] = u[Apellido2_P] ∧
t[Teléfono_P] = u[Teléfono _P])}

Consulta del Ejemplo 6.6 (diferencia): mostrar un listado de todos los profesores de la especialidad *Historia* pero que no sean de *Almería*.

{t / (t ∈ Profesores ∧ t[Especialidad_P] = "Historia") ∧ ¬(t ∈ Profesores ∧ t[Ciudad_P] =
"Almería")}

Consulta del Ejemplo 6.7 (intersección): mostrar un listado de todos los profesores de la especialidad *Historia* pero que sean de *Almería*.

{t / (t ∈ Profesores ∧ t[Especialidad_P] = "Historia") ∧ (t ∈ Profesores ∧ t[Ciudad_P] =
"Almería")}

Consulta nueva (producto natural): consultar el nombre y apellidos de los profesores que imparten la asignatura de *bases de datos*.

{t / ∃s ∈ Profesores ∧ t[Nombre_P] = s[Nombre_ P] ∧ t[Apellido1_P] = s[Apellido1_P] ∧
t[Apellido2_P] = s[Apellido2_P] ∧ (∃u ∈ Asignaturas ∧ s[DNI_P] = u[DNI_P] ∧ u[Nombre_As] =
"bases de datos")}

EJERCICIO GUIADO 6.3

Si realizamos en cálculo relacional orientado a tuplas las consultas expuestas en el Ejercicio guiado 6.1, tenemos que:

1. **Nombres de clientes que han comprado *langosta*:**

   {t / ∃s ∈ Clientes ∧ t[Nombre_C] = s[Nombre_C] ∧ (∃u∈ Pedidos ∧
   s[NIF_C] = u[NIF_C] ∧ (∃v ∈ Consta_de ∧ u[Código_S] = v[Código_S] ∧
   u[Número_P] = v[Número_P] ∧ (∃x ∈ Artículos ∧ v[Código_A] = x[Código_A] ∧
   x[Descrip_A] = "langosta"))))}

2. **Nombre y domicilio de los empleados que no han entregado ningún pedido:**

   {t / ∃s ∈ Empleados ∧ t[Nombre_E] = s[Nombre_E] ∧ t[Domicilio_E] =
   s[Domicilio _E] ∧ ¬(∃u ∈ Pedidos ∧ s[NIF_E] = u[NIF_E])}

3. **Nombres y domicilios de empleados que, a su vez, son clientes del supermercado (han hecho algún pedido en alguna ocasión):**

   {t / (∃s ∈ Empleados ∧ t[Nombre_E] = s[Nombre_E] ∧ t[Domicilio_E] =
   s[Domicilio_E]) U (∃u ∈ Clientes ∧ t[Nombre_E] = u[Nombre_E] ∧
   t[Domicilio_E] = u[Domicilio_E] ∧ (∃v∈ Pedidos ∧
   u[NIF_C] = v[NIF_C]))}

4. **Fechas de pedidos que incluyen *tomates* y que han sido ordenados por *Amalia Gallegos*:**

   {t / ∃s ∈ Pedidos ∧ t[Fecha_P] = s[Fecha_P] ∧ (∃u ∈ Clientes ∧
   s[NIF_C] = u[NIF_C] ∧ u[Nombre_C] = "Amalia Gallegos") ∧ (∃v ∈ Consta_de ∧
   s[Código_S] = v[Código_S] ∧ s[Número_P] = v[Número_P] ∧ (∃x ∈ Artículos ∧
   v[Código_A] = x[Código_A] ∧ x[Descrip_A]= "tomates"))}

5. **Función con la que trabaja el empleado *Francisco Javier Martínez* en los supermercados en lo que ha trabajado que tengan una superficie superior a 1000 m^2:**

   {t / ∃s ∈ Trabajan ∧ t[Función_ES] = s[Función_ES] ∧ (∃u ∈ Empleados ∧
   s[NIF_E] = u[NIF_E] ∧ u[Nombre_E] = "Francisco Javier Martínez") ∧ (∃v ∈
   Supermercado ∧ s[Código_S] = v[Código_S] ∧ v[Superficie_S] > 1000)}

6.5 TRANSFORMACIÓN DE CONSULTAS ENTRE ÁLGEBRA Y CÁLCULO RELACIONAL

Tanto el álgebra relacional como el cálculo relacional tienen una equivalente capacidad de expresión a la hora de formular consultas, es decir, todas las consultas que se pueden formular utilizando álgebra relacional pueden también formularse utilizando el cálculo relacional, y viceversa.

Esta afirmación fue probada por Codd en 1972, a través de la invención de un algoritmo, el cual fue denominado **Algoritmo de Reducción de Codd,** mediante el cual una expresión arbitraria del cálculo relacional se puede reducir a la expresión semánticamente equivalente del álgebra relacional.

No obstante a lo anteriormente afirmado, en la literatura hay autores que afirmar que los lenguajes basados en el cálculo relacional son de "más alto nivel" o "más declarativos" que los basados en el álgebra relacional, ya que el álgebra especifica (parcialmente) el orden de las operaciones, mientras el cálculo lo traslada a un compilador o intérprete que determina el orden de evaluación más eficiente.

El algoritmo de reducción de Codd, que convierte cualquier expresión del cálculo relacional en una expresión del álgebra relacional, sigue los siguientes pasos:

1. Obtener el rango de cada tupla, aplicando las restricciones que sean posible (*rangos restringidos*).
2. Construir el producto cartesiano de los rangos obtenidos en el paso (1).
3. Restringir el producto cartesiano del paso (2) usando las restricciones del *join*.
4. Aplicar los cuantificadores de derecha a izquierda del siguiente modo:
   - ∃ Rx: proyectar el resultado actual para eliminar todos los atributos de la relación asociada a Rx.
   - ∀ Rx: dividir el resultado actual entre la relación que contiene el rango restringido asociado a Rx.
5. Proyectar el resultado del paso (4) de acuerdo con las especificaciones de la lista de objetivos.

6.6 LENGUAJES COMERCIALES: SQL (*STRUCTURED QUERY LANGUAGE*), QBE (*QUERY BY EXAMPLE*)

Los lenguajes anteriormente descritos se emplean como base para el desarrollo de los lenguajes comerciales, que dan soporte a los productos comerciales para la consulta y definición de datos en los DBMS relacionales.

Existen tres lenguajes principalmente, que se corresponden con los tres lenguajes formales estudiados:

- **SQL**, que está basado en álgebra relacional.
- **QUEL**, el cual se basa en el cálculo relacional de tuplas.
- Y, por último, **QBE**, basado en el cálculo relacional de dominios.

En este apartado desarrollaremos principalmente el lenguaje *Query by example* (QBE) y daremos un mínimo esbozo del *Structured Query Language* (SQL), ya que nos centraremos en este último lenguaje en los siguientes apartados y, especialmente, en el capítulo séptimo.

De esta forma, comenzaremos explicando que el **QBE** es un lenguaje de manipulación de datos creado por IBM y que se caracteriza por:

- Como ya hemos dicho, se basa en el cálculo relacional de dominios, por lo que es un lenguaje no procedimental.
- Su sintaxis es bidimensional, como veremos a continuación.
- Para realizar las consultas, no se indica exactamente lo que queremos obtener, sino un ejemplo de lo que deseamos conseguir.

  Para ello, se utilizan los denominados *esqueletos* (tablas vacías), de manera que, tras seleccionar los que serán utilizados, se introducirá un ejemplo en los campos de las filas de dichos esqueletos.

Veamos, a continuación, unos ejemplos relativos a su sintaxis, tanto para la realización de consultas como la modificación de relaciones.

Consultas simples

EJEMPLO 6.12

En este ejemplo vamos a desarrollar algunas consultas realizadas en QBE, explicando su significado:

1. **Mostrar una relación en su totalidad:** "P" significa *mostrar/imprimir*.

Nombre_tabla	Campo1	Campo2	Campo3
	P.valor1	P.valor2	P.valor3

2. **Mostrar determinados campos de una relación:** QBE elimina duplicados.

Nombre_tabla	Campo1	Campo2	Campo3
		P.valor2	

3. **Mostrar determinados campos estableciendo condiciones:** ***Campo2>16***

Nombre_tabla	Campo1	Campo2	Campo3
	P.valor1	>16	

4. **Mostrar los valores de *Campo1* si *Campo2>16* y *Campo3* es *Cádiz:***

Nombre_tabla	Campo1	Campo2	Campo3
	P.valor1	>16	Cádiz

5. **Mostrar los valores de *Campo1* si *Campo2≥16:***

Nombre_tabla	Campo1	Campo2	Campo3
	P.valor1	>16	
	P.valor2	=16	

Consultas con varias relaciones

EJEMPLO 6.13

Mostrar los valores de *Campo1* de *Nombre_tabla1* cuando los valores del *Campo3* de esa tabla sean iguales al *Campo1* de la tabla *Nombre_tabla2* y cuando el *Campo3* de la segunda tabla sea igual a *Cádiz*.

Nombre_tabla1	Campo1	Campo2	Campo3
	P.valor1		Valor

Nombre_tabla2	Campo1	Campo2	Campo3
	Valor		Cádiz

Ordenar las filas

Se puede realizar de manera ascendente (P.A.O.) o descendente (P.D.O.).

EJEMPLO 6.14

Mostrar los valores de *Campo1* ordenados de forma ascendente mediante el *Campo3*.

Nombre_tabla	Campo1	Campo2	Campo3
	P.valor1		P.A.O.

Funciones de agregación

Como hemos comentado anteriormente, QBE eliminar, por defecto, los duplicados. Por ello, si no queremos que se eliminen lo tenemos que especificar mediante la palabra reservada ***ALL***.

Las funciones de agregación que maneja son las siguientes:

- MAX: devuelve el valor máximo.
- MIN: devuelve el valor mínimo.
- AVG: devuelve el promedio de las tuplas seleccionadas.
- SUM: devuelve la suma de las tuplas seleccionadas.
- CNT: devuelve el número total de las tuplas que cumplan una condición.

EJEMPLO 6.15

Mostrar los valores de *Campo1* junto a la media de todos sus valores del *Campo3*.

Nombre_tabla	Campo1	Campo2	Campo3
	P.valor1		P.AVG.ALL.valor

NOTA

Para la consulta del Ejemplo 6.15 se podría utilizar la capacidad que posee QBE de realizar agrupamientos de tuplas. De este modo, empleando el operador *G*, se llevaría a cabo la misma función que con el *Group By* de SQL. Por tanto, la consulta podría plantearse de la siguiente forma:

Nombre_tabla	Campo1	Campo2	Campo3
	P.G		P.AVG.ALL.valor

Supresión de tuplas

Podremos también suprimir tuplas completas, o solo determinadas columnas. En este último caso, los valores serán sustituidos por valores nulos.

Del mismo modo, para la selección de varias tuplas se utilizará una consulta. Por su parte, para suprimir tuplas en varias relaciones, se tendrá que realizar el proceso de forma independiente para cada relación.

El operador empleado para la eliminación es *D*.

EJEMPLO 6.16

Eliminar las tuplas cuya dado un valor del *Campo3*.

Nombre_tabla	Campo1	Campo2	Campo3
D			Valor

Inserción de tuplas

También podemos insertar una o un conjunto de tuplas. Para la selección de varias tuplas, se utilizará una consulta.

El operador empleado es *I*.

EJEMPLO 6.17

Ejemplo de inserción.

Nombre_tabla	Campo1	Campo2	Campo3
I	valor1		valor3

Modificación o actualización

Por último, QBE también permite sustituir un valor por otro nuevo o realizar cambios basándonos en el valor existente. Como comentábamos en la inserción, para la selección de varias tuplas, se utiliza una consulta.

El operador utilizado es *U.*

EJEMPLO 6.18

Actualización del *Campo3* si el *Campo1* cumple un criterio.

Nombre_tabla	Campo1	Campo2	Campo3
U	Criterio		nuevoValor

6.7 ORÍGENES Y EVOLUCIÓN DEL SQL

En el origen del actual SQL se le denominó *Structured English Query Language* (SEQUEL). Fue creado en 1974, siendo desarrollado por Donald D. Chamberlin y otros integrantes de IBM Research entre los que destacan Raymond F. Boyce; estos se basaron en el modelo relacional propuesto por Edgar Frank Codd. Dicho lenguaje fue implementado en un prototipo de IBM denominado SEQUEL-XRM**.**

INFORMACIÓN

Si tienes un nivel de inglés mínimo, os recomendamos echéis un vistazo al artículo original donde se presentaba el nuevo lenguaje. Este artículo se titula "SEQUEL: a Structured English Query Language" y es obra de los dos autores mencionados anteriormente:

http://www.almaden.ibm.com/cs/people/chamberlin/sequel-1974.pdf

Dicho artículo está publicado en la propia página web de **IBM Research**.

IBM Research

En dicho artículo, podemos ver, entre otras aspectos relevantes, las continuas referencias a los trabajos de Codd.

Entre los años 1976 y 1977 se definió una revisión de SEQUEL, la cual se denominó SEQUEL/2, aunque más tarde pasaría a llamarse finalmente SQL, siendo la actual denominación del lenguaje. Ese mismo año de 1977 se desarrolló un nuevo prototipo, denominado System R, el cual produjo un salto cuantitativo y cualitativo al proyecto SQL.

De este modo, para su testeo, System R fue instalado en un número de puestos de usuario, tanto internos de IBM como de una selección de clientes, y, gracias el rotundo éxito obtenido, IBM inició el desarrollo de productos comerciales que implementaban el lenguaje SQL basado en la tecnología System R.

NOTA

Tal fue la aceptación de SQL y System R que, durante los años siguientes, otras empresas desarrollaron productos basados en este lenguajes, tales como:

Producto	Empresa
SQL/DS	IBM
DB2	IBM
ORACLE	Oracle Corp.
DG/SQL	Date General Corp.
SYBASE	SyBase Inc.

En 1982, la American National Standards Institute (ANSI) propuso a su Comité de Bases de Datos X3H2 el desarrollo de un estándar para lenguajes relacionales. Este encargo se materializó, en 1986, con la propuesta del lenguaje SQL de IBM. Dicho estándar se denominaría SQL-86 O SQL1. SQL-86 también sería aceptado por la Organización Internacional de Estandarización (ISO), en 1987.

Sin embargo, esta versión (y sus versiones posteriores, como la de 1989) no cumplían con todos los requerimientos que necesitaban los desarrolladores, por lo que, en 1992, se lanza un nuevo estándar ampliado y revisado del SQL, esta vez denominado SQL-92 o SQL2. Esta versión se convirtió en una estándar ratificado por la International Standard ISO/IEC 9075:1992 y es la versión a la que normalmente nos referimos cuando hablamos de **SQL estándar.**

Durante los años, SQL ha sufrido diversas revisiones, algunas menores, otras más significativas. De todas ellas podemos destacar:

Año	Nombre	Comentario
1974-75	SEQUEL-XRM	Antecesor del actual SQL para IBM
1976-77	SEQUEL/2	Revisión de SEQUEL y cambio de nombre a SQL por motivos legales
1986	SQL-86 (SQL1)	Primera versión estandarizada por ANSI y, en 1987, por ISO
1989	SQL-89	Revisión menor
1992	SQL-92 (SQL2)	Revisión mayor y cambio en la estandarización. Es el llamado ANSI SQL o SQL estándar
1999	SQL:1999 (SQL2000 o SQL3)	Se agregaron expresiones regulares, consultas recursivas (para relaciones jerárquicas), *triggers* y algunas características orientadas a objetos
2003	SQL:2003	Introduce algunas características de XML, cambios en las funciones, estandarización del objeto *sequence* y de las columnas autonuméricas
2006	SQL:2006	ISO/IED 9075:2006 define las maneras en las que SQL se puede utilizar con XML. También define maneras de importar y guardar datos XML en una base de datos SQL, manipulándolos en la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, permite integrar el uso de XQuery, lenguaje de consultas SML publicado por el W3C para acceso concurrente a datos ordinarios SQL y documentos XML
2008	SQL:2008	Permite el uso de la cláusula *ORDER BY* fuera de las definiciones de los cursores. Incluye también los disparadores de tipo *INSTEAD OF* y la sentencia *TRUNCATE*
2011	SQL:2011	Añade características de manejo de datos temporales, así como tablas versionadas en el sistema, etc.
2012	SQL:2012	Última versión a fecha de la edición de esta publicación

6.8 CARACTERÍSTICAS DEL SQL

En la actualidad, SQL es el estándar de la mayoría de los DBMS comerciales, tales como:

- MySQL
- PostgreeSQL
- DB2
- Informix
- Microsoft SQL
- Access
- Oracle
- SQL Server
- ASE
- Firebird
- SyBase

Aunque las diferentes empresas proponen para sus productos una variedad de añadidos particulares, el estándar sigue basándose en el SQL-92.

Como hemos comentado a lo largo de este libro, SQL es un lenguaje declarativo de alto nivel y no procedimental, orientado al trabajo con conjuntos de registros (no a registros individuales), que permite una alta productividad en codificación y la orientación a objetos.

Entre las principales características de SQL, podemos destacar las siguientes:

- El lenguaje de definición de datos proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación.
- El lenguaje de manipulación de datos incluye la posibilidad de realizar consultas basadas en álgebra relacional y en cálculo relacional de tuplas.
- El lenguaje de definición de datos incluye comandos para especificar restricciones de integridad sobre los datos de la base de datos.
- El lenguaje de definición de datos permite definir vistas.
- SQL implementa comando para especificar el comienzo y el fin de una transacción.
- SQL incorpora instrucciones para lenguajes de programación como C, C++, Java, PHP, entre otros.
- El lenguaje de definición de datos incorpora comandos para especificar los derechos de acceso a las relaciones y a las vistas.

Existen realmente pocas desventajas en relación al uso de SQL. Entre estas, podemos destacar:

- La implementación es inconsistente entre las empresas que lo comercializan (como antes hablábamos, aunque se basa en un mismo estándar). Esto es especialmente significativo en la sintaxis para trabajar con fechas, caracteres nulos, concatenación y caracteres especiales.

  De hecho, las empresas comercializadoras tienden a aumentar la incompatibilidad para fidelizar a sus clientes, es decir, para aumentar el coste de aprendizaje en el cambio de producto.
- La facilidad con la que se puede inducir al error, por ejemplo, con una mala sintaxis de un comando.
- La gramática SQL es ciertamente compleja y poco intuitiva.

6.9 SISTEMAS DE GESTIÓN DE BASES DE DATOS CON SOPORTE SQL

RECUERDA

En la sección 4.1.2.3 "Selección del equipo físico y lógicos necesarios" pudimos estudiar los principales tipos de DBMS con soporte SQL, junto a sus características más destacadas.

6.10 EJERCICIOS RESUELTOS

NOTA

Este extracto de ejercicio pertenece a un ejercicio de las **Oposiciones a cuerpos docentes en Andalucía (Convocatoria de 2004), concretamente para la especialidad de *Sistemas y Aplicaciones Informáticas.***

1. Partiendo del modelo Entidad-Relación del Ejercicio resuelto número 1 del capítulo 2, se pide realizar las siguientes consultas en SQL y, las que sean posible, en álgebra y cálculo relacional:

- **Cursos coordinados, en el año 2003, por "Jesús López Gil".**

```
SELECT Nombre_C
FROM Cursos, Profesores
WHERE Cursos.NIF_Coordinador = Profesores.NIF_Pr AND Nombre_Pr = "Jesús López Gil";
```

$\pi_{Nombre\_C}(\sigma_{Nombre\_Pr= \text{“Jesús López Gil”}}((Cursos * Profesores)_{Cursos.NIF\_Coordinador=Profesores.NIF\_Pr}))$

$\{<p_2> / \exists\, p_1, p_3, p_4\, (<p_1, p_2, p_3, p_4> \in Cursos \wedge \exists\, q_2, q_3 (<p_4, q_2, q_3> \in Profesores \wedge q_2 = \text{“Jesús López Gil”}))\}$

$\{t / \exists s \in Cursos \wedge t[Nombre\_C] = s[Nombre\_C] \wedge (\exists u \in Profesores \wedge s[NIF\_Coordinador] = u[NIF\_Pr] \wedge u[Nombre\_Pr] = \text{“Jesús López Gil”})\}$

- **Cursos del área de lengua, que hayan tenido un número de alumnos entre 15 y 20, con más de un ponente.**

```
SELECT Nombre_C
FROM Cursos C, Adscrito D, Áreas A, Asisten S, Alumnos L, Ponentes P, Participan R
WHERE C.Id_C = D.Id_C AND D.Código_A = A.Código_A AND C.Id_C = S.Id_C
AND S.NIF_Al = L.NIF_Al AND C.Id_C = R.Id_C AND R.NIF_Po = P.NIF_Po
AND Nombre_A = "Lengua"
GROUP BY Nombre_C
HAVING (COUNT (DISTINCT S.NIF_Al) BETWEEN 15 AND 20)
AND (COUNT (DISTINCT NIF_Po) > 1);
```

2. Partiendo del modelo Entidad-Relación del Ejercicio resuelto número 2 del capítulo 2, se pide realizar las siguientes consultas en SQL y, las que sean posible, en álgebra y cálculo relacional:

- **¿Cuáles son los nombres de los clientes que han realizado una reserva en el mes de enero de 2005 en régimen de pensión completa?**

  SELECT Nombre_Cli

  FROM Reservas R, Clientes C

  WHERE R.DNI_Cli = C.DNI_Cli AND Entrada_R BETWEEN '01/01/2005' AND '31/01/2005'

  AND Tipo_R = "Pensión completa";

  $\pi_{Nombre\_Cli}(\sigma_{Entrada\_R >= '01/01/2005' \wedge Entrada\_R <= '31/01/2005' \wedge Tipo\_R = "Pensión\ completa"}$ ((Reservas * Clientes)$_{Reservas.DNI\_Cli = Clientes.DNI\_Cli}$))

  $\{<p_2> / \exists\ p_1\ (<p_1, p_2> \in Clientes \wedge \exists\ q_1, q_2, q_3, q_4, q_5, q_6\ (<q_1, q_2, q_3, q_4, p_1, q_5, q_6> \in Reservas \wedge q_3 >= '01/01/2005' \wedge q_3 <= '31/01/2005' \wedge q_6 = "Pensión\ completa"))\}$

  $\{t / \exists s \in Clientes \wedge t[Nombre\_Cli] = s[Nombre\_Cli] \wedge (\exists u \in Reservas \wedge s[DNI\_Cli] = u[DNI\_Cli] \wedge u[Entrada\_R] >= '01/01/2005' \wedge u[Entrada\_R] <= '31/01/2005' \wedge u[Tipo\_R] = "Pensión\ completa")\}$

- **¿Cuántas reservas se han hecho para cada régimen de alojamiento?**

  SELECT Tipo_R, COUNT (Código_R)

  FROM Reserva

  GROUP BY Tipo_R;

- **Nombres de los clientes que han realizado alguna reserva en enero de 2005 de alojamientos equipados con calefacción central.**

  SELECT Nombre_Cli

  FROM Reservas R, Clientes C, Ofrece O, Equipamiento E

  WHERE R.DNI_Cli = C.DNI_Cli AND R.Código_A = O.Código_A AND O.Código_E = E.Código_E

  AND Entrada_R BETWEEN '01/01/2005' AND '31/01/2005'

  AND Descripción_E = "Calefacción central";

  $\pi_{Fecha\_P}((((((\sigma_{Entrada\_R >= '01/01/2005' \wedge Entrada\_R <= '31/01/2005'}$ (Reservas)) * Clientes)$_{Clientes.DNI\_Cli=Reservas.CNI\_Cli}$) * Ofrece)$_{Reservas.Código\_A=Ofrece.Código\_A}$) * ($\sigma_{Descrip\_E= "Calefacción\ central"}$ (Equipamiento)))$_{Ofrece.Código\_E=Equipamiento.Código\_E}$))

  $\{<p_2> / \exists\ p_1\ (<p_1, p_2> \in Clientes \wedge \exists\ q_1, q_2, q_3, q_4, q_5, q_6\ (<q_1, q_2, q_3, q_4, p_1, q_5, q_6> \in Reservas \wedge q_3 >= '01/01/2005' \wedge q_3 <= '31/01/2005') \wedge \exists\ r_1, r_2 (<q_5, r_1, r_2> \in Ofrece) \wedge \exists\ s_1\ (<r_1, s_1> \in Equipamiento \wedge s_1 = "Calefacción\ central"))\}$

  $\{t / \exists s \in Clientes \wedge t[Nombre\_Cli] = s[Nombre\_Cli] \wedge (\exists u \in Reservas \wedge s[DNI\_Cli] = u[DNI\_Cli] \wedge u[Entrada\_R] >= '01/01/2005' \wedge u[Entrada\_R] <= '31/01/2005') \wedge (\exists v \in Ofrece \wedge u[Código\_A] = v[Código\_A]) \wedge (\exists x \in Equipamiento \wedge v[Código\_E] = x[Código\_E] \wedge x[Desc\_E] = "Calefacción\ central")\}$

3. Partiendo del modelo Entidad-Relación del Ejercicio resuelto número 3 del capítulo 2, se pide realizar las siguientes consultas en SQL y, las que sean posible, en álgebra y cálculo relacional:

- **¿Cuáles son los títulos de las revistas publicadas a partir del 1 de febrero de 1990?**

```
(SELECT Título_R
FROM Revistas)
MINUS
(SELECT Título_R
FROM Revistas R, Números N
WHERE R.NRegistro_R = N.NRegistro_R AND Fecha_N <= '01/02/1990');
```

$(\pi_{\text{Título\_R}}(\text{Revistas})) -$

$(\pi_{\text{Título\_R}}(\sigma_{\text{Fecha\_N <= '01/02/1990}}((\text{Revistas} * \text{Números})_{\text{Revistas.NRegistro\_R=Números.NRegistro\_R}})))$

$\{<p_1, p_2, p_3, p_4, p_5> / ((<p_1, p_2, p_3, p_4, p_5> \in \text{Revistas} \wedge \neg(<p_1, p_2, p_3, p_4, p_5> / \exists\, p_1\, (<p_1, p_2, p_3, p_4, p_5> \in \text{Revistas}) \wedge \exists\, q_1, q_2, q_3\, (<p_1, q_1, q_2, q_3> \in \text{Números} \wedge q_1 <= \text{'01/02/1990'}))))\}$

$\{t / t \in \text{Revistas} \wedge \neg(\, t \in \text{Revistas} \wedge (\exists u \in \text{Números} \wedge t[\text{NRegistro\_R}] = u[\text{NRegistro\_R}] \wedge$

$u[\text{Fecha\_N}] <= \text{'01/02/1990'}))\}$

- **¿Cuáles son las secciones fijas de la revista "Teleindispuesta"?**

```
SELECT Título_Sec
FROM Revistas R, Secciones S
WHERE R.NRegistro_R = S.NRegistro_R AND Título_R = "Teleindispuesta";
```

$\pi_{\text{Título\_Sec}}(\sigma_{\text{Título\_R = "Teleindispuesta"}}((\text{Revistas} * \text{Secciones})_{\text{Revistas.NRegistro\_R=Secciones.NRegistro\_R}}))$

$\{<p_2> / \exists\, p_1, p_3 (<p_1, p_2, p_3> \in \text{Secciones}) \wedge \exists\, q_1, q_2, q_3, q_4 (<p_1, q_1, q_2, q_3, q_4> \in \text{Revistas} \wedge q_1 = \text{"Teleindispuesta"}))\}$

$\{t / \exists s \in \text{Secciones} \wedge t[\text{Nombre\_Sec}] = s[\text{Nombre\_Sec}] \wedge (\exists u \in \text{Revistas} \wedge$

$t[\text{NRegistro\_R}] = u[\text{NRegistro\_R}] \wedge u[\text{Título\_R}] = \text{"Teleindispuesta"})\}$

- **¿Cuáles son los nombres de los periodistas que escriben en revistas del corazón?**

```
SELECT Nombre_P, Apellidos_P
FROM Periodistas P, Escriben E, Revistas R
WHERE P.DNI_P = E.DNI_P AND E.NRegistro_R = R.NRegistro_R AND Tipo_R = "Corazón";
```

$\pi_{\text{Nombre\_P, Apellidos\_P}}((((((\sigma_{\text{Tipo\_R = "Corazón"}}(\text{Revistas})) *$

$\text{Escriben})_{\text{Revistas.NRegistro\_R=Escriben.NRegistro\_R}}) * \text{Periodistas})_{\text{Escriben.DNI\_P=Periodistas.DNI\_P}}))$

$\{<p_2, p_3> / \exists\, p_1, p_4 (<p_1, p_2, p_3, p_4> \in \text{Periodistas}) \wedge \exists\, q_1 (<q_1, p_1> \in \text{Escriben}) \wedge \exists\, s_1, s_2, s_3, s_4 (<q_1, s_1, s_2, s_3, s_4> \in \text{Revistas} \wedge s_2 = \text{"Corazón"}))\}$

$\{t / \exists s \in \text{Periodistas} \wedge t[\text{Nombre\_P}] = s[\text{Nombre\_P}] \wedge t[\text{Apellidos\_P}] = s[\text{Apellidos\_P}] \wedge$

$(\exists u \in \text{Escriben} \wedge t[\text{DNI\_P}] = u[\text{DNI\_P}]) \wedge$

$(\exists v \in \text{Revistas} \wedge u[\text{NRegistro\_R}] = v[\text{NRegistro\_R}] \wedge v[\text{Tipo\_R}] = \text{"Corazón"})\}$

6.11 EJERCICIOS PROPUESTOS

1. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 3 del capítulo 2, se propone realizar las siguientes consultas en SQL y, las que sean posible, en álgebra y cálculo relacional:

- ¿Qué clientes tienen facturas pendientes del mes de enero de 2004?
- ¿Cuáles son los nombres y teléfonos de los responsables de pueblos en los que se paga impuesto ecológico?
- ¿Cuáles son las fechas de aviso de pago de basura y códigos de propiedad correspondiente de Mario López?
- ¿Cuánto se ha recaudado en cada pueblo cada año?

2. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 4 del capítulo 2, se propone realizar las siguientes consultas en SQL y, las que sean posible, en álgebra y cálculo relacional:

- ¿Nombres de pilotos con más de 10 años de antigüedad que realicen vuelos intercontinentales?
- ¿Cuál es el nombre del pasajero que tiene una reserva realizada para el 8/9/2003 en el vuelo Almería-Dresde en el asiento 12A?
- ¿Cuáles son los nombres de los componentes de la tripulación (piloto, copiloto y personal auxiliar) del vuelo IB0987?
- Número de pasajeros de cada clase del vuelo IB0987.

3. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 5 del capítulo 2, se propone realizar las siguientes consultas en SQL y, las que sean posible, en álgebra y cálculo relacional:

- ¿Está libre la pista 1 el 20/12/2012 de 20:00 a 21:30?
- Nombre y apellidos de los socios apuntados al curso denominado “Adultos Grupo 1” que tienen pendiente de pago el recibo del mes de noviembre de 2005.
- ¿Cuántos alumnos tiene el profesor XX YY en cada curso?
- Socios que hayan realizado más de 20 reservas en el mes de noviembre de 2010 en pistas homologadas para campeonatos.

4. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 1 del capítulo 5, se propone realizar las siguientes consultas en SQL y, las que sean posible, en álgebra y cálculo relacional:

- Nombres de clientes que hayan comprado cocinas que incluyan como extra un botellero.
- ¿Cuántas cocinas corresponden a una composición que incluya botellero?
- Clientes que hayan comprado más de una cocina.
- Composiciones que tienen un precio mayor que la media.

5. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 2 del capítulo 5, se propone realizar las siguientes consultas en SQL y, las que sean posible, en álgebra y cálculo relacional:

- ¿Cuáles son los sitios web de los alpargateros que han expuesto en ZAPA 2005?
- Nombre y domicilio de los visitantes que han pasado por el stand de “Cocinas C” en cualquier feria.
- ¿Cuántos visitantes ha tenido cada edición de la feria MOVI?
- Nombres de las empresas que han gastado más de 6.000 € en stands de ferias.

6. Partiendo del modelo Entidad-Relación del Ejercicio propuesto número 3 del capítulo 5, se propone realizar las siguientes consultas en SQL y, las que sean posible, en álgebra y cálculo relacional:

- ¿Qué atracciones tuvieron como incidencia "sustitución de bombilla" el 20 de agosto de 2012?
- ¿Cuántas personas subieron el 20 de agosto de 2008 a atracciones de clasificación adultos y peligrosidad alta?
- ¿Cuáles son los nombres y apellidos de los empleados que estaban trabajando en la atracción "Montaña rusa" el 20 de agosto de 2008 a las 20:00 horas?
- Número de atracciones clasificadas por tramo de edad y peligrosidad que hayan sufrido incidencias en el mes de agosto de 2006.

6.12 TEST DE CONOCIMIENTOS

NOTA

Esta consulta pertenece a un ejercicio de las **Oposiciones al Cuerpo de Gestión de Sistemas e Informática de la Administración del Estado (Convocatoria de 2015)**.

1 ¿Cuál de las siguientes operaciones del álgebra relacional es una operación unaria?
- **a)** Proyección
- **b)** Diferencia de conjuntos
- **c)** Producto cartesiano
- **d)** Unión

2 ¿Qué operación del álgebra relacional selecciona las tuplas de una relación que satisfaga una determinada condición?
- **a)** Proyección
- **b)** Selección
- **c)** Producto cartesiano
- **d)** Unión

3 Los operadores de la teoría de conjuntos son:
- **a)** Selección, proyección y diferencia
- **b)** Intersección, unión y diferencia
- **c)** Unión, producto cartesiano, diferencia e intersección
- **d)** Selección, proyección, diferencia, unión, producto cartesiano e intersección

4 Señala la respuesta correcta:
- **a)** Los operadores relacionales sirven para establecer condiciones y son "<, >, <=, >=, =, <>".

b) Los conectores lógicos sirven para concatenar varias condiciones y son "∧, ∨, ¬".
c) (a) y (b) son correctas.
d) Ninguna de las anteriores.

5 La expresión *"(TABLA $_1$) – (σ_{Cond1} (TABLA$_2$))"*:
a) Devuelve las tuplas de *TABLA$_1$* que cumplen la condición *Cond1*.
b) Devuelve las tuplas de *TABLA$_2$* que cumplen la condición *Cond1*.
c) Devuelve las tuplas de *TABLA$_1$* menos las que coincidan con las tuplas de *TABLA$_2$*.
d) Ninguna de las anteriores.

6 La expresión *"(TABLA $_1$ U TABLA$_2$)"*:
a) Devuelve las tuplas de *TABLA$_1$* y *TABLA$_2$*.
b) Devuelve las tuplas de *TABLA$_1$* que coinciden con las tuplas de *TABLA$_2$*.
c) Devuelve las tuplas de *TABLA$_1$* salvo las que coinciden con las tuplas de *TABLA$_2$*.
d) Ninguna de las anteriores.

7 La expresión *"(TABLA $_1$ ∩ TABLA$_2$)"*:
a) Devuelve las tuplas de *TABLA$_1$* y *TABLA$_2$*.
b) Devuelve las tuplas de *TABLA$_1$* que coinciden con las tuplas de *TABLA$_2$*.
c) Devuelve las tuplas de *TABLA$_1$* salvo las que coinciden con las tuplas de *TABLA$_2$*.
d) Ninguna de las anteriores.

8 La expresión *"(TABLA $_1$ - TABLA$_2$)"*:
a) Devuelve las tuplas de *TABLA$_1$* y *TABLA$_2$*.
b) Devuelve las tuplas de *TABLA$_1$* que coinciden con las tuplas de *TABLA$_2$*.
c) Devuelve las tuplas de *TABLA$_1$* salvo las que coinciden con las tuplas de *TABLA$_2$*.
d) Ninguna de las anteriores.

9 La expresión en álgebra relacional siguiente *"$\pi_{Nombre\_P}(\sigma_{Edad\_P = 18}$(Personas)"* equivale a la siguiente expresión en cálculo relacional de dominios:
a) {<q> / ∃ p, r, s, t(<p, q, r, s, t > ∈ Personas ∧ s = 18)}
b) {t / ∃s ∈ Personas ∧ t[Edad_P] = 18}
c) SELECT Nombre_P FROM Personas WHERE Edad_P=18;
d) Ninguna de las anteriores.

10 La expresión en álgebra relacional siguiente *"$\pi_{Nombre\_P}(\sigma_{Edad\_P = 18}$(Personas)"* equivale a la siguiente expresión en cálculo relacional de tuplas:
a) {<q> / ∃ p, r, s, t(<p, q, r, s, t > ∈ Personas ∧ s = 18)}
b) {t / ∃s ∈ Personas ∧ t[Edad_P] = 18}
c) SELECT Nombre_P FROM Personas WHERE Edad_P=18;
d) Ninguna de las anteriores.

11 La expresión en álgebra relacional siguiente "$\pi_{Nombre\_P}(\sigma_{Edad\_P = 18}(Personas)$" equivale a la siguiente expresión en SQL:

a) {<q> / ∃ p, r, s, t(<p, q, r, s, t > ∈ Personas ∧ s = 18)}
b) {t / ∃s ∈ Personas ∧ t[Edad_P] = 18}
c) SELECT Nombre_P FROM Personas WHERE Edad_P=18;
d) Ninguna de las anteriores.

12 Indica la afirmación correcta:

a) SQL está basado en cálculo relacional de tuplas, QUEL es cálculo relacional de dominios y QBE en álgebra relacional.
b) SQL está basado en álgebra relacional, QUEL es cálculo relacional de dominios y QBE en cálculo relacional de tuplas.
c) SQL está basado en cálculo relacional de tuplas, QUEL es álgebra relacional y QBE en cálculo relacional de dominios.
d) SQL está basado en álgebra relacional, QUEL es cálculo relacional de tuplas y QBE en cálculo relacional de dominios.

13 La revisión de SQL que introduce la posibilidad de utilizar XML es:

a) SQL:2006
b) SQL:1999
c) SQL:2008
d) SQL:2011

14 ¿Cuál de estas no es una característica de SQL?

a) SQL no emplea comandos para especificar el comienzo y el fin de una transacción.
b) El lenguaje de definición de datos proporciona comandos para la definición de esquemas de relación, borrado de relaciones y modificaciones de los esquemas de relación.
c) El lenguaje de manipulación de datos incluye la posibilidad de realizar consultas basadas en álgebra relacional y en cálculo relacional de tuplas.
d) El lenguaje de definición de datos incluye comandos para especificar restricciones de integridad sobre los datos de la base de datos.

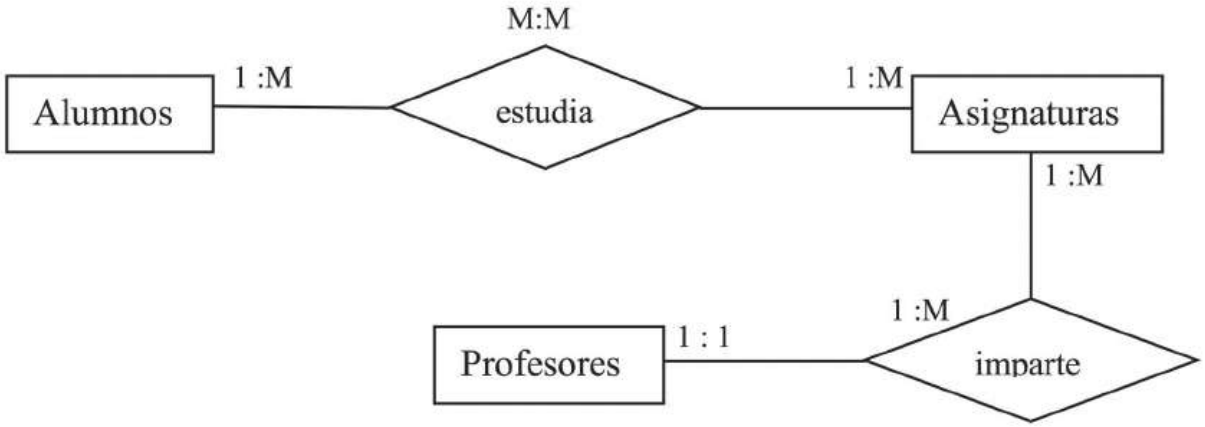
15 ¿Cuál de estas no es una característica de SQL?

a) El lenguaje de definición de datos incorpora comandos para especificar los derechos de acceso a las relaciones, pero no a las vistas.
b) El lenguaje de definición de datos incluye comandos para especificar restricciones de integridad sobre los datos de la base de datos.
c) El lenguaje de definición de datos permite definir vistas.
d) SQL incorpora instrucciones para lenguajes de programación como C, C++, Java, PHP, entre otros.

7 El lenguaje de manipulación de la base de datos

RECUERDA

Al igual que hiciéramos en el capítulo 6, para todos los ejemplos de consultas que se van a ir mostrando a lo largo de este capítulo emplearemos la base de datos con la que hemos estado trabajando, con la posible incorporación de algunos atributos adicionales:



- **Alumnos** (DNI_Al, Nombre_Al, Apellido1_Al, Apellido2_Al,Edad_Al, Teléfono_Al, Dirección_Al, Ciudad_Al)
- **Estudian** (DNI_Al, Cód_As, Nota_Al_As, Convocatoria_Al_As)
- **Asignaturas** (Cód_As, Nombre_As, Créditos_As, Facultad_As, DNI_P)
- **Profesores** (DNI_P, Nombre_P, Apellido1_P, Apellido2_P, NCuenta_P, Teléfono_P, Especialidad_P, Dirección_P, Ciudad_P)

7.1 EL LENGUAJE DE DEFINICIÓN DE DATOS (DDL)

Ya hemos comentado en diferentes apartados de este libro que un lenguaje de definición de datos (DDL) posibilita la definición de los elementos que conforman una base de datos (tablas, vistas, esquemas, índices), a través de diferentes tipos de sentencias, como son:

- Sentencia **create**, que se utiliza para crear los elementos anteriormente citados.
- Sentencia **alter**, que permite la modificación de las definiciones anteriores.
- Sentencia **drop**, la cual elimina cualquier tipo de elemento de los anteriormente comentados.

En este apartado, tendremos la oportunidad de profundizar en estas sentencias.

7.1.1 TIPOS DE DATOS DEL LENGUAJE

Los principales tipos de datos del estándar ANSI SQL son los siguientes:

Tipo	Descripción
CHARACTER (longitud)	Cadena de caracteres de longitud fija
CHARACTER VARYING (longitud)	Cadena de caracteres de longitud variable
BIT (longitud)	Cadena de bits de longitud fija
BIT VARYING (longitud)	Cadena de bits de longitud variable
NUMERIC (precisión, escala)	Número decimal con tantos dígitos como indique la precisión y tantos decimales como indique la escala.
DECIMAL (precisión, escala)	Número decimal con tantos dígitos como indique la *precisión* y tantos decimales como indique la escala
INTEGER	Número entero
SMALLINT	Número entero pequeño
REAL	Número con coma flotante con precisión predefinida
FLOAT (precisión)	Número con coma flotante con la *precisión* especificable
DOUBLE PRECISION	Número con coma flotante con más precisión predefinida que la del tipo *REAL*
DATE	Fecha, compuesta por YEAR (año), MONTH (mes) y DAY (día)
TIME	Hora, compuesta por HOUR (hora), MINUT (minuto) y SECOND (segundo)
TIMESTAMP	Fecha y hora, compuesta por los elementos de los tipos anteriores

NOTA

Los tipos de datos es uno de los aspectos que más difiere del lenguaje SQL de los distintos proveedores de DBMS.

Un ejemplo de ello lo establece MySQL, que cuenta con un abanico mucho más amplio de tipos de datos, como podemos comprobar en los enlaces siguientes:

• En primer lugar vemos los principales tipos de datos, utilizando, como fuente, el manual de referencia del producto:

http://dev.mysql.com/doc/refman/5.7/en/data-types.html

• Y, en segundo lugar, vemos esta misma información, en castellano, de mano de la famosa web *desarrolloweb.com:*

http://www.desarrolloweb.com/articulos/1054.php

Además, SQL cuenta con los siguientes modificadores de tipo:

- NULL: se admiten valores nulos para este atributo.
- NOT NULL: no se admiten valores nulos para este atributo.
- UNIQUE: no se admiten valores repetidos.

Adicionalmente a los tipos de datos predefinidos, el SQL92 nos ofrece la posibilidad de trabajar con dominios definidos por el usuario.

De este modo, para crear un dominio es necesario utilizar la sentencia *CREATE DOMAIN*, siguiendo la siguiente *sintaxis*:

CRATE DOMAIN <nombre_dominio> [[9]AS] tipos_datos [def_defecto] [restricciones_dominio];

Donde *restricciones_dominio* tiene, a su vez, la siguiente *sintaxis*:

[CONSTRAINT nombre_restricción] CHECK (condiciones)

NOTA

Veremos el uso de dominios con mayor profundidad en la sección 7.11.2 "Especificación de restricciones de integridad".

7.1.2 CREACIÓN, MODIFICACIÓN Y BORRADO DE TABLAS

Creación de tablas

Para crear una tabla, debemos indicar el nombre de la misma, definir sus atributos, o campos (nombre, tipo), y las restricciones a aplicar tanto a la tabla como a los campos que la forman.

Sintaxis:

CREATE TABLE <nombre_tabla> ([[10]<restricción_tabla>] <descripción_campo>,([<restricción_tabla>] <descripción_campo>)*);

Donde <*descripción_campo*> se compone de:

<nombre_campo><tipo_dato> [<modificadores_tipo>]

Definición de claves

Al definir la tabla, debemos indicar cuál de los campos constituyen la clave principal y cuáles son claves secundarias.

Sintaxis para la definición de claves:

PRIMARY KEY <nombre_campo/s>

9 En la sintaxis, los operadores [<algo>] significan que <algo> es un elemento opcional.

10 En la sintaxis, los operadores [<algo>] significan que <algo> es un elemento opcional.

Del mismo modo, también podemos indicar cuándo la clave de otra tabla se convierte en un campo de otra (por ejemplo para representar relaciones).

<descripción_campo> REFERENCES <nombre_tabla> (<nombre_campo/s>)

Y para exigir integridades referenciales:

ON DELETE <opción_referencia> ON UPDATE <opción_referencia>

Donde <*opción_referencia*> puede ser:

- RESTRICT: rechaza la operación de eliminación o actualización de la tabla principal.
- CASCADE: elimina o actualiza la fila de la tabla principal y automáticamente también borra o actualiza los registros coincidentes en las tablas secundarias.
- SET NULL: elimina o actualiza la fila de la tabla principal y establece el campo de la clave externa de la tabla secundaria a NULL.
- NO ACTION: rechaza la operación de eliminación o actualización de la tabla principal, si hay una relación de valor foráneo con alguna otra tabla.

EJEMPLO 7.1

En este ejemplo vamos a crear las tablas que vamos a emplear en los ejemplos de este capítulo:

```
CREATE TABLE Alumnos (
DNI_Al CHARACTER(9) NOT NULL,
Nombre_Al CHARACTER(50) NOT NULL,
Apellido1_Al CHARACTER(50) NOT NULL,
Apellido2_Al CHARACTER(50) NOT NULL,
Edad_Al INTEGER,
Teléfono_Al CHARACTER(9),
Dirección_Al CHARACTER(100),
Ciudad_Al CHARACTER(20),
PRIMARY KEY (DNI_Al) );
CREATE TABLE Asignaturas (
Cód_As INTEGER NOT NULL,
Nombre_As CHARACTER(50) NOT NULL,
Créditos_As INTEGERNOT NULL,
Facultad_As CHARACTER(50),
DNI_P CHARACTER(9) REFERENCES Profeosres(DNI_P) ON UPDATE CASCADE ON DELETE CASCADE,
PRIMARY KEY (Cód_As));
CREATE TABLE Estudian (
DNI_Al CHARACTER(9) NOT NULL REFERENCES Alumnos(DNI_Al) ON UPDATE CASCADE ON DELETE
CASCADE,
Cód_As INTEGER NOT NULL REFERENCES Asignaturas(Cód_As) ON UPDATE CASCADE ON DELETE
CASCADE,
Nota_Al_As INTEGER,
PRIMARY KEY (DNI_Al, Cód_As));
```

```
CREATE TABLE Profesores (
DNI_P CHARACTER(9) NOT NULL,
Nombre_P CHARACTER (50) NOT NULL,
Apellido1_P CHARACTER (50) NOT NULL,
Apellido2_P CHARACTER (50) NOT NULL,
NCuenta_P CHARACTER (20),
Teléfono_P CHARACTER (9),
Especialidad_P CHARACTER (20),
Dirección_P CHARACTER (100),
Ciudad_PCHARACTER (20),
PRIMARY KEY (DNI_P));
```

Modificación de tablas

Podemos emplear la instrucción *ALTER TABLE* para modificar la estructura de una tabla, añadiendo o modificando campos.

Sintaxis:

ALTER TABLE <nombre_tabla>

{ ADD (<descripción_campo>*[11]) | MODIFY (<descripción_campo>*) | DROP (<nombre_campo>*) };

EJEMPLO 7.2

Añadiremos el campo *Convocatoria_Al_As en la tabla Estudian*

```
ALTER TABLE Estudian ADD Convocatoria_Al_As INT;
```

Eliminación de tablas

Si contamos con los privilegios adecuados, también podremos eliminar tablas.

Sintaxis:

DROP TABLE <nombre_tabla>;

EJEMPLO 7.3

Eliminar la tabla *Profesor*.

```
DROP TABLE Profesor;
```

11 El carácter "*" significa que puede haber ninguna o muchas ocurrencias de este elemento.

7.1.3 CREACIÓN, MODIFICACIÓN Y BORRADO DE VISTAS

La vista tiene la apariencia de una tabla, pero, a diferencia de esta, ocupa menos espacio, debido a que se almacena solo la definición y no sus datos.

El objetivo de la vista es mostrar al usuario una visión parcial de los datos, de modo que podemos mostrar a estos solo aquellos datos que son de su interés. Esto constituye un elemento de seguridad importante. Otro motivo para su utilización es la adaptación del esquema de datos para un determinado usuario o aplicación.

Creación de vistas

Se crean aplicando una consulta a la base de datos, pero estas quedan almacenadas, posibilitando su reutilización.

Sintaxis:

CREATE VIEW <nombre_vista> AS <subconsulta>;

EJEMPLO 7.4

Crear una vista con un listado de *Profesores* y sus *Teléfonos*.

```
CREATE VIEW Tfno_Profe
AS SELECT Nombre_P, Teléfono_P
FROM Profesor;
```

NOTA

Si a la hora de crear la vista empleamos el comando REPLACE en lugar de CREATE, conseguimos que, en caso de que ya exista una vista con el nombre repetido, no se producirá un error, sino que se sustituirá la antigua vista por la recién creada.

Modificación de vistas

Sintaxis:

ALTER VIEW <nombre_vista> AS <subconsulta>;

Eliminación de vistas

Sintaxis:

DROP VIEW <nombre_vista> [RESTRICT | CASCADE];

Donde:

- RESTRICT: indica que la vista no se borrará si está referenciada, por ejemplo, por otra vista.
- CASCADE: indica que todo lo que referencie a la vista se borrará con ella.

EJEMPLO 7.5

Eliminar la vista *Tfno_Profe*.

```
DROP VIEW Tfno_Profe;
```

7.1.4 CREACIÓN, MODIFICACIÓN Y BORRADO DE ÍNDICES

Los índices son un elemento que permiten acceder a la información de una tabla, mediante un determinado orden, de forma rápida, aumentando el rendimiento de los accesos a la información.

Creación de índices

Sintaxis:

```
CREATE INDEX <nombre_índice> ON <nombre_tabla> (<campo>*);
```

EJEMPLO 7.6

Crear un índice sobre la tabla *Profesor*.

```
CREATE INDEX Ind_Profe ON Profesor (DNI_P);
```

Eliminación de índices

Sintaxis:

```
DROP INDEX <nombre_índice>;
```

EJEMPLO 7.7

Eliminar el índice sobre la tabla *Profesor* creado anteriormente.

```
DROP INDEX Ind_Profe;
```

7.1.5 ESPECIFICACIÓN DE RESTRICCIONES DE INTEGRIDAD

Pese a que ya lo hemos adelantado al comienzo de este capítulo, seguidamente vamos a explicar las principales restricciones de integridad, clasificándolas, a su vez, en restricciones sobre columnas y restricciones de tabla.

Restricciones de columna

En cada una de las columnas de la tabla (campos), una vez hemos definido el nombre y el dominio (o tipo de datos), podemos añadir restricciones, las cuales tendrán que ser cumplidas con obligatoriedad. Las principales restricciones que se aplican a una columna de una determinada tabla son:

- **NOT NULL:** indica que la columna no puede tener valores nulos.
- **UNIQUE:** indica que la columna no puede tener valores repetidos. Es una clave alternativa.
- **PRIMARY KEY:** indica que la columna no puede tener valores repetidos ni nulos. Es la clave primaria.
- **REFERENCES <tabla> [<columna>]:** en este caso, la columna es la clave foránea de la columna de la tabla especificada.
- **CHECK (condiciones):** la columna debe cumplir las condiciones especificadas.

Restricciones de tabla

Tras definir una tabla e indicar sus columnas (indicando las restricciones sobre ellas), también se puede aplicar restricciones sobre la propia tabla, que, al igual que en las columnas, se deberá cumplir obligatoriamente. Las restricciones sobre tablas son:

- **UNIQUE <nombre_campo/s>:** indica que el conjunto de las columnas especificadas no puede tener valores repetidos. Es una clave alternativa.
- **PRIMARY KEY <nombre_campo/s>:** el conjunto de columnas o campos especificados no pueden tener valores nulos ni repetidos. Es una clave primaria.
- **FOREIGN KEY <nombre_campo/s> REFERENCES <nombre_tabla> (<nombre_campo/s2>):** indica que el conjunto de los campos especificados es una clave foránea que referencia la clave primaria formada por el conjunto de las columnas2 de la tabla dada. Si las columnas y las columnas2 se denominan exactamente igual, entonces no sería estrictamente necesario poner columnas2 (nosotros lo haremos siempre).
- **CHECK (condiciones):** la columna debe cumplir las condiciones especificadas.

7.2 EL LENGUAJE DE MANIPULACIÓN DE DATOS (DML)

En la manipulación de datos (*Data Manipulation Language*, DML) se incluyen las operaciones de inserción, eliminación y modificación de registros en las diferentes tablas, así como la realización de consultas de selección sobre estas.

7.2.1 CONSTRUCCIÓN DE CONSULTAS DE SELECCIÓN: AGREGACIÓN, SUBCONSULTAS, UNIÓN, INTERSECCIÓN, DIFERENCIA

Para realizar las consultas de selección emplearemos la siguiente *sintaxis*:

SELECT { [DISTINCT]<campo_a_mostrar> [<alias>] (,[DISTINCT] <campo_a_mostrar> [<alias>])* | (*) }
FROM <nombre_tabla> (,<nombre_tabla>)*
[WHERE <condición>]
[ORDER BY <nombre_campo> [DESC] (,<nombre_campo> [DESC])*];

Teniendo en cuenta lo siguiente:

- En la **cláusula SELECT** podemos indicar qué elementos queremos mostrar en la consulta con la posibilidad de escribir "*" y mostrar, de este modo, todos los campos que intervienen en la citada consulta.
- Mediante la **cláusula FROM** indicaremos las tablas sobre las que se efectuarán la consulta.
- La **cláusula WHERE** sirve para indicar la condición a cumplir por los registros de la consulta.
- Por su parte, la **cláusula ORDER BY** nos permite ordenar los registros seleccionados en la consulta en base a unos atributos especificados, por orden ascendente o descendente.

Además, también podemos agrupar los registros en grupos, con la opción **GROUP BY**, y establecer condiciones sobre dichos grupos, mediante **HAVING** (más adelante veremos estos operadores).

EJEMPLO 7.8

Obtener el *Nombre_Al* y la *Edad_Al* de los *Alumnos* matriculados en *Programación*.

SELECT Nombre_Al, Edad_Al
FROM Alumnos, Estudian, Asignaturas
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Estudian.Cód.As = Asignaturas.Cód.As
AND Nombre_As = 'Programación';

A continuación, veremos, con mayor detenimiento, los operadores y los diferentes tipos de consulta.

Operador DISTINCT

Para evitar que en una consulta aparezcan resultados duplicados, utilizaremos la opción DISTINCT.

EJEMPLO 7.9

Mostrar las edades de los *Alumnos*.

SELECT DISTINCT Edad_Al
FROM Alumnos;

Predicados

En la **cláusula WHERE** podemos especificar una condición lógica, donde también podremos hacer uso de los operadores relacionales (<, >, >=, <=, =, ≠) mediante los cuales podemos efectuar comparaciones.

Además, podemos hacer uso de los conectores lógicos (AND, OR, NOT) con los que evaluar condiciones más complejas.

Operador BETWEEN

En aquellas comparaciones en las que debemos evaluar si un determinado atributo toma un valor comprendido entre un rango, utilizaremos este operador.

EJEMPLO 7.10

Mostrar los *Alumnos* cuyas edades (Edad_Al) están comprendidas entre 18 y 22 años.

```
SELECT (*)
FROM Alumnos
WHERE Edad_Al BETWEEN 18 AND 22;
```

Operador LIKE

En condiciones en las que intervienen cadenas de caracteres puede ser deseable poder efectuar comparaciones con respecto a un patrón.

También se usarán los caracteres especiales.

- % → Representa cualquier subcadena
- _ → Representa un solo carácter
- \ → Escape

EJEMPLO 7.11

Mostrar los *Nombre_P* y *Teléfono_P* cuyo primer apellido (*Apellido1_P)* comience por "P".

```
SELECT Nombre_P, Teléfono_P
FROM Profesor
WHERE Apellidos_P LIKE 'P%';
```

Pertenencia a un conjunto

Podemos determinar si un atributo pertenece a un determinado conjunto usando el operador **IN**.

EJEMPLO 7.12

Mostrar *Nombre_Al* y primer apellido (*Apellido1_Al*) de *Alumnos* que reciban clase en asignaturas que imparta el *Profesor* "Pepe Pérez Pérez".

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Estudian.Cód_As     IN    (SELECT Cód_As
                               FROM Asignaturas, Profesor
                               WHERE Asignaturas.DNI_P = Profesor.DNI_P
                               AND Nombre_P = 'Pepe' AND Apellidos_P =
                               'Pérez Pérez' );
```

Lo contrario, es decir, la no pertenencia, sería utilizando la expresión **NOT IN**.

Comparaciones de conjuntos

En SQL podemos hacer comparaciones entre conjuntos de registros, para lo cual utilizamos los operadores **SOME/ANY** y **ALL**, que se acompañan de los operadores relacionales.

EJEMPLO 7.13

***Alumnos* con la máxima *Nota* en alguna *Asignatura*.**

```
SELECT (*)
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Nota_Al_As>= ALL (SELECT Nota_Al_As
                               FROM Estudian);
```

NOTA

Como veremos más adelante, este es un ejemplo de varias consultas anidadas, también llamada consulta con subconsultas relacionadas.

Básicamente lo que hace la subconsulta es mostrar todas las notas de la base de datos, mientras que la consulta principal muestra los datos de los alumnos cuya nota sea superior o igual a **todas** (mediante el operador **ALL**) las notas (cosa que nos ofrece la subconsulta).
Por otro lado, el operador **SOME/ANY** no necesita evaluar todo el conjunto, pues basta con que se cumpla en uno de sus miembros para que sea cierto en la consulta principal.

Variables de relación

SQL permite utilizar variables para referirnos a una relación (o tabla), pudiendo emplear la variable en lugar del nombre de la relación.

Para definirla, basta con indicar el nombre de dicha variable, junto al nombre de la relación, en la cláusula FROM.

EJEMPLO 7.14

***Alumnos* que residen en la misma *Ciudad_Al* que "Fran Martínez".**

```
SELECT A1.Nombre_Al
FROM Alumnos A1, Alumnos A2
WHERE A1.Ciudad_Al = A2.Ciudad_Al AND A2.Nombre_Al = 'Fran' AND
A2.Apellido1_Al = 'Martínez';
```

Cláusula ORDER BY (ordenación)

La **cláusula SELECT** incluye una opción para ordenar las relaciones devueltas por ésta. El orden se indica mediante **ORDER BY**, indicando el campo (o campos) por el cual vamos a ordenar.

Por defecto, la ordenación se lleva a cabo de manera ascendente, pero podemos indicar lo contrario, si añadimos al nombre del atributo la palabra reservada **DESC**.

EJEMPLO 7.15

***Nombre_Al* y *Apellido1_Al* de *Alumnos*, por orden de *Edad_Al*, de forma descendente.**

```
SELECT Nombre_Al, Apellidos_Al, Edad_Al
FROM Alumnos
ORDER BY Edad_Al DESC;
```

Cláusula GROUP BY (agrupación)

Otra posibilidad interesante es la agrupación de relaciones en el resultado, de manera que podamos aplicar funciones sobre dichos grupos de relaciones.

Para agrupar, se emplea la cláusula **GROUP BY**, seguida del atributo o atributos por el cual deseamos agrupar.

Existen además algunas funciones que podemos aplicar sobre los grupos resultantes, como son:

- **AVG:** media aritmética.
- **MAX:** máximo valor.
- **MIN:** mínimo valor.
- **SUM:** suma.
- **COUNT:** número de elementos.

Además, podemos aplicar una condición al grupo de relaciones, para seleccionar qué grupo o grupos nos interesa devolver y cuál o cuáles descartar. Para ello, utilizamos la cláusula **HAVING** seguida de la condición que se debe cumplir.

EJEMPLO 7.16

***Alumnos* matriculados en más de 3 *Asignaturas*.**

```
SELECT Alumnos.DNI_Al, Alumnos.Nombre_Al
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
GROUP BY Alumnos.DNI_Al
HAVING COUNT(*) > 3;
```

7.2.2 CONSTRUCCIÓN DE CONSULTAS DE INSERCIÓN

La *sintaxis* para una consulta de inserción sería la siguiente:

```
INSERT INTO <nombre_tabla>
[(<campo> (, <campo>)* )]
VALUES (<valor> (,<valor>)*);
```

EJEMPLO 7.17

En este ejemplo insertaremos nuevas filas a diversas tablas.

```
INSERT INTO Asignaturas
VALUES (0012102121, "Programación", 7, "Escuela Superior de Ingeniería",
"11111111A"), (0033338844, "Bases de Datos", 6, "Escuela Superior de Ingeniería",
"22222222B");
INSERT INTO Alumnos
(DNI_Al, Nombre_Al, Apellido1_Al, Apellido2_Al, Teléfono_Al)
VALUES ("23232323C", "José", "Pérez","Dil", "111111111"));
```

7.2.3 CONSTRUCCIÓN DE CONSULTAS DE MODIFICACIÓN

La *sintaxis* para una consulta de modificación sería la siguiente:

```
UPDATE <nombre_tabla>
SET <nombre_campo> = <valor> (,<nombre_campo> = <valor>)*
[WHERE <condición>];
```

EJEMPLO 7.18

En este ejemplo asignaremos un nuevo *Nombre_P* y *Apellido1_P* al *Profesor* cuyo *DNI_P* sea *75248143D*.

```
UPDATE Profesor
SET Nombre_P = 'Pedro', Apellido1_P = 'Pérez'
WHERE DNI_P = '75248143D';
```

7.2.4 CONSTRUCCIÓN DE CONSULTAS DE BORRADO

La *sintaxis* para una consulta de borrado sería la siguiente:

```
DELETE FROM <nombre_tabla> [WHERE <condición>];
```

EJEMPLO 7.19

En este ejemplo eliminaremos los *Profesores* cuyo primer apellido (*Apellido1_P*) sea "Martínez".

```
DELETE FROM Profesor WHERE Apellido1_P LIKE '%Martínez%';
```

7.3 CLÁUSULAS DEL LENGUAJE PARA LA AGRUPACIÓN Y ORDENACIÓN DE LAS CONSULTAS

RECUERDA

Las cláusulas fundamentales para estas dos operaciones, GROUP BY para agrupación y ORDER BY para ordenación, ya las hemos estudiado en el apartado 7.2.1. "Construcción de consultas de selección: Agregación, Subconsultas, Unión, Intersección, Diferencia".
No obstante, en este apartado las analizaremos en mayor profundidad e incluiremos una mayor cantidad de ejemplos de uso.

Consultas con agrupación de filas de una tabla (GROUP BY)

Para ello, añadiremos a la consulta base (estructurada como SELECT – FROM - [WHERE]) los siguientes elementos:

- **GROUP BY:** este operador es indispensable para la agrupación, ya que es el que permite agrupar las filas, de acuerdo a una/s columna/s concretas, que también se deben indicar en la consulta.
- **HAVING:** se trata de un operador opcional, que se utiliza en caso de necesitar especificar condiciones de búsqueda para los grupos de filas establecidos por el GROUP BY.

  Su utilidad es similar al WHERE pero, en este caso, la condición la deben cumplir los grupos formados, no cada fila (tupla) de manera independiente.

EJEMPLO 7.20

Mostrar las *Notas* que tiene cada *Alumno* y la *Asignatura* a la que corresponde la misma.

```
SELECT Nombre_Al, Nombre_As, Nota_Al_As
FROM Alumnos, Estudian, Asignaturas
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Estudian.Código_As = Asignaturas.Código_As
GROUP BY Nombre_Al;
```

NOTA

Los factores de agrupación de la cláusula GROUP BY deben ser, como mínimo, las columnas que figuran en el SELECT, salvo columnas afectadas por funciones de agregación (SUM, AVG, COUNT…).
Del mismo modo, también se puede agrupar en base a más de una columna, como veremos en el siguiente ejemplo.

EJEMPLO 7.21

Mostrar las *Notas* que ha puesto cada *Profesor* para cada una de las *Asignaturas* que imparte.

```
SELECT Nombre_P, Nombre_As, Nota_Al_As
FROM Profesores, Estudian, Asignaturas
WHERE Profesores.DNI_P = Asignaturas.DNI_P
AND Estudian.Código_As = Asignaturas.Código_As
GROUP BY Nombre_P, Nombre_As;
```

Seguidamente, veremos un ejemplo de inclusión de la cláusula HAVING a una consulta agrupada.

EJEMPLO 7.22

Mostrar las *Notas* que tiene cada *Alumno* y la *Asignatura* a la que corresponde la misma PERO solo para alumnos que cursan más de dos asignaturas.

```
SELECT Nombre_Al, Nombre_As, Nota_Al_As
FROM Alumnos, Estudian, Asignaturas
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Estudian.Código_As = Asignaturas.Código_As
GROUP BY Nombre_Al HAVING COUNT(*) > 2;
```

Como también hemos comentado anteriormente, las agrupaciones se pueden combinar con el uso de funciones de agregación, como se puede ver en el ejemplo anterior (en el que hemos empleado el operador COUNT para contar el número de filas de cada grupo y filtrar las que sean mayores a 2) y en este siguiente:

EJEMPLO 7.23

Mostrar la *Nota* media, mínima y máxima que ha puesto cada *Profesor* para cada una de las *Asignaturas* que imparte.

```
SELECT Nombre_P, Nombre_As, AVG(Nota_Al_As), MAX(Nota_Al_As),
MIN(Nota_Al_As)
FROM Profesores, Estudian, Asignaturas
WHERE Profesores.DNI_P = Asignaturas.DNI_P
AND Estudian.Código_As = Asignaturas.Código_As
GROUP BY Nombre_P, Nombre_As;
```

Consultas con ordenación de filas en el resultado (ORDER BY)

Para esta opción, añadiremos a la consulta base (estructurada como SELECT – FROM - [WHERE]) la palabra reservada ORDER BY junto al campo o campos sobre los que se desee ordenar el resultado.

EJEMPLO 7.24

Mostrar un listado de *Alumnos* y sus *Teléfono_Al* ordenados alfabéticamente.

```
SELECT Apellido1_Al, Apellido2_Al, Nombre_Al, Teléfono_Al
FROM Alumnos
ORDER BY Apellido1_Al, Apellido2_Al, Nombre_Al;
```

Si no se especifica nada más, se seguiría un orden ascendente, pero si necesitamos generar un orden descendente en alguno de los campos sobre los que se ordena, al nombre del campo de ordenación le añadiremos el modificador DESC para cambiar dicho orden y convertirlo en descendente.

EJEMPLO 7.25

Mostrar un listado de *Alumnos* ordenados por su *Edad_Al* de mayor a menor y, en caso de igualdad, ordenados alfabéticamente.

```
SELECT Apellido1_Al, Apellido2_Al, Nombre_Al
FROM Alumnos
ORDER BY Edad_Al DESC, Apellido1_Al, Apellido2_Al, Nombre_Al;
```

NOTA

Como podemos observar en la consulta anterior, al contrario que con la cláusula GROUP BY, en este caso no es necesario mostrar en SELECT todos los campos empleados en la ordenación.

7.4 CAPACIDADES ARITMÉTICAS, LÓGICAS Y DE COMPARACIÓN DEL LENGUAJE

RECUERDA

En el apartado 7.2.1. "Construcción de consultas de selección: Agregación, Subconsultas, Unión, Intersección, Diferencia" ya tuvimos la oportunidad de presentar los distintos operadores lógicos y de comparación del lenguaje existentes.

No obstante, en este apartado los analizaremos en mayor profundidad e incluiremos una mayor cantidad de ejemplos de su uso. Del mismo modo, presentaremos los operadores aritméticos.

Operadores aritméticos

Se emplean para realizar cálculos aritméticos. Se trata de los ya conocidos: '+', '-' , '*' , '/' y '%'.
Devuelven un valor numérico como resultado de realizar los cálculos indicados.
Algunos de ellos se pueden utilizar también con **fechas**:

- **fecha1 - fecha2** devuelve el número de días que hay entre las fechas 1 y 2.
- **fecha + n** devuelve una fecha que es el resultado de sumar *n* días a la fecha.

Operadores de concatenación

Para unir dos o más cadenas se utiliza el operador de concatenación "||".

EJEMPLO 7.26

La expresión *'Francisco'* || *'Javier'* daría como resultado *'FranciscoJavier'*.

EJEMPLO 7.27

¿Cómo quedarían las *Notas* de "Bases de datos" si sumamos 1 punto a todos los alumnos?

```
SELECT Nombre_Al, Apellido1_Al, Nota + 1
FROM Alumnos, Estudian, Asignaturas
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Estudian.Código_As = Estudian.Código_As
AND Nombre_As = "Bases de datos";
```

Operadores lógicos

Ya hemos indicado que los operadores de comparación devuelven un valor lógicos de tipo *verdadero*/*falso* (*true*/*false*). En ocasiones, se necesita trabajar con varias expresiones de comparación (por ejemplo cuando queremos formar una expresión de búsqueda que cumpla dos condiciones), en cuyo caso debemos recurrir a los operadores lógicos **AND**, **OR** y **NOT**, cuya tabla de verdad es la que se muestra a continuación:

P	Q	p AND q	p OR q	NOT p
Verdadero	**Verdadero**	Verdadero	Verdadero	Falso
Verdadero	**Falso**	Falso	Verdadero	
Falso	**Verdadero**	Falso	Verdadero	Verdadero
Falso	**Falso**	Falso	Falso	

EJEMPLO 7.28

Mostrar los alumnos que no han sacado una nota entre 7 y 9 o que viven en Almería.

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND (Nota_Al_As NOT BETWEEN 7 AND 9 OR Ciudad_Al = "Almería");
```

Operadores de comparación

Las expresiones formadas con operadores de comparación dan como resultado un valor de tipo *verdadero/falso* (*true/false).*

Los principales operadores de comparación son:

- **Igual:** =
- **Distinto:** != ó <>
- **Menor que:** <
- **Menor o igual que:** <=
- **Mayor que:** >
- **Mayor o igual que:** >=

También existen otros operadores como **LIKE** (para cadenas de caracteres), **BETWEEN** (para intervalos), **IN** (para conjunto de datos), **IS** o **IS NULL** (especialmente útil para subconsultas), etc.

EJEMPLO 7.29

Mostrar los alumnos que han aprobado alguna *Asignatura*.

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Nota_Al_As >= 5;
```

7.5 FUNCIONES AGREGADAS DEL LENGUAJE

RECUERDA

En el apartado 7.2.1. "Construcción de consultas de selección: Agregación, Subconsultas, Unión, Intersección, Diferencia" y 7.3. "Cláusulas del lenguaje para la agrupación y ordenación de las consultas", ya tuvimos la oportunidad de presentar este tipo de funciones.
No obstante, en este apartado las analizaremos en mayor profundidad e incluiremos una mayor cantidad de ejemplos de su uso.

SQL permite agrupar las filas de una tabla o las seleccionadas como resultado de una consulta y obtener resultados calculados a partir de operaciones con dichas filas.

Las salidas obtenidas son los resultados de aplicar funciones de columna a los grupos de filas previamente seleccionados. Estas funciones de columna pueden ser:

Funciones de columna	Descripción
SUM (*expresión / [DISTINCT] columna)*	Calcula la suma de los valores de la expresión o del campo de esa columna (si se emplea *DISTINCT* no incluye los valores repetidos)
AVG (*expresión / [DISTINCT] columna)*	Calcula la media aritmética de los valores de la expresión o del campo indicado
MIN (*expresión)*	Devuelve el valor mínimo de los valores de la expresión
MAX (*expresión*)	Devuelve el valor máximo de los valores de la expresión
COUNT (*[DISTINCT] columna / **)	Devuelve el número de valores de cada columna o el número de filas existentes (aunque tengan valores NULL)

EJEMPLO 7.30

Mostrar la *Nota_Al_As* media del alumno "Fran Martínez".

```
SELECT AVG(Nota_Al_As)
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Alumnos.Nombre_Al = "Fran"
AND Alumnos.Apellido1_Al = "Martínez";
```

EJEMPLO 7.31

Mostrar, de la *Asignatura* "Programación", la nota máxima, mínima y la diferencia entre ambas. Devolver también el número de alumnos que la han cursado.

```
SELECT MAX(Nota_Al_As), MIN(Nota_Al_As), MAX(Nota_Al_As) - MIN(Nota_Al_As), COUNT (Alumnos.DNI_Al)
FROM Asignaturas, Estudian
WHERE Asignaturas.Código_As = Estudian.Código_As
AND Asignaturas.Nombre_As = "Programación";
```

RECUERDA

Estas funciones de columnas, además de operar sobre datos numéricos, también se pueden emplear con otros tipos de datos, como por ejemplo fechas, como podemos ver en el Ejemplo 7.33.

EJEMPLO 7.32

Devolver la fecha de nacimiento del *Alumno* mayor (con *Fecha_Nac_Al* menor) (NOTA: suponemos dicho nuevo campo en la tabla *Alumnos*).

```
SELECT MIN(Fecha_Nac_Al)
FROM Alumnos;
```

Como ya comentamos en los diferentes apartados en el que hemos visto esta cláusula, este tipo de operadores se suelen utilizar junto al binomio GROUP BY y HAVING, obteniendo operaciones de los grupos creados por dichas cláusulas. Lo veremos en el siguiente ejemplo.

EJEMPLO 7.33

Mostrar la *Nota_Al_As* media de cada *Alumno*.

```
SELECT DNI_Al, Nombre_Al, AVG(Nota_Al_As)
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
GROUP BY Alumnos.DNI_Al;
```

7.6 TRATAMIENTO DE VALORES NULOS

En SQL, la ausencia de valor se expresa como valor nulo (*NULL*). Esta ausencia de valor o valor nulo **no equivale** en modo alguno al valor 0, por lo que un campo que conforme una clave primaria nunca podrá contener valores nulos (aunque si puede contener dicho valor *Null*, por ejemplo, una clave ajena/foránea).

De esta forma, cualquier expresión aritmética que contenga algún valor nulo retornará un valor nulo.

EJEMPLO 7.34

Si en la tabla *Estudian* tenemos los siguientes valores (NOTA: Duplicamos el campo *Nota_Al_As* en dos notas parciales: *Nota1_Al_As* y *Nota2_Al_As*).

DNI_Al	Código_As	Nota1_Al_As	Nota2_Al_As
11111111A	001	7	10
11111111A	002	8	Null
22222222B	001	Null	5
22222222B	003	4	8
33333333C	003	9	7

Si queremos mostrar la nota media de cada alumno en cada asignatura ((Nota1_Al_As+Nota2_Al_As)/2) mediante la consulta:

SELECT DNI_Al, Código_As, (Nota1_Al_As+Nota2_Al_As)/2 AS "Nota Media"
FROM Estudian;
el resultado de la consulta sería:

DNI_Al	Código_As	Nota Media
11111111ª	001	8.5
11111111ª	002	Null
22222222B	001	Null
22222222B	003	6
33333333C	003	8

RECUERDA

En lo que se refiere a las operaciones lógicas (AND, OR, NOT) nos regiremos por lo explicado en la tabla de la sección 3.4.1 "Valor "Null" en el modelo".

7.7 CONSTRUCCIÓN DE CONSULTAS ANIDADAS

Las "consultas anidadas" o "consultas con subconsultas" se dan cuando existe una consulta incluida dentro de las condiciones de una cláusula *WHERE* o *HAVING* de otra consulta. Este se debe a que, en ciertas ocasiones, para expresar según qué condiciones no hay más remedio que obtener el valor que buscamos como resultado de una consulta.

EJEMPLO 7.35

***Alumnos* cuya nota en *Bases de Datos* superen la media de la clase.**

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos, Estudian, Asignaturas
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Estudian.Código_As = Asignaturas.Código_As
AND Nombre_As = "Bases de Datos"
AND Nota_Al_As >    (SELECT AVG(Nota_Al_As)
                     FROM Estudian, Asignaturas
                     WHERE Estudian.Código_As = Asignaturas.Código_As
                     AND Nombre_As = "Bases de Datos");
```

Como podemos ver en el Ejemplo 7.36, para llevar a cabo la consulta solicitada tenemos que hacer dos pasos:

1. En primer lugar, calculamos la nota media de la asignatura indicada. Esta sería la subconsulta.

   Dicha subconsulta nos devolvería un único valor (la nota media de esa asignatura).

2. Posteriormente, la consulta principal mostraría los alumnos cuya nota en dicha asignatura fuera superior a la nota media (obtenida previamente por la subconsulta).

Cuando trabajamos con consultas anidadas, podemos encontrarnos dos casos para los que hay que trabajar de manera diferente:

- **Subconsultas que generan valores simples:** en cuyo caso las condiciones de las comparaciones lógicas seguirían siendo las que se utilizan habitualmente: =, <>, <, >, <=, >=, BETWEEN, LIKE…
- **Subconsultas que generan conjuntos de valores:** en cuyo caso tendríamos que trabajar con operadores que establecen comparaciones con dichos conjuntos de valores.

  De estos operadores que trabajan con conjuntos podemos destacar los siguientes:

Operador	Descripción
[NOT] IN	Este operador comprueba que un valor coincide con alguno de los elementos de una lista (dada o generada por una subconsulta). En caso de utilizarse con el modificador NOT, se comprobará que el valor no esté incluido en dicha lista.
ANY / SOME	Combinado con una comparación lógica (<, >, <=, >=), estos operadores comprueban si un valor cumple dicha comparación lógica para algún valor del conjunto de valores con los que se compara.
ALL	Combinado con una comparación lógica (<, >, <=, >=), este operador comprueba si un valor cumple dicha comparación lógica para todos los valores del conjunto de valores con los que se compara.
[NOT] EXISTS	Se emplea para comprobar si una subconsulta produce alguna fila de resultados (EXISTS) o no produce ningún resultado (NOT EXISTS).

EJEMPLO 7.36

Alumnos* que están matriculados en asignaturas que imparte *Amalia Gallegos.

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Código_As IN    (SELECT Código_As
                     FROM Asignaturas, Profesores
                     WHERE Asignaturas.DNI_P = Profesores.DNI_P
                     AND Nombre_P = "Amalia"
                     AND Apellido1_P = "Gallegos");
```

EJEMPLO 7.37

***Alumno* con la mejor nota de toda la base de datos.**

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Nota_Al_As >= ALL       (SELECT Nota_Al_As
                             FROM Estudian);
```

7.8 UNIÓN, INTERSECCIÓN Y DIFERENCIA DE CONSULTAS

SQL cuenta con las operaciones unión, intersección y diferencia, definidas en el álgebra relacional, de modo que podemos operar sobre dos consultas como si se tratara de conjuntos de registros.

RECUERDA

Para hacer estas operaciones los conjuntos deben ser UNION-COMPATIBLES:
- Mismo número de campos en cada conjunto.
- Mismos dominios dos a dos.

Unión

Devuelve los registros que están en uno u otro conjunto.

EJEMPLO 7.38

Mostrar los nombres y teléfonos de todas las personas de la base de datos:

```
(SELECT Nombre_Al, Teléfono_Al
FROM Alumnos)
UNION
(SELECT Nombre_P, Teléfono_P
FROM Profesor);
```

NOTA

El comando *UNION* suprime los registros que estén duplicados en ambos conjuntos. En caso de que interese verlos duplicados, utilizaremos *UNION ALL*.

Intersección

Esta operación devuelve aquellos registros que están en ambos conjuntos.

EJEMPLO 7.39

Mostrar los nombres y teléfonos de *Profesores* que, a su vez, sean *Alumnos*:

```
(SELECT Nombre_Al, Teléfono_Al FROM Alumnos)
INTERSECT
(SELECT Nombre_P, Teléfono_P FROM Profesor);
```

Diferencia

Devuelve aquellos registros que pertenecen a un primer conjunto y no pertenecen al segundo.

EJEMPLO 7.40

Mostrar los nombres y teléfonos de *Profesores* que no sean *Alumnos*:

```
(SELECT Nombre_P, Teléfono_P
FROM Profesor)
MINUS
(SELECT Nombre_Al, Teléfono_Al
FROM Alumnos);
```

7.9 CONSULTAS DE TABLAS CRUZADAS

Una consulta de referencias cruzadas es aquella que nos permite visualizar los datos en filas y en columnas, como si fuera una tabla, a diferencia de como se nos están mostrando las consultas hasta este apartado (que es como se ejemplifica en el Ejemplo 7.41).

EJEMPLO 7.41

Dada la información del Ejemplo 7.35:

NI_Al	Código_As	Nota1_Al_As	Nota2_Al_As
11111111A	001	7	10
11111111A	002	8	Null
22222222B	001	Null	5
22222222B	003	4	8
33333333C	003	9	7

Si queremos llevar a cabo una consulta en la que se muestren la nota media de cada alumno en cada asignatura, tal y como hemos estado proponiendo hasta ahora, tendríamos que escribir:

```
SELECT DNI_Al, Código_As, (Nota1_Al_As+Nota2_Al_As)/2 AS "Nota Media"
FROM Estudian;
```

El resultado de la consulta sería:

DNI_Al	Código_As	Nota Media
11111111A	001	8.5
11111111A	002	Null
22222222B	001	Null
22222222B	003	6
33333333C	003	8

Frente a la forma de representar la información, las consultas de tablas cruzadas permiten mostrar el resultado de la consulta de la siguiente forma:

lumnos / Asignaturas	001	002	003
11111111ª	8.5	Null	
22222222B	Null		6
33333333C			8

Para conseguir esto, debemos seguir la siguiente *sintaxis*:

TRANSFORM <función_agregada>
<instrucción_select>
PIVOT <campo_pivot>
[IN <valor> (,<valor>)*]

Donde:

- **<función_agregada>:** es una función SQL que se emplea para agregar (agrupar) los datos seleccionados.
- **<instrucción_select>:** es una consulta encabezada por SELECT.
- **<campo_pivot>:** es el campo o expresión que se utilizará para crear las cabeceras de la columna en el resultado de la consulta final.
- **<valor>:** es el valor/es fijo/s utilizado/s para crear las cabeceras de la columna.

EJEMPLO 7.42

Para transformar el resultado de la consulta del Ejemplo 7.42 en una tabla cruzada, tendríamos que realizar la siguiente consulta:

```
TRANSFORM (Nota1_Al_As+Nota2_Al_As)/2
SELECT DNI_Al AS “Alumnos”
FROM Estudian
GROUP BY DNI_Al
PIVOT Cód_As;
```

EJEMPLO 7.43

Realizar una tabla que muestre la media de la *Nota1_Al_As* de cada *Profesor* en cada una de *Asignaturas* que imparte:

```
TRANSFORM AVG(Nota1_Al_As)
SELECT Nombre_As AS “Asignatura”, AVG(Nota1_Al_As)
FROM Estudian, Asignaturas, Profesores
WHERE Estudian.Código_As = Asignaturas.Código_As
AND Asignaturas.DNI_P = Profesores.DNI_P
GROUP BY Nombre_As
PIVOT Nombre_P;
```

7.10 OTRAS CLÁUSULAS DEL LENGUAJE

A lo largo de este capítulo, hemos realizado un importante número de consultas en las que hemos involucrado a más de una tabla, para lo cual nos hemos asegurado que cada dos tablas tenían un campo común, que sirvan para establecer la relación, y lo hemos especificado así en la cláusula WHERE.

En este apartado vamos a presentar un operador muy versátil (aunque no lo empleemos demasiado reconocemos su gran valía), como es el JOIN y sus diferentes variantes.

La sentencia JOIN permite consultar datos de dos o más tablas. Dichas tablas estarán relacionadas entre sí de alguna forma (como lo hemos hecho hasta ahora), a través de alguno de sus campos. El propósito de JOIN es unir información de diferentes tablas, para no tener que repetir datos entre ellas.

Seguidamente, veremos sus diferentes opciones y variantes:

INNER JOIN

Se trata de la sentencia JOIN por defecto y consiste en combinar cada fila de una tabla con cada una de las filas de la otra tabla, siempre y cuando se cumpla una determinada condición para poder combinarlas.

Sintaxis:
SELECT { [DISTINCT]<campo_a_mostrar> [<alias>] (,[DISTINCT] <campo_a_mostrar> [<alias>])* | (*) }
FROM <nombre_tabla1>INNER JOIN <nombre_tabla2> ON <condición_combinación>
[WHERE <condición>]
[ORDER BY <nombre_campo> [DESC] (,<nombre_campo> [DESC])*];

EJEMPLO 7.44

La consulta que hacíamos sin el operador JOIN en el Ejemplo 7.30 y que mostraba los alumnos que han aprobado alguna *Asignatura*:

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos, Estudian
WHERE Alumnos.DNI_Al = Estudian.DNI_Al
AND Nota_Al_As >= 5;
```

Se podría plantear de la siguiente forma y obtendríamos el mismo resultado:

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos INNER JOIN Estudian ON Alumnos.DNI_Al = Estudian.DNI_Al
WHERE Nota_Al_As >= 5;
```

NATURAL JOIN

En caso de existir campos con el mismo nombre en las relaciones que se combinan (como es el caso de todos los ejemplos en este libro), podríamos usar NATURAL JOIN. De este modo, no tendríamos que especificar la condición del "ON" y solo se incluirá (por defecto) uno de los campos repetidos en el resultado de la combinación.

Sintaxis:

```
SELECT { [DISTINCT]<campo_a_mostrar> [<alias>] (,[DISTINCT] <campo_a_mostrar> [<alias>])* | (*) }
FROM <nombre_tabla1>NATURAL JOIN <nombre_tabla2>
[WHERE <condición>]
[ORDER BY <nombre_campo> [DESC] (,<nombre_campo> [DESC])*];
```

EJEMPLO 7.45

La consulta del Ejemplo 7.45 la podríamos expresar con NATURAL JOIN de la siguiente forma:

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos NATURAL JOIN Estudian
WHERE Nota_Al_As >= 5;
```

INNER JOIN USING (<nombre_campo_común>)

A su vez, cuando nos encontramos con el caso anterior, es decir, que los campos de ambas tablas a combinar se repitan, podemos utilizar INNER JOIN USING(<nombre_campo>) con el mismo resultado.

Sintaxis:

```
SELECT { [DISTINCT]<campo_a_mostrar> [<alias>] (,[DISTINCT] <campo_a_mostrar> [<alias>])* | (*) }
FROM <nombre_tabla1>INNER JOIN <nombre_tabla2> USING(<nombre_campo_común>)
[WHERE <condición>]
[ORDER BY <nombre_campo> [DESC] (,<nombre_campo> [DESC])*];
```

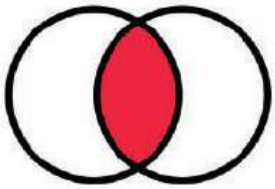
EJEMPLO 7.46

La consulta del Ejemplo 7.46 la podríamos expresar de la siguiente forma:

```
SELECT Nombre_Al, Apellido1_Al
FROM Alumnos INNER JOIN Estudian USING(DNI_Al)
WHERE Nota_Al_As >= 5;
```

NOTA

Podemos observar gráficamente que el resultado de la combinación utilizando WHERE, INNER JOIN, NATURAL JOIN y INNER JOIN USING(<nombre_campo_común>) es el mismo, es decir, las tuplas con un elemento común de dos relaciones.



LEFT/RIGHT/FULL OUTER JOIN

LEFT OUTER JOIN

Esta sentencia combina las filas de la primera tabla con las de la segunda tabla que cumplan una determinada condición de combinación. No obstante, a diferente de los operadores anteriores, si no existe coincidencia entre una fila de la primera tabla con otra de la segunda, se mantiene dicha fila y los valores correspondientes a la fila de la segunda tabla con valores *Null*.

Sintaxis:

SELECT { [DISTINCT]<campo_a_mostrar> [<alias>] (,[DISTINCT] <campo_a_mostrar> [<alias>])* | (*) }
FROM <nombre_tabla1>LEFT OUTER JOIN <nombre_tabla2> ON <condición_combinación>
[WHERE <condición>]
[ORDER BY <nombre_campo> [DESC] (,<nombre_campo> [DESC])*];

EJEMPLO 7.47

Si los datos de las tablas *Asignaturas* y *Profesores* en un momento dado fueran los siguientes (llevando a cabo una simplificación de campos):

Tabla "Asignaturas":

Cód_As	Nombre_As	...	DNI_P
001	Bases de Datos	...	11111111A
002	Redes	...	11111111A
003	Programación	...	22222222B
004	Tecnología	...	

Tabla "Profesores":

DNI_P	Nombre_P	...
11111111A	Francisco Javier	...
22222222B	Amalia	...
33333333C	María del Mar	...

NOTA

Esta situación se justifica debido a que la docente "María del Mar" no imparte docencia esa temporada y a que para la asignatura de "Tecnología" aún no se ha contratado docente para impartirla.

En esta situación, la consulta

```
SELECT *
FROM Asignaturas INNER JOIN Profesores ON Asignaturas.DNI_P = Profesores.DNI_P;
```

Mostraría el siguiente resultado:

Cód_As	Nombre_As	...	DNI_P	DNI_P	Nombre_P	...
001	Bases de Datos	...	11111111A	11111111A	Francisco Javier	...
002	Redes	...	11111111A	11111111A	Francisco Javier	...
003	Programación	...	22222222B	22222222B	Amalia	...

Por el contrario, la consulta

```
SELECT *
FROM Asignaturas LEFT OUTER JOIN Profesores ON Asignaturas.DNI_P = Profesores.DNI_P;
```

Mostraría este resultado:

Cód_As	Nombre_As	...	DNI_P	DNI_P	Nombre_P	
001	Bases de Datos	...	11111111A	11111111A	Francisco Javier	...
002	Redes	...	11111111A	11111111A	Francisco Javier	...
003	Programación	...	22222222B	22222222B	Amalia	...
004	Tecnología	...	Null	Null	Null	Null

RIGHT OUTER JOIN

Esta sentencia combina las filas de la primera tabla con las de la segunda tabla que cumplan una determinada condición de combinación. No obstante, a diferencia de los operadores anteriores, si no existe coincidencia entre una fila de la segunda tabla con otra de la primera, se mantiene dicha fila (de la segunda tabla) y los valores correspondientes a la fila de la primera tabla con valores *Null*.

Sintaxis:

SELECT { [DISTINCT]<campo_a_mostrar> [<alias>] (,[DISTINCT] <campo_a_mostrar> [<alias>])* | (*) }
FROM <nombre_tabla1>RIGHT OUTER JOIN <nombre_tabla2> ON <condición_combinación>
[WHERE <condición>]
[ORDER BY <nombre_campo> [DESC] (,<nombre_campo> [DESC])*];

EJEMPLO 7.48

Para la misma situación del Ejemplo 7.48, la consulta:

SELECT *
FROM Asignaturas RIGHT OUTER JOIN Profesores ON Asignaturas.DNI_P = Profesores.DNI_P;

Mostraría el siguiente resultado:

Cód_As	Nombre_As	...	DNI_P	DNI_P	Nombre_P	
001	Bases de Datos	...	11111111A	11111111A	Francisco Javier	...
002	Redes	...	11111111A	11111111A	Francisco Javier	...
003	Programación	...	22222222B	22222222B	Amalia	...
Null	Null	Null	Null	33333333C	María del Mar	

FULL OUTER JOIN

Y, por último, esta sentencia combina las filas de la primera tabla con las de la segunda tabla. La diferencia entre los operadores anteriores radica en que FULL OUTER JOIN siempre devolverá las filas de las dos tablas, aunque no cumplan la condición de combinación.

Sintaxis:

SELECT { [DISTINCT]<campo_a_mostrar> [<alias>] (,[DISTINCT] <campo_a_mostrar> [<alias>])* | (*) }
FROM <nombre_tabla1>FULL OUTER JOIN <nombre_tabla2> ON <condición_combinación>
[WHERE <condición>]
[ORDER BY <nombre_campo> [DESC] (,<nombre_campo> [DESC])*];

EJEMPLO 7.49

Para la misma situación del Ejemplo 7.48, la consulta:

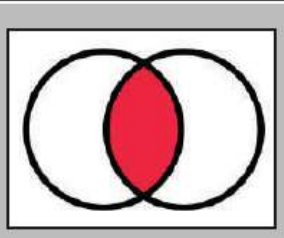
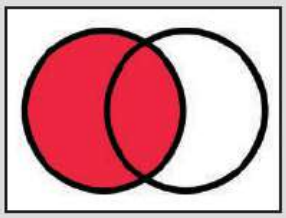
```
SELECT *
FROM Asignaturas FULL OUTER JOIN Profesores ON Asignaturas.DNI_P = Profesores.DNI_P;
```

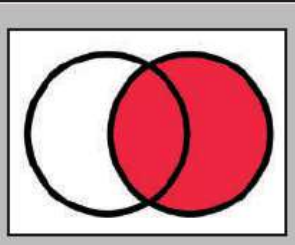
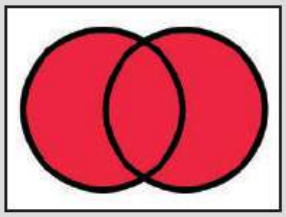
Mostraría el siguiente resultado:

Cód_As	Nombre_As	...	DNI_P	DNI_P	Nombre_P	...
001	Bases de Datos	...	11111111A	11111111A	Francisco Javier	...
002	Redes	...	11111111A	11111111A	Francisco Javier	...
003	Programación	...	22222222B	22222222B	Amalia	...
Null	Null	...	Null	33333333C	María del Mar	...
004	Tecnología	...	Null	Null	Null	

NOTA

Gráficamente podemos observar la diferencia entre las diferentes formas de combinación:

Operador	Simbolización gráfica de la operación
INNER JOIN, NATURAL JOIN ...	
LEFT OUTER JOIN	

Operador	Simbolización gráfica de la operación
RIGH OUTER JOIN	
FULL OUTER JOIN	

EJERCICIO GUIADO 7.1

Dado el diagrama Entidad-Relación del Ejercicio guiado 2.1 pasado a tablas en el Ejercicio guiado 5.1, tenemos el siguiente diseño lógico:

Supermercado (Código_S, Domicilio_S, Superficie_S, Teléfono_S)

Empleados (NIF_E, Nombre_E, Domicilio_E)

Capacidades_E (NIF_E, Capacidad_E)

Clientes (NIF_C, Nombre_C, Domicilio_C)

Artículos (Código_A, Descrip_A)

Pedidos (Código_S, Número_P, Fecha_P, Hora_P, Total_P, NIF_E, NIF_C)

Trabajan (Código_S, NIF_E, Función_ES)

Consta_de (Código_S, Número_P, Código_A, Cantidad_APS, Precio_APS)

A esta base de datos le podemos realizar las consultas del Ejercicio guiado 6.1 en SQL:

1. **Nombres de clientes que han comprado *langosta*:**

```
SELECT Nombre_C
FROM Clientes C, Pedidos P, Consta de D, Artículos A
WHERE C.NIF_C = P.NIF_C AND P.Código_S = D.Código_S AND
P.Número_P = D.Número_P AND D.Código_A = A.Código_A AND
Descrip_A = “langosta”;
```

2. **Nombre y domicilio de los empleados que no han entrado ningún pedido:**

```
(SELECT Nombre_E, Domicilio_E
FROM Empleados)
MINUS
(SELECT Nombre_E, Domicilio_E
FROM Empleados, Pedidos
WHERE Empleados.NIF_E = Pedidos.NIF_E)
```

3. **Nombres y domicilios de empleados que, a su vez, son clientes del supermerdado (han hecho algún pedido en alguna ocasión):**

```
(SELECT Nombre_E, Domicilio_E
FROM Empleados)
INTERSECT
(SELECT Nombre_C, Domicilio_C
FROM Clientes, Pedidos
WHERE Clientes.NIF_C = Pedidos.NIF_C)
```

4. **Fechas de pedidos que incluyen *tomates* y que han sido ordenados por *Amalia Gallegos*:**

```
SELECT Fecha_P
FROM Clientes C, Pedidos P, Consta de D, Artículos A
WHERE C.NIF_C = P.NIF_C AND P.Código_S = D.Código_S AND
P.Número_P = D.Número_P AND D.Código_A = A.Código_A AND
Descrip_A = "tomates" AND Nombre_C = "Amalia Gallegos";
```

5. **Función con la que trabaja el empleado *Francisco Javier Martínez* en los supermercados en lo que ha trabajado que tengan una superficie superior a 1000 m^2:**

```
SELECT Supermercado.Código_S, Función_ES
FROM Empleados, Trabajan, Supermercado
WHERE Empleados.NIF_E = Trabajan.NIF_E
AND Trabajan.Código_S = Supermercado.Código_S
AND Nombre_E = "Francisco Javier Martínez" AND Superficie_S > 1000;
```

Además, añadiremos las siguientes consultas:

6. **Código de los supermercados que tienen más de cinco empleados que trabajan como cajeros y cuatro que trabajan como reponedores:**

```
(SELECT Código_S  FROM Trabajan
WHERE Función = "Cajero"
GROUP BY Código_S HAVING COUNT(*) > 5)
INTESECT
(SELECT Código_S  FROM Trabajan
WHERE Función_ES = "Reponedor"
GROUP BY Código_S HAVING COUNT(*) = 4)
```

7. **Código, descripción y cantidad total de artículos pedidos por *María López*:**

```
SELECT A.Código_A, Descrip_A, SUM(Cantidad_APS)
FROM Clientes C, Pedidos P, Consta de D, Artículos A
WHERE C.NIF_C = P.NIF_C AND P.Número_P = D.Número_P
AND P.Código_S = D.Código_S AND D.Código_A = A.Código_A
AND Nombre_C = "María López"
GROUP BY A.Codigo_A;
```

7.11 EXTENSIONES DEL LENGUAJE

Debido a que se trata de elementos transversales a todo lo que estamos estudiando en este capítulo, la mayor parte de estas ya han sido estudiadas en el mismo, por lo que indicaremos dónde pueden ser consultadas.

En caso de no haber sido tratadas en otro apartado, al igual que hemos venido haciendo, explicaremos su principal cometido, hablaremos de su sintaxis y expondremos diversos ejemplos de uso.

7.11.1 CREACIÓN, MANIPULACIÓN Y BORRADO DE VISTAS

RECUERDA

En el apartado 7.1.3. "Creación, manipulación y borrado de vistas" ya tuvimos la oportunidad de presentar este tipo de funciones.

7.11.2 ESPECIFICACIÓN DE RESTRICCIONES DE INTEGRIDAD

La integridad de la base de datos asegura que se cumplen una serie de restricciones predefinidas. Concretamente existen tres tipos de posibles restricciones de integridad:

Restricciones sobre dominios

De esta forma podemos elegir los valores que queremos que tomen los atributos de nuestra base de datos, reduciéndolo a un conjunto que se especifique.

Sintaxis:

```
CRATE DOMAIN <nombre_dominio>
[AS <tipo_dato>]
[<Valor_por_defecto>]
[CONSTRAINT <Nombre_restricción>]
CHECK (<expresión_condicional>);
```

EJEMPLO 7.50

Crearemos una restricción de dominio para limitar el nombre de la asignatura a solo ciertos valores.

```
CREATE DOMAIN asignaturas CHAR(11)
CONSTRAINT restricción_asignaturas
CHECK (VALUE IN (“Bases de Datos”, “Programación”, “Redes”));
```

EJEMPLO 7.51

Definir la restricción de dominio para asegurarse que las notas introducidas estén entre 0 y 10.

```
CREATE DOMAIN Notas
AS FLOAT
DEFAULT 0
CONSTRAINT restricción_notas
CHECK (VALUE >=0 AND VALUE <=10);
```

Al igual que creamos restricciones de dominios, también nos interesará poder modificar o borrar dichas restricciones.

Eliminar dominios

Sintaxis:

DROP DOMAIN <nombre_dominio> {RESTRICT|CASCADE};

Donde:

- La opción **RESTRICT** hace que el dominio solo se pueda borrar si no se utiliza en ningún sitio.
- La opción **CASCADE** borra el dominio aunque esté referenciado, y pone el tipo de dato del dominio allí donde se utilizaba.

Modificar dominios

Sintaxis:
ALTER DOMAIN <nombre_dominio>
{<acción_modificar_dominio>|<acción_modificar_restricción_dominio>};

Donde:

- **<acción_modificar_dominio>** puede ser:
  - SET def_defecto
  - DROP DEFAULT
- **<acción_modificar_restricción_dominio>** puede ser:
  - ADD <restricciones_dominio>
  - DROP CONSTRAINT <nombre_restricción>

Restricciones generales

En este caso, se trata de restricciones generales sobre columnas de una o varias tablas.

Crear restricciones generales

Sintaxis:
CREATE ASSERTION <nombre_restricción>
CHECK (<expresión_condicional>)

EJEMPLO 7.52

Definir la restricción general para asegurarse que las notas introducidas estén entre 0 y 10.

```
CREATE ASSERTION Notas
CHECK (          SELECT * FROM Asignaturas
                 WHERE Nota_Al_As>=0 AND Nota_Al_As<=10);
```

Eliminar restricciones generales

Sintaxis:
DROP ASSERTION <nombre_restricción>;

Restricciones sobre una tabla base

Estas restricciones afectan a una o varias tablas. Se definen cuando se crea la tabla y podemos alterarlas o destruirlas.

RECUERDA

En el apartado 7.1.2. "Creación, modificación y borrado de tablas" ya tuvimos la oportunidad de presentar las restricciones de integridad cuando explicábamos la creación de tablas (ya que ese sería el momento de especificar dichas restricciones).

Concretamente, en el momento de crear las tablas podemos especificar los siguientes aspectos:

- **Claves primarias o candidatas:**
  - **PRIMARY KEY** se usa en el caso de que no estemos definiendo claves candidatas.
  - **UNIQUE** se utiliza para claves candidatas.

EJEMPLO 7.53

Aplicaremos restricciones en la creación la tabla *Asignaturas*.

```
CREATE TABLE Asignaturas (
Cód_As INTEGER PRIMARY KEY,
Nombre_As CHARACTER (50) NOT NULL,
Créditos_As INTEGERNOT NULL,
Facultad_As CHARACTER (50),
DNI_P CHARACTER(9));
```

- **Claves foráneas.**

  Sigue la siguiente *sintaxis*:

  ```
  FOREIGN KEY <columnas>
  REFERENCES <tabla[columna]>
  [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  ```

  Donde:
  - Mediante **ON DELETE** especificamos las acciones que queremos que se realicen cuando se borren los datos.
  - Mediante **ON UPDATE** especificamos las acciones a realizar cuando se produce una actualización de los mismos.

  Las posibilidades para ambas opciones pueden ser:

Opción	Significado
NO ACTION	No se realiza ninguna acción
CASCADE	Se borra/actualiza en cascada
SET DEFAULT	Se le asigna el valor por defecto
SET NULL	Se le asigna el valor Null

EJEMPLO 7.54

Aplicaremos restricciones en la creación la tabla *Estudian*.

```
CREATE TABLE Estudian (
DNI_Al CHARACTER (9) NOT NULL REFERENCES Alumnos(DNI_Al) ON UPDATE
CASCADE ON DELETE CASCADE,
Cód_As INTEGER NOT NULL REFERENCES Asignaturas(Cód_As) ON UPDATE
CASCADE ON DELETE CASCADE,
Nota1_Al_As INTEGER,
Nota2_Al_As INTEGER,
Convocatoria_Al_As INTEGER,
PRIMARY KEY (DNI_Al, Cód_As));
```

- Restricciones con CHECK:

  En el momento de definir crear una tabla podemos definir la restricción que necesitemos sobre una columna simplemente añadiendo la palabra reservada CHECK junto a la condición que debe cumplir.

EJEMPLO 7.55

Añadiremos, en la creación la tabla *Estudian*, una restricción para que la *Convocatoria_Al_As* sea siempre positiva.

```
CREATE TABLE Estudian (
DNI_Al CHARACTER (9) NOT NULL REFERENCES Alumnos(DNI_Al) ON UPDATE
CASCADE ON DELETE CASCADE,
Cód_As INTEGER NOT NULL REFERENCES Asignaturas(Cód_As) ON UPDATE
CASCADE ON DELETE CASCADE,
Nota1_Al_As INTEGER,
Nota2_Al_As INTEGER,
Convocatoria_Al_As INTEGER CHECK (VALUE>0),
PRIMARY KEY (DNI_Al, Cód_As));
```

NOTA

Todas las restricciones vistas en este apartado pueden ser combinadas, lo que nos permitirá tener una gran cantidad de posibilidades para definir la integridad de nuestra base de datos.

7.11.3 INSTRUCCIONES DE AUTORIZACIÓN

Cualquier usuario que quisiera realizar una operación sobre un objeto de la base de datos debe tener permiso para llevarla a cabo.

El administrador es, por defecto, el usuario que tiene asignados todos los permisos de todos los objetos de la base de datos, y él será la persona encargada de dar los permisos necesarios a todos los demás usuarios de la misma.

El administrador puede realizar dos operaciones sobre un permiso:

- **GRANT:** conceder un determinado permiso.
- **REVOKE:** denegar un permiso o revocarlo.

Concesión de autorizaciones

Sintaxis:

```
GRANT <privilegios> ON <objeto> TO <usuarios>
[WITH GRANT OPTION];
```

Donde:

- Los **<privilegios>**pueden ser:
  - **ALL PRIVILEGES:** todos los privilegios sobre el objeto especificado. Se trata del que tiene el administrador por defecto.
  - **USAGE:** utilización del objeto especificado, en este caso del dominio.
  - **SELECT:** de consulta.
  - **INSERT[<columnas>]**: de inserciones, pudiendo concretar qué columnas.
  - **UPDATE[<columnas>]:** de modificaciones, pudiendo concretar qué columnas.
  - **DELETE:** privilegio para borrar.
  - **REFERENCES[<columnas>]:** privilegio para referenciar el objeto con restricciones de integridad, pudiendo concretar qué columnas.
  - **TRIGGER:** permiso para definir *triggers*.
  - **CONNECT:** permite conectarse a la base de datos (es el primer derecho, el fundamental).
  - **SELECT:** permite seleccionar.
  - **UPDATE:** permite actualizar la base de datos.
  - **DELETE:** permite eliminar la base de datos.
  - **INDEX:** permite crear índices.
  - **RESOURCE:** permite crear objetos.
- **<objeto>** puede ser:
  - **DOMAIN:** dominio.
  - **TABLE:** tabla.
  - **VIEW:** vista.
- Los **<usuarios>** pueden ser PUBLIC (todo el mundo) o bien una lista de los identificadores de los usuarios que queramos autorizar.
- La opción **WITH GRANT OPTION** posibilita ceder su permiso (es opcional).

EJEMPLO 7.56

Añadimos el privilegio para que el usuario FRAN pueda seleccionar la tabla *Profesores*.

```
GRANT SELECT ON TABLE Profesores TO FRAN;
```

Revocación de autorizaciones

Sintaxis:

```
REVOKE [GRANT OPTION FOR] <privilegios> ON <objeto> FROM <usuarios>
[RESTRICT | CASCADE];
```

Donde:

- Los **<privilegios>**, **<objeto>** y **<usuarios>**son los que definimos en la explicación de GRANT.
- La opción opcional **GRANT OPTION FOR** se emplea si se desea eliminar el derecho a autorizar.
- Y, por último, tenemos dos opciones:
  - **CASCADE:** se utiliza para que, en caso de que un usuario autorizado, a su vez haya autorizado a otros, todos los usuarios queden desautorizados con dicha instrucción.
  - **RESTRICT:** no nos permite desautorizar a un usuario si este ha autorizado a otros.

EJEMPLO 7.57

Eliminar la autorización anterior.

```
REVOKE SELECT ON TABLE Profesores FROM FRAN CASCADE;
```

7.11.4 CONTROL DE LAS TRANSACCIONES

Una transacción es una unidad lógica de trabajo o conjunto de sentencias que se ejecutan como si fuesen una sola. En general, las sentencias que forman parte de una transacción se interrelacionan entre sí, y no tiene sentido que se ejecute una sin que se ejecuten las demás.

La mayoría de las transacciones se inician de forma implícita al utilizar alguna sentencia que empieza con CREATE, ALTER, DROP, SET, DECLARE, GRANT o REVOKE, aunque existe la sentencia SQL para iniciar transacciones, siguiendo la siguiente *sintaxis*:

```
SET TRANSACTION {READ ONLY | READ WRITE}
{COMMIT | ROLLBACK}
[WORK];
```

Donde:

- **READ ONLY:** si no queremos actualizar la base de datos.
- **READ WRITE:** si queremos actualizar la base de datos.
- **COMMIT:** confirma todos los cambios producidos contra la base de datos durante la ejecución de la transacción.
- **ROLLBACK:** deshace todos los cambios que se hayan producido en la base de datos y la deja como estaba antes del inicio de nuestra transacción.
- **WORK:** es una palabra reservada opcional que se emplea para aclarar lo que hace la sentencia.

EJEMPLO 7.58

Crear una transacción para sumar un punto a la *Nota1_Al_As* y restar un punto a la *Nota2_Al_As* de la asignatura "002".

```
SET TRANSACTION READ WRITE;
UPDATE Estudian SET Nota1_Al_As = Nota1_Al_As + 1;
UPDATE Estudian SET Nota2_Al_As = Nota2_Al_As – 1 WHERE Código_As = 002;
COMMIT;
```

7.12 EL LENGUAJE DE CONTROL DE DATOS (DCL)

El lenguaje de Control de Datos (DCL) es con el cual podremos controlar el acceso a la información de nuestra base de datos, proporcionando la seguridad de la misma, dándole integridad a nuestros datos.

Podemos resumir sus características de la siguiente forma:

- Garantiza la seguridad de nuestros datos.
- Previene el acceso ilegal a nuestros datos.
- Monitoriza el acceso a nuestros datos.
- Especifica, a cada usuario, lo que puede realizar en nuestra base de datos.

Como podemos observar, por medio de este lenguaje ofrecemos seguridad a la administración de una base de datos. Como ya hemos visto en apartados anteriores, se basan en dos comandos principalmente: *GRANT* y *REVOKE*.

7.12.1 TRANSACCIONES

El concepto de **transacciones** está ligado a dos conceptos totalmente independientes. Por un lado, está relacionado con el acceso concurrente de varios clientes a un elemento común, en este caso, la base de datos, mientras que el otro concepto con el que se relaciona es con tener un sistema persistente a fallos.

Así, comenzaremos viendo cómo funciona la estructura de los DBMS y la interacción con los clientes. En la Figura 7.1 podemos observar que los datos son almacenados en el disco, el cual se comunica con el sistema de gestión de base de datos, controlador de las interacciones con los datos. A menudo se cuenta con *software* adicional (en una capa de abstracción superior al propio DBMS), tal vez un servidor de aplicaciones o servidor web, que luego interactúa con los posibles usuarios, ya que en este nivel es donde suceden los problemas relacionados con el acceso concurrente de múltiples usuarios.

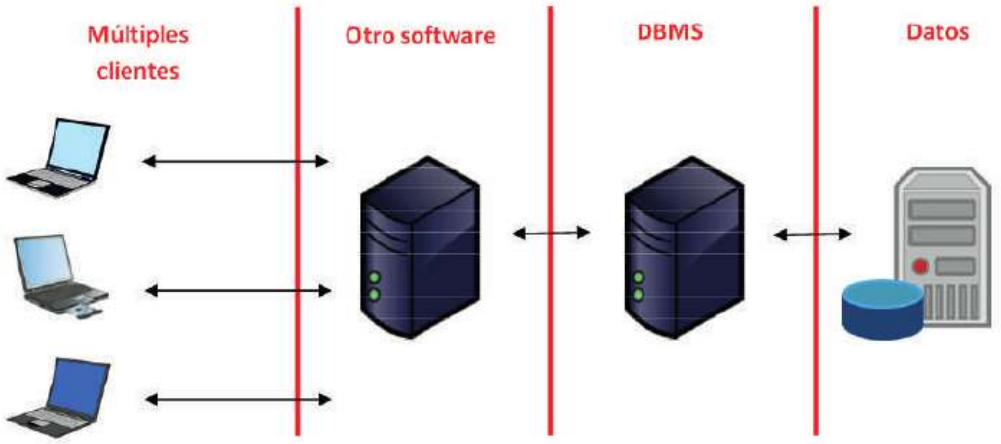


Figura 7.1. *Esquema general de una base de datos multiusuario*

En el segundo sentido, es decir, en el relacionado con la integridad de las transacciones, podemos definir una transacción como un conjunto de operaciones (u órdenes) que se ejecutan en forma indivisible (atómica) sobre una base de datos.

El DBMS debe mantener la integridad de los datos, haciendo que estas transacciones no puedan finalizar en un estado intermedio. De este modo, si, por algún motivo, se tuviera que cancelar una transacción, el DBMS comenzará a deshacer las órdenes ejecutadas hasta dejar la base de datos en su estado inicial (llamado punto de integridad), como si la orden de la transacción nunca se hubiese realizado.

7.12.2 PROPIEDADES DE LAS TRANSACCIONES: ATOMICIDAD, CONSISTENCIA, AISLAMIENTO Y PERMANENCIA

Para poder trabajar con bases de datos concurrentes, debemos conocer las principales propiedades de las transacciones (conocidas con las siglas *ACID*):

- **A → Atomicidad:** esta propiedad implica que todas las acciones de la transacción se deberán realizar o ninguna de ellas se llevará a cabo, lo cual significa que si la transacción se interrumpe, sus resultados parciales deben ser deshechos (y, por tanto, las órdenes ejecutadas anuladas).
- **C → Consistencia:** esta propiedad implica que solo los valores o datos válidos serán escritos en la base de datos. Por tanto, si una transacción no cumple la propiedad, se dejará la base de datos en su estado de consistencia anterior. Por el contrario, si la transacción se ejecuta correctamente, la base de datos pasará de su estado de consistencia anterior a un nuevo estado de consistencia.
- **I → Aislamiento:** la tercera propiedad garantiza que las transacciones no se afecten entre sí, es decir, que al realizar dos o más transacciones concurrentes sobre la misma información, dichas transacciones sean independientes y no generen ningún tipo de error.

  Esta propiedad define cómo y cuándo los cambios producidos por una operación se hacen visibles para las demás operaciones concurrentes. El aislamiento puede alcanzarse en distintos niveles, siendo el parámetro esencial a la hora de seleccionar el DBMS.

- **D → Durabilidad** (o **permanencia**)**:** es la propiedad que asegura que, una vez realizada la operación, esta persistirá y no se podrá deshacer aunque falle el sistema y que, de esta forma, los datos sobrevivan de alguna manera.

7.12.2.1 Estados de una transacción: activa, parcialmente comprometida, fallida, abortada y comprometida

Los estados por los que pasa una transacción pueden ser los siguientes:

- **Activa:** se trata del estado inicial y en el que permanece una transacción que está siendo ejecutada.
- **Parcialmente comprometida:** estado al que evoluciona tras ejecutar una transacción su última instrucción.
- **Fallida:** estado al que se llega cuando se constata que es imposible de continuar su ejecución normal.
- **Abortada:** estado al que se llega tras una transacción retrocedida y con la base de datos restaurada al estado anterior a su ejecución. Se puede reiniciar o cancelar.
- **Comprometida:** estado al que se llega tras completarse con éxito la transacción.

Veamos un diagrama de transiciones con los siguientes estados y sus posibles evoluciones:

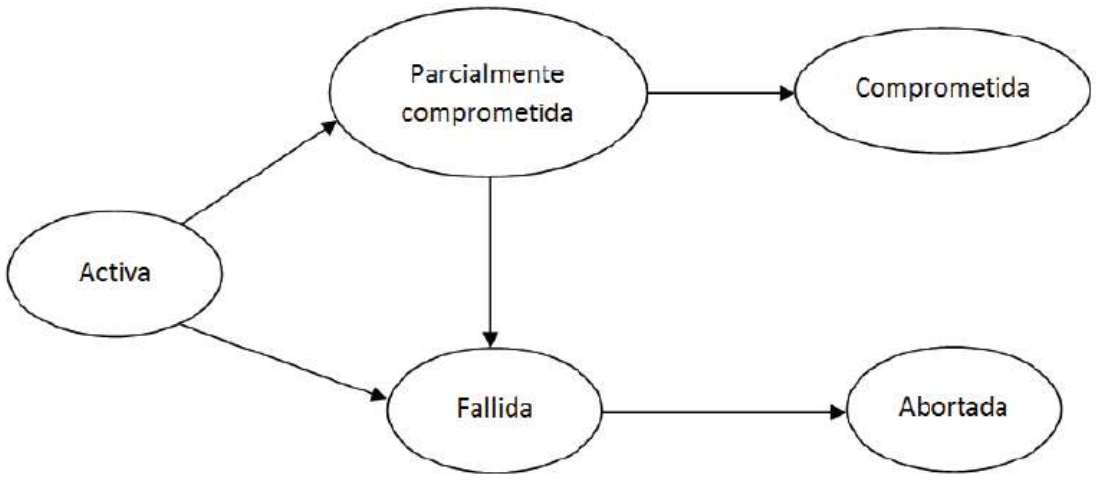


Figura 7.2. *Diagrama de transiciones con los estados de una transacción*

7.12.2.2 Consultas y almacenamiento de estructuras en XML

Almacenamiento de estructuras en XML

XML (*Extensible Markup Language*) es un estándar para el intercambio de información entre diferentes sistemas en Internet, el cual es independiente del formato de almacenamiento de dichos sistemas. Este estándar es un metalenguaje que puede ser utilizado para describir la estructura lógica y el contenido de una gran variedad de documentos, y puede ser adaptado para satisfacer una gran cantidad de aplicaciones.

NOTA

En términos comparativos, se puede decir que XML ha causado el mismo impacto que produjo la aparición del SQL.

De este modo, XML proporciona un formato cómodo para transferir el contenido de un Dataset a/desde clientes remotos, mientras que los objetos XML sincronizan y transforman los datos. Podemos ver un esquema a continuación.

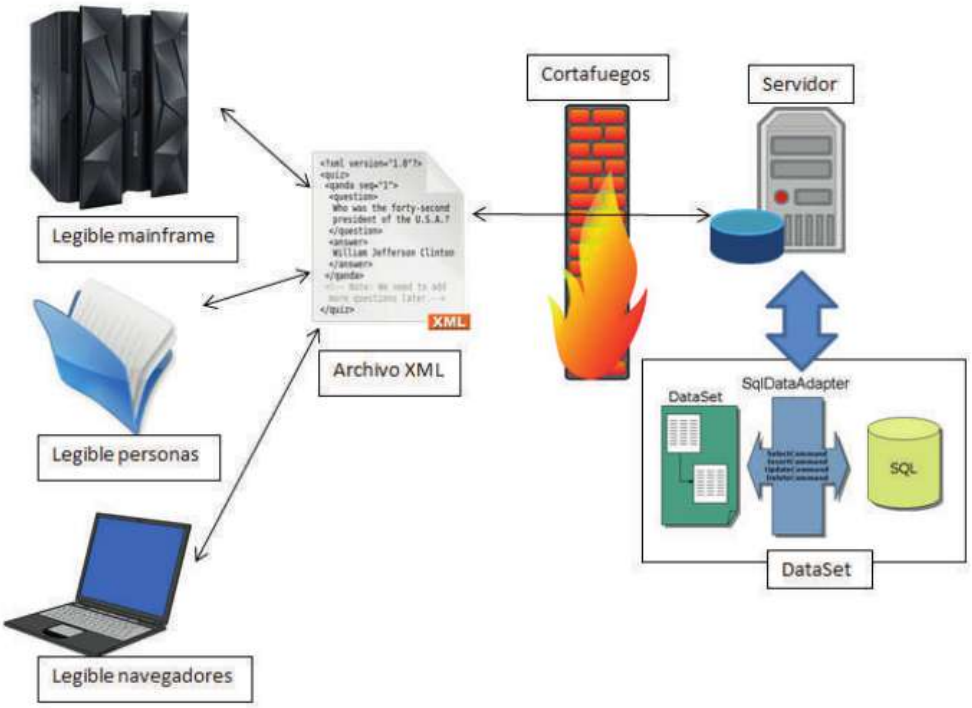


Figura 7.3. *Esquema de utilidad de XML*

En la actualidad, muchas aplicaciones requieren el almacenamiento de datos XML en bases de datos "convencionales" y, del mismo modo, en muchas ocasiones necesitamos poder transferir estas bases de datos a través de archivos XML, por lo que es de vital importancia el poder importar/exportar información entre estos dos formatos (XML y base de datos deben poder "comunicarse").

Entre los estándares de bases de datos que la literatura utiliza junto con el lenguaje XML podemos destacar:

- Bases de datos nativas.
- Bases de datos orientadas a Objetos (BDOO).
- Y, por supuesto, las que ocupan el objetivo de este libro, las bases de datos relacionales.

En este apartado, dados los objetivos del módulo profesional, trabajaremos solo con este último tipo de bases de datos y explicaremos el procedimiento para convertir manualmente una base de datos relacional en archivos XML. Lo haremos a través de un ejercicio guiado.

INFORMACIÓN

Aunque vayamos a explicar un proceso manual de conversión, debemos tener en cuenta que existe un gran número de *middlewares* encargados de la transferencia de información entre estructuras XML y bases de datos relacionales (y viceversa).

De estos, podemos destacar:
- **ADO**, de Microsoft

Allora, de HiT Software.

ASP2XML, de Stonebroom
Y, como propuesta open source, **Data Junction**, de la empresa Data Junction Inc.

EJERCICIO GUIADO 7.2

Dado un extracto del diagrama Entidad-Relación del Ejercicio guiado 2.1 pasado a tablas en el Ejercicio guiado 5.1, tenemos el siguiente diseño lógico:

Supermercado (Código_S, Domicilio_S, Superficie_S, Teléfono_S)
Empleados (NIF_E, Nombre_E, Domicilio_E, Código_S, Número_P)
Clientes (NIF_C, Nombre_C, Domicilio_C)
Pedidos (Código_S, Número_P, Fecha_P, Hora_P, Total_P, NIF_E, NIF_C)

NOTA

En este ejemplo, supondremos que un *Pedido* puede ser servido por varios *Empleados* (de ahí los cambios en color rojo).

A partir de dicho diseño, transformaremos la tabla principal *Pedidos* a un documento XML, para lo cual seguiremos las *Reglas de Transformación* y generaremos el DTD (*Declaración de Tipo de Documento*) y el Documento XML.

Empleados

NIF_E	Nombre_E	Domicilio_E

Clientes

NIF_C	Nombre_C	Domicilio_C

Pedidos

Supermercado	Número_P	Fecha_P	Hora_P	Total_P	Empleado	Cliente

Supermercado

Código_S	Domicilio_S	Superficie_S	Telefono_S

Regla1: para cada tabla en el esquema de la base de datos hay que crear un elemento con el mismo nombre de la tabla y la cardinalidad apropiada.

```
<!DOCTYPE pedidos [
    <!ELEMENT pedidos(pedido)*>
]>
```

Regla 2: las columnas de la tabla son incluidas en otro elemento (subelemento del elemento creado en la *Regla 1*), que representa un registro de dicha tabla.

```
<!ELEMENT pedido(supermercado, numero_p, fecha_p, hora_p, total_p, empleado, cliente)>
```

Regla 3: para cada columna en la tabla cuyo tipo de datos es simple (char, integer, float…) crear un elemento, subelemento del elemento creado en la *Regla 2*, de tipo *#PCDATA* con el mismo nombre de la columna.

```
<!ELEMENT numero_p (#PCDATA)>
<!ELEMENT fecha_p (#PCDATA)>
…
```

Regla 4: para cada columna en la tabla cuyo tipo de datos es complejo (tipo objeto), crear un elemento complejo, subelemento del elemento creado en la *Regla 2*, con el mismo nombre de la columna. Para cada propiedad del tipo objeto, crear un elemento con el mismo nombre de la propiedad.

```
<!ELEMENT empleados (empleado)+>
    <!ELEMENT empleado (nif_e, nombre_e, domicilio_e)>
    <!ELEMENT nif_e (#PCDATA)>
    <!ELEMENT nombre_e (#PCDATA)>
    <!ELEMENT domicilio_e (#PCDATA)>
```

Regla 5: para cada columna en la tabla que es una tabla anidada, crear un elemento con el mismo nombre de esa columna y la cardinalidad apropiada. Repetir todos los pasos desde la *Regla 2*.

DTD Resultante:

```
<!DOCTYPE pedidos[
    <!ELEMENT pedidos(pedido)*>
    <!ELEMENT pedido(supermercado, numero_p, fecha_p, hora_p, total_p, empleado, cliente)>
```

```
        <!ELEMENT supermercado (codigo_s, domicilio_s, superficie_s, telefono_s)>
                <!ELEMENT codigo_s (#PCDATA)>
                <!ELEMENT domicilio_s (#PCDATA)>
                <!ELEMENT superficie_s (#PCDATA)>
                <!ELEMENT telefono_s (#PCDATA)>
                <!ELEMENT numero_p (#PCDATA)>
                <!ELEMENT fecha_p (#PCDATA)>
                <!ELEMENT hora_p (#PCDATA)>
                <!ELEMENT total_p (#PCDATA)>
                <!ELEMENT empleados (empleado)+>
                <!ELEMENT empleado (nif_e, nombre_e, domicilio_e)>
                        <!ELEMENT nif_e (#PCDATA)>
                        <!ELEMENT nombre_e (#PCDATA)>
                        <!ELEMENT domicilio_e (#PCDATA)>
                <!ELEMENT cliente (nif_c, nombre_c, domicilio_c)>
                <!ELEMENT nif_c (#PCDATA)>
                <!ELEMENT nombre_c (#PCDATA)>
                <!ELEMENT domicilio_c (#PCDATA)>
]>
```

Ejemplo de documento XML:

```
<?xml version:"1.0">
<pedidos>
    <pedido>
        <supermercado>
            <codigo_s>001</codigo_s>
            <domicilio_s>C/ Inglés, 6</domicilio_s>
            <superficie_s>1000</superficie_s>
            <telefono_s>950000000</telefono_s>
        </supermercado>
        <numero_p>000001</numero_p>
        <fecha_p>09/01/2016</fecha_p>
        <total_p>240</total_p>
        <empleados>
            <empleado>
                <nif_e>11111111A</nif_e>
                <nombre_e>Fran</nombre_e>
                <domicilio_e>C/ Fran, 1</domicilio_e>
            </empleado>
            <empleado>
                <nif_e>22222222B</nif_e>
                <nombre_e>Amalia</nombre_e>
                <domicilio_e>C/ Amalia, 2</domicilio_e>
            </empleado>
        </empleados>
        <cliente>
```

```
                    <nif_c>33333333C</nif_e>
                    <nombre_c>Manuel</nombre_e>
                    <domicilio_c>C/ Manuel, 3</domicilio_e>
                </cliente>
        </pedido>
</pedidos>
```

NOTA

En el ejercicio guiado anterior hemos visto algunos operadores que necesitan ser explicados (además de otros que no han aparecido):

- **Operador '*':** significa ninguna ocurrencia o muchas.
- **Operador '+':** significa una ocurrencia o muchas.
- **Operador '|':** significa "o" lógico.

Consultas a datos XML

Para desarrollar este apartado hemos escogido **XQuery**, que es un lenguaje de consulta diseñado para colecciones de datos XML. La decisión se debe a que XQuery es semánticamente similar a SQL, aunque incluye algunas opciones muy interesantes de programación.

XQuery, el cual todavía se encuentra en proceso de elaboración, fue creado por el grupo Query Working Group de la W3C, en octubre de 1999. XQuery ha sido construido sobre la base de XPath, sobre el cual se fundamenta, y está diseñado para escribir consultas sobre colecciones de datos expresados en XML. Una consulta en XQuery es una expresión que lee una secuencia de datos en XML y devuelve como resultado otra secuencia de datos en XML.

XQuery añade capacidades para:

- Acceso a fuentes de información.
- Creación de documentos XML.
- Ejecución de expresiones FLOWR (*For Let Where Order by Return*), de las que más adelante hablaremos.

EJEMPLO 7.59

Dado el archivo *alumnos.xml* que almacena la información parcial de la base de datos que se utiliza para los ejemplos de este capítulo, concretamente que almacena los datos de los *Alumnos* y las asignaturas que ellos *Estudian*:

```
<?xml version:"1.0">
<alumnos>
    <alumno>
        <dni_al>44444444D</dni_al>
        <nombre_al>Mercedes</nombre_al>
        <apellido1_al>López</apellido1_al>
        <apellido2_al>Cruz</apellido2_al>
```

```
            <edad_al>20</edad_al>
            <telefono_al>950111111</telefono _al>
            <direccion_al>C/ Mercedes, 4</direccion _al>
            <ciudad_al>Almería</ciudad _al>
            <estudian>
                  <estudia>
                        <cod_as>001</cod_as>
                        <nota_al_as>2</nota_al_as>
                        <convocatoria_al_as>1</convocatoria_al_as>
                  </estudia>
                  <estudia>
                        <cod_as>002</cod_as>
                        <nota_al_as>8</nota_al_as>
                        <convocatoria_al_as>2</convocatoria_al_as>
                  </estudia>
            </estudian>
      </alumno>
      <alumno>
            <dni_al>55555555E</dni_al>
            <nombre_al>María del Mar</nombre_al>
            <apellido1_al>Macías</apellido1_al>
            <apellido2_al>Martínez</apellido2_al>
            <edad_al>16</edad_al>
            <telefono_al>950222222</telefono _al>
            <direccion_al>C/ Mar, 5</direccion _al>
            <ciudad_al>Granada</ciudad _al>
            <estudian>
                  <estudia>
                        <cod_as>002</cod_as>
                        <nota_al_as>4</nota_al_as>
                        <convocatoria_al_as>2</convocatoria_al_as>
                  </estudia>
                  <estudia>
                        <cod_as>003</cod_as>
                        <nota_al_as>10</nota_al_as>
                        <convocatoria_al_as>2</convocatoria_al_as>
                  </estudia>
            </estudian>
      </alumno>
</alumnos>
```

Consultaremos los alumnos y asignaturas aprobados:

```
for $a doc("alumnos.xml")//alumno
      where $a/nota_al_as >= 5
return
<aprobado>{
      $a/@dni_al,
```

```
        $a/@cod_as,
        $a/nota_al_as
}</aprobado>
```

El resultado será un archivo XML:

```
<aprobado dni=”44444444D” cod_as = 002>
<nota_al_as>8</nota_al_as>
<aprobado dni=”55555555E” cod_as = 003>
<nota_al_as>10</nota_al_as>
```

Lenguaje XQuery. Sintaxis básica

XQuery es un lenguaje basado en expresiones, con las siguientes expresiones básicas:

- **Números:** 6.3
- **Cadenas:** “Hello World!”
- **Constructores:** date(“2016-7-25”)
- **Operadores:** (7 + 9) / 2
- **Secuencias:** (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) o, del mismo modo (1 to 10)
- **Variables:** $variable
- **Llamada a funciones:** concat(‘Hello ‘,’World!’) →*Resultado: ‘Hello World!’*

Para acceder a los datos de entrada a partir de un elemento externo, podemos emplear las siguientes funciones del lenguaje:

- **doc(URI):** devuelve el nodo raíz del documento accesible a través de la URI.
- **Collection (URI):** devuelve una secuencia de nodos a partir de una URI.

Por otra parte, una consulta XQuery consta de dos partes bien diferenciadas:

- **Prólogo:** en esta parte incluimos los *imports*, declaraciones de funciones, espacios de nombres, variables, etc.

EJEMPLO 7.60

Ejemplo de prólogo:

```
xquery version=”1.0” encoding=”UTF-8”;
declare variable $var := expr;
declare variable $var := expr;
...
declare function name (params) as type { expr };
declare function name (params) as type { expr };
…
declare namespace prefix = “URI”;
declare namespace prefix = “URI”;
```

- **Expresiones de la consulta:** se trata de la consulta propiamente dicha. En esta sección, XQuery puede incluir las siguientes posibilidades:
  - Expresiones basadas en XPath para seleccionar nodos.

EJEMPLO 7.61

Consulta para obtener los alumnos aprobados:

```
doc("alumnos.xml")//alumno[nota_al_as >= 5]
```

  - Expresiones para la creación de nodos.

EJEMPLO 7.62

Consulta para obtener la primera nota de cada alumno del archivo XML "alumnos.xml":

```
<primeras_notas>
    <p>{doc(alumnos.xml)//alumno/nota_al_as[1]}</p>
</primeras_notas>
```

  - Crear nodos mediante constructores.

EJEMPLO 7.63

Consulta para obtener la primera nota de cada alumno del archivo XML "alumnos.xml":

```
element primeras_notas{
    element p {doc(alumnos.xml)//alumno/nota_al_as[1]}
}
```

  - Expresiones FLWOR (*For Let Where Order by Return*), que permiten:
    - **Cláusula *for*:** vincula una o más variables a expresiones escritas en XPath.
    - **Cláusula *let*:** vincula una variable al resultado completo de una expresión.
    - **Cláusula *where*:** filtra los elementos seleccionados por *for* con condiciones.
    - **Cláusula *order by*:** ordena los resultados por uno o varios criterios.
    - **Cláusula *return*:** indica los resultados que se devuelven.

EJEMPLO 7.64

Dado el archivo *alumnos.xml* que utilizamos en el Ejemplo 7.60, realizaremos las siguientes consultas a través de diferentes opciones que nos ofrece XQuery:

1. **Seleccionar los alumnos aprobados:**
   for $x in doc("alumnos.xml")//alumnos
   where $x/nota_al_as >= 5
   return $x/nombre_al
2. **Seleccionar los alumnos aprobados ordenados por nombre:**
   for $x in doc("alumnos.xml")//alumnos
   where $x/nota_al_as >= 5
   order by $x/nombre_al
   return $x/nombre_al
3. **Separar los alumnos entre aprobados y suspensos:**
   for $x in doc("alumnos.xml")//alumnos
   return if($x/nota_al_as >= 5)
   then <aprobados>{data($x/nombre_al)}</aprobados>
   else <suspensos>{data($x/nombre_al)}</suspensos>

7.12.2.3 estructura del diccionario de datos

RECUERDA

Este apartado ya lo hemos estudiado en el epígrafe 5.4 "El Diccionario de Datos: concepto y estructura".

7.12.3 CONTROL DE LAS TRANSACCIONES

RECUERDA

Este apartado ya lo hemos estudiado en el epígrafe 7.11.4 "Control de las transacciones".

7.12.4 PRIVILEGIOS: AUTORIZACIONES Y DESAUTORIZACIONES

RECUERDA

Este apartado ya lo hemos estudiado en el epígrafe 7.11.3 "Instrucciones de autorización".

7.13 PROCESAMIENTO Y OPTIMIZACIÓN DE CONSULTAS

Un sistema se considera eficiente cuando la respuesta a una consulta se realiza en el menor tiempo posible. Para ello, el sistema debe contar con un método que optimice dicha respuesta; es decir, cuando hablamos de optimización de consultas nos referimos al objetivo de encontrar la manera de devolver la respuesta a una consulta minimizando su latencia de respuesta. En DBMS relacionales la optimización de consultas es un aspecto muy significativo, ya que la complejidad que pueden alcanzar algunas consultas es muy considerable.

De hecho, las consultas a la base de datos son uno de los aspectos claves que afectan a la degradación de una aplicación en explotación (funcionamiento) y, especialmente, sobre las operaciones SELECT. La degradación de una consulta consiste en que una consulta que inicialmente ofrecía un tiempo de respuesta poco significativo, con el paso de los meses o los años, toma latencias altas en la obtención de la respuesta. Según la literatura, estos tiempos de respuestas altos suelen ser ocasionados a dos motivos fundamentales:

- **Crecimiento** importante **del número de registros en las tablas** de la base de datos, no tenido en cuanto en la fase de diseño, implementación y pruebas de la base de datos (de hecho lo más frecuente es realizar las pruebas con muy pocos registros).
- **Crecimiento del número de usuarios de la base de datos**, tampoco tenido en cuenta en las fases anteriores.

En este apartado veremos las fases fundamentales del proceso de optimización de una consulta, en qué nos basamos para llegar a dicha optimización y, finalmente, propondremos herramientas para la optimización de consultas sobre una base de datos.

Aunque, al igual que se ha hecho a lo largo del libro, intentaremos dar una visión general de las bases de datos relacionales (sin hablar de ningún DBMS en particular), el proceso de optimización y las herramientas para dicho proceso están muy relacionados con los DBMS. Así pues, citaremos algunos sistemas gestores de bases de datos concretos, en diferentes momentos, especialmente MySQL y, en ocasiones, también Oracle.

7.13.1 PROCESAMIENTO DE UNA CONSULTA

Cuando hablamos de las fases de procesamiento de una consulta, aunque este aspecto varía mucho según los autores, hablamos de tres grandes procesos:

NOTA

En DBMS como MySQL u Oracle a estas fases le precede una de caché.
Estos sistemas registran las consultas de tipo SELECT y sus resultados, ya que las consultas repetidas son muy frecuentes. Naturalmente, las consultas de modificación de datos (Insert, Delete…) invalidan las consultas afectadas de la caché, por lo que son eliminadas inmediatamente ante estas modificaciones.

1. **Análisis de la consulta:** en esta fase se obtiene una representación arbórea de la estructura de la consulta. Concretamente, las subtareas implicadas en esta fase son:
   - Generación del árbol de la consulta, realizando las comprobaciones léxicas y sintácticas de la sentencia SQL.
   - Preprocesado, es decir, llevar a cabo las comprobaciones semánticas, tales como si las tablas existen en la base de datos, si los atributos están definidos para las tablas o si los tipos de los atributos usados en las condiciones son compatibles.

NOTA

MySQL trabaja con los siguientes tipos de índices:
- B-Tree
- Hash
- R-Tree
- Textuales

2. **Reescritura de la consulta (Plan lógico de la consulta):** en esta fase transformaremos el árbol a álgebra relacional y aplicaremos sucesivas reglas de optimización algebraicas para conseguir optimizar la consulta.

   Veamos un extracto de algunas reglas básicas de transformación.

   Transformación de la SELECCIÓN

   - Ejecutar en primer lugar las selecciones, ya que así disminuimos el número de tuplas sobre las que realizaremos el resto de la consulta.

     Es decir, transformar esta consulta:

     $$\sigma_{Predicado1}\ (\sigma_{Predicado2}\ (Relación1 * Relación2)_{CondiciónProductoCartesiano}))$$

     A esta otra expresión:

     $$\sigma_{Predicado1}(Relación1 * \sigma_{Predicado2}(Relación2)_{CondiciónProductoCartesiano}))$$

   - Representar las expresiones complejas que utilizan operadores, desglosándolos en otros más sencillos.

     Es decir, transformar esta expresión:

     $$\sigma_{Predicado1\ \wedge\ Predicado2}\ (expresión)$$

     A esta otra:

     $$\sigma_{Predicado1}\ (\sigma_{Predicado2}\ (expresión))$$

   Transformación de la PROYECCIÓN

   Es aconsejable ejecutar las proyecciones lo antes posible en una consulta. Es decir, ante una consulta con la siguiente estructura:

   $$\pi_{Atributos}((\sigma_{Predicado}(Relación1) * Relación2)_{CondiciónProductoCartesiano}))$$

Se debería representar con la siguiente estructura alternativa:

$$\pi_{Atributos}((\pi_{Atributos}(\sigma_{Predicado}(Relación1))) * Relación2)_{CondiciónProductoCartesiano}))$$

Operaciones equivalentes

Debido a que se pueden aplicar las propiedades asociativas y conmutativas a la hora de combinar las relaciones, se podrían definir las siguientes equivalencias:

Expresión	Expresión equivalente
$\sigma_{Predicado}$ (Relación1 U Relación2)	$\sigma_{Predicado}$ (Relación1) U $\sigma_{Predicado}$ (Relación2)
$\sigma_{Predicado}$ (Relación1 - Relación2)	$\sigma_{Predicado}$ (Relación1) - $\sigma_{Predicado}$ (Relación2)
(Relación1 U Relación2) U Relación3	Relación1 U (Relación2 U Relación3)
Relación1 U Relación2	Relación2 U Relación1

NOTA

En la *Guía de Oracle* podemos ver un interesante *post* con más información sobre cómo optimizar una consulta SQL e información complementaria sobre este apartado que estamos estudiando:
http://epnbdd-oracle.blogspot.com.es/2012/05/optimizacion-basica-de-sql.html

3. **Generación de un plan físico de ejecución:** en último lugar, transformaremos cada operación lógica de la fase anterior, a un algoritmo de recuperación concreto. El planificador es el punto crítico en la resolución de una consulta, ya que del plan de ejecución que elabore dependerá que la consulta tarde unas pocas décimas de segundo o varios minutos.

   El planificador debe decidir dos aspectos principales en el plan de ejecución: qué índice se va a utilizar para resolver la consulta y en qué orden se realizarán los *joins* de las tablas.

Cada consulta puede tener diversos planes de ejecución, con su coste asociado. Por ello, el DBMS debe decidir cuál es el plan óptimo, mediante diferentes criterios de optimización, entre los cuales destacan, como veremos en el apartado siguiente, el **basado en reglas** y el **basado en costes**.

7.13.2 TIPOS DE OPTIMIZACIÓN: BASADA EN REGLAS, BASADAS EN COSTES, OTROS

El **optimizador basado en reglas** se basa en ciertas reglas para realizar las consultas preestablecidas en el DBMS. Este tipo de optimización no tiene en cuenta el estado actual de la base de datos, ni el número de usuarios conectados, ni la carga de datos, etc., sino que es un sistema de optimización estático.

Por su parte, el **optimizador basado en costes** se basa también en reglas básicas, pero teniendo en cuenta el estado actual de la base de datos: cantidad de memoria disponible, entradas/salidas, estado de la red, etc.

7.13.3 HERRAMIENTAS DE LA BBDD PARA LA OPTIMIZACIÓN DE CONSULTAS

A continuación vamos a presentar algunas actuaciones y comandos propios de los DBMS que nos permiten llevar a cabo la optimización de consultas, siendo estos propios de cada DBMS. Por ello, haremos un especial hincapié en el sistema MySQL, aunque también citaremos algún equivalente para Oracle. No obstante, los comandos y su sintaxis son similares en la mayoría de los DBMS.

En este sentido, clasificaremos las actuaciones y comandos en tres tipos:

- **Detección,** explicando una actuación que nos permite observar cuándo una consulta muestra altas latencias de respuesta, en cuyo caso podemos advertir posibles degradaciones de rendimiento en nuestras consultas.
- **Vista del plan de ejecución**, para lo cual hablaremos de la sentencia *EXPLAIN*.
- **Mantenimiento de las estadísticas de la base de datos**, para que el planificador tenga datos actualizados a la hora de poder elegir el correcto plan de ejecución (eligiendo los índices correctos, por ejemplo). Para ello, se emplearán los comandos *ANALYZE TABLE* y *OPTIMIZE TABLE*.

Detección

Es obvio que para arreglar el problema de la degradación de rendimiento en consultas el primer paso consiste en detectar que la base de datos se está convirtiendo en un cuello de botella. Para ello, MySQL tiene la opción de activar un *logging* para este tipo de consultas, de modo que podamos hacer *debug* e identificar las consultas lentas (*Slow Queries*) y, de esta forma, poder optimizarlas.

Activar el *log* de consultas lentas en MySQL supondrá que en el momento en que una consulta SQL supere el tiempo de ejecución definido a través de la variable *long_query_time,* esta quedará registrada en el *log* establecido.

El primer paso es activar el *logging*, para ello, primero revisamos si está activado:

```
# mysqladmin var |grep log_slow_queries
| log_slow_queries          | OFF                  |
```

Si se encuentra desactivado, para activar dicho *log* nos bastará con abrir el fichero de configuración (denominado *my.cnf*) y realizar la siguiente modificación en la sección *mysqld:*

```
log-slow-queries=/var/log/mysql-slow-queries.log
long_query_time = 5
log-queries-not-using-indexes
```

Donde:

- Con la primera directiva estamos indicando que en el fichero *log-slow-queries* vamos a guardar el *log* de las consultas lentas.
- Con la segunda directiva, asignamos a la variable *long_query_time* el tiempo (en segundos) a partir del cual se considera lenta una consulta. En este caso, cualquier consulta que tarde más de 5 segundos en ejecutarse se considerará lenta y será registrada en el *log*.
- Y, por último, mediante la última directiva, activaremos *log-queries-not-using-indexes* si queremos "loggear" las consultas que no utilicen índices para su resolución (esta opción es útil porque estas consultas suelen ser las más lentas y menos optimizadas).

Seguidamente, crearemos el fichero del log y le asignamos el propietario correspondiente:

```
touch /var/log/mysql-slow-queries.log
chown mysql.root /var/log/mysql-slow-queries.log
```

Para validar estos cambios, tendremos que reiniciar el servicio MySQL:

```
/etc/init.d/mysql restart
```

INFORMACIÓN

Para ampliar este contenido, podemos revisar los principales *logs* de MySQL en el manual oficial de este sistema de gestión de bases de datos:
http://dev.mysql.com/doc/refman/5.7/en/server-logs.html

EJEMPLO 7.65

A continuación, se muestra un ejemplo de detección de una consulta calificada como lenta (ya que su tiempo de ejecución ha excedido los cinco segundos):

```
# Time: 260816 18:48:58
# User@Host: XXX_phpbb1[XXX_phpbb1] @ localhost []
# Query_time: 15  Lock_time: 0  Rows_sent: 10  Rows_examined: 12356
SELECT t.forum_id, t.topic_id, p.post_time
```

Analizaremos los resultados obtenidos:

- Time: hora en la que se ha ejecutado la consulta.
- User@Host: usuario y *Host* desde el que se ha ejecutado la consulta, local o remoto.
- Query_time: tiempo que ha tardado la consulta en ejecutarse (15 segundos).
- Lock_time: tiempo de bloqueo que se ha necesitado para que la consulta se ejecute (0 segundos).
- Rows_sent: registros enviados por la consulta SQL (10 registros).
- Rows_examined: cantidad de registros examinados por la consulta (12356 registros).
- Finalmente, se muestra la consulta SQL calificada como lenta.

Vista del plan de ejecución

Se trata de la más potente herramienta que MySQL nos propone para orientarnos sobre cómo está ejecutando una consulta el motor de la base de datos, ya que nos muestra información muy relevante cuando creamos consultas tales como qué índices va a utilizar, de qué tipo son, cómo de efectiva será la respuesta al usar esos índices, etc.

De este modo, con los resultados del comando EXPLAIN podremos observar si la consulta está o no bien estructurada, si necesitamos el uso de algún índice, dónde utilizarlo, entre otra información relevante.

La *sintaxis* para uso es muy sencilla, basándose solo en la siguiente estructura:

EXPLAIN <*consulta_en_SQL*>

Así pues, lo más interesante de analizar es la sintaxis de los resultados que nos muestra. Dichos resultados guardan la siguiente estructura:

- **id:** identificador de la tabla en la consulta. Se le asignará un identificador distinto a cada tabla que interviene en dicha consulta.
- **select_type:** indica el tipo de SELECT que interviene en la consulta. Las opciones pueden ser:
  - **SIMPLE:** SELECT simple, sin UNION o subconsultas.
  - **PRIMARY:** se refiere a la consulta "más externa" a la principal.
  - **UNION:** resto de consulta/s (segunda, tercera, etc.) en una UNION.
  - **DEPENDENT UNION:** resto de consulta/s (segunda, tercera, etc.) en una UNION que depende una consulta "externa" PRIMARY.
  - **UNION RESULT:** resultado de una UNION.
  - **SUBQUERY:** primera consulta de una subconsulta.
  - **DEPENDENT SUBQUERY:** primera consulta de una subconsulta de la que depende una consulta "externa".
  - **DERIVED:** tabla derivada en una SELECT. Subquery en la cláusula FROM.
  - **UNCACHEABLE SUBQUERY:** subconsulta que no puede ser almacenada en caché, o revaluada.
  - **UNCACHEABLE UNION:** resto de consulta/s (segunda, tercera, etc.) en una UNION que no puede ser almacenada en caché, o re-evaluada.
- **table:** nombre de la tabla a la que se refiere la fila del resultado.
- **type:** el tipo de unión que realizará MySQL según la relación entre datos de ambas tablas. Estos pueden ser:
  - **system:** la tabla únicamente tiene una fila y es una tabla del sistema.
  - **const:** en la tabla coincide una única fila que el optimizador toma como constante.
  - **Eq_ref:** una fila de una tabla es leída por cada combinación de filas de una segunda tabla. Se utiliza cuando se usan todas las partes de una clave primaria o un índice único.
  - **ref:** todas las filas que coincidan con los valores de un índice serán leídos por cada combinación de filas de las tablas anteriores. En JOIN se utiliza cuando no se puede seleccionar una única fila en base al valor de la clave.
  - **fulltext:** el JOIN se realiza con un índice FULLTEXT.
  - **ref** or **null:** el tipo de JOIN es como *ref* pero hay una búsqueda extra para campos nulos. Este tipo se suele utilizar para resolver subconsultas.
  - **index_merge:** se utiliza la optimización *index merge*.
  - **unique_subquery:** sustituye a *ref* en algunas subconsultas (de tipo SELECT clave_primaria) que utilizan *IN*.
  - **index_subquery:** sustituye a *ref* en algunas subconsultas (de tipo SELECT valor_columna) que utilizan *IN* y que no son únicas.

- **range:** cuando se devuelve filas dentro de un rango.
- **index:** igual que un *full scan* pero se escanea el índice.
- **ALL:** *full scan*. Normalmente se debe evitar.

- **posible_keys:** la lista de índices candidatos que MySQL ha encontrado para la consulta.
- **key:** el índice elegido para resolver la consulta.
- **key_len:** tamaño del índice, en bytes.
- **ref:** las columnas o valores con los que se filtrará el índice.
- **rows:** el número estimado de filas que MySQL cree que tendrá que examinar para resolverla consulta.
- **Extra:** información adicional

INFORMACIÓN

Para ver un mayor número de ejemplos sobre el uso de EXPLAIN podemos visitar la siguiente dirección (además, la hemos utilizado para la explicación de la respuesta del comando, la cual acabamos de estudiar):
http://ridersofthebit.net/tag/explain/

Para completar esta explicación, estudiaremos dos ejemplos con diferentes grados de complejidad.

EJEMPLO 7.66

El resultado de la consulta EXPLAIN que, a continuación, se muestra:

EXPLAIN SELECT * FROM Alumnos WHERE DNI_Al='11111111A'\G

Es la siguiente:

*********************************** 1. row ***********************************
id:1
select_type: SIMPLE
table: Alumnos
type: const
possible_keys: PRIMARY
key: PRIMARY
key_len: 36
ref:const
rows: 1
Extra:
1 row in set (0.00 sec)

Analizaremos los resultados obtenidos:

- select_type: la consulta es un SELECT SIMPLE, es decir no tiene SELECT anidados, uniones, etc.
- type: el valor *const* indica que la tabla tiene, como máximo, un resultado para la consulta.

  Por tanto, el planificador puede considerar la columna como una constante para el resto de su evaluación.
- possible_keys: el único índice posible es la clave primaria (PRIMARY).
- key: el índice elegido es la clave primaria (PRIMARY).
- key_len: el tamaño de cada entrada del índice elegido es 36 bytes.
- ref: en la comparación con el índice, estamos utilizando la constante '*11111111A*'.
- rows: el número de registros que se estiman como salida será de 1.

INFORMACIÓN

En este enlace podemos ver un proceso de optimización, gracias a la herramienta EXPLAIN muy ilustrativo:
http://www.dataprix.com/422-sentencia-explain

EJEMPLO 7.67

El resultado de la consulta EXPLAIN que, a continuación, se muestra:

EXPLAIN SELECT * FROM Alumnos WHERE Edad_Al= (SELECT MAX(Edad_Al) FROM Alumnos)

Es la siguiente:

*********** 1. row ***********

id:1
select_type: PRIMARY
table: Alumnos
type: const
possible_keys: PRIMARY
key: PRIMARY
key_len: 36
ref:const
rows: 1
Extra:

*********************************** *2. row* ***

id:2
select_type: SUBQUERY
table: NULL
type: NULL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: NULL
Extra: Select tables optimized way
2 rows in set (0.00 sec)

Analizaremos los resultados obtenidos.

Podemos observar que el plan contiene dos consultas:

- La consulta con *id=1*, tiene valores y significados iguales al Ejemplo 7.67.
- La consulta con *id=2*, tiene la mayor parte de sus valores a *NULL*, ya que no es necesario utilizar ninguna tabla ni índices, ya que la subconsulta devuelve un único valor, dado por el operador *MAX*.

  En el campo **Extra**, MySQL indica que posee la información necesaria almacenada en memoria.

Mantenimiento de las estadísticas de la base de datos

Para lograr que el planificador tenga estadísticas actualizadas al elegir el índice, MySQL (y otros DBMS) emplean los comandos ANALYZE TABLE y OPTIMIZE TABLE.

ANALYZE TABLE

Este comando analiza y almacena la distribución de claves de una tabla. Durante el análisis, la tabla se bloquea con un bloqueo de lectura.

La *sintaxis* del comando es la siguiente:

*ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE <nombre_tabla1>(, Nombre_Tabla2)**

El comando retorna una tabla con las siguientes columnas:

- **Tabla:** nombre de la tabla implicada.
- **Op:** en este caso, devolvería el valor *analyze.*
- **Tipo_mensaje**. Cuyos valores pueden ser:
  - **status**
  - **error**
  - **info**
  - **warning**
- **Texto_mensaje:** mensaje de salida.

OPTIMIZE TABLE

Este comando se usa en caso de borrados de una gran parte de una tabla o si se han hecho cambios en una tabla con registros de longitud variable. Los registros borrados se mantienen en una lista enlazada y las operaciones de inserción posteriores no usan posiciones de registros anteriormente borrados.

También se puede usar este comando para otras operaciones como desfragmentar el fichero de datos.

La *sintaxis* del comando es la siguiente:

*OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE <nombre_tabla1>(, Nombre_Tabla2)**

INFORMACIÓN

Una explicación más detallada de estos comandos, junto a otros comandos relacionados con el mantenimiento de las bases de datos, los podemos encontrar en el manual de MySQL:
http://ftp.nchu.edu.tw/MySQL/doc/refman/5.0/es/table-maintenance-sql.html

7.14 EJERCICIOS RESUELTOS

1. El ejercicio consiste en escribir en SQL todas las operaciones y consultas que a continuación se relacionan:

- **Crear las tablas *Alumnos*, *Asignaturas* y *Estudian* con los atributos siguientes:**
  Alumnos (NIF, Nombre, Apellidos, Edad, Teléfono, Ciudad)
  Asignaturas (Cod, Nombre, Núm_Créditos, Profesor, Área)
  Estudian (NIF_Alumno, Cód_Asignatura, Nota)

```
CREATE TABLE Alumnos(
NIF_A VARCHAR(9),
Nombre_A VARCHAR(50),
Apellidos_A VARCHAR(50),
Edad_A INT(3),
Telefono_A INT(9),
Ciudad VARCHAR(50),
PRIMARY KEY(NIF_A));
CREATE TABLE Asignaturas(
Cod_As INT(3),
Nombre_As VARCHAR(50),
Num_Creditos_As INT(3),
Profesor_As VARCHAR(50),
Area_As VARCHAR(30),
PRIMARY KEY(Cod_As));
CREATE TABLE Estudian(
NIF_A VARCHAR(9) REFERENCES Alumnos(NIF_A) ON UPDATE CASCADE ON DELETE CASCADE
```

```
Cod_As INT(3) REFERENCES Alumnos(NIF_A) ON UPDATE CASCADE ON DELETE CASCADE
Nota_A_As INT(3),
PRIMARY KEY (NIF_A, Cod_As));
```

- **Añadir a la tabla *Estudian* la columna *Num_Convocatoria* de tipo entero.**

```
ALTER TABLE Estudian ADD Num_Convocatoria INTEGER(1);
```

- **Insertar cinco alumnos (de los cuales algunos tengan la misma edad), cinco asignaturas (que compartan profesores y donde haya un área de programación) y diez relaciones en la tabla *Estudian*.**

```
INSERT INTO Asignaturas VALUES(1, "Bases de Datos", 6, "Fran", "Programación"),(2, "Redes", 4.5, "Amalia",
"Arquitectura ordenadores"),(3, "Matemáticas Discreta", 9, "Fran", "Matemáticas"),(4, "Sistemas Operativos", 6,
"Alex", "Programación"),(5, "EDA", 7.5, "Fran", "Programación");

INSERT INTO Alumnos VALUES("11111111A", "Carlos", "Martínez Martínez", 30, "950111111"), ("22222222B",
"Laura", "López López", 26, "950222222"), ("33333333C", "Manuel", "Gallegos Gallegos", 23, "950333333"),
("44444444D", "Leonor", "Capel Capel", 30, "950444444"), ("55555555E", "Pepe", "Giménez Jiménez", 30,
"950555555");

INSERT INTO Estudian VALUES("11111111A", 1, 10, 1), ("11111111A", 3, 5.6, 3), ("22222222B", 2, 6, 3),
("22222222B", 1, 8.4, 2), ("33333333C", 5, 7, 1), ("44444444D", 4, 9, 2), ("44444444D", 5, 5.6, 5), ("44444444D", 1,
7, 2), ("55555555E", 3, 6, 2), ("55555555E", 4, 8.9, 1);
```

- **Subir un punto a todos los *Alumnos* de 18 años en *Asignaturas* del *Área_As* de "Programación".**

```
UPDATE Estudian SET Nota_A_As = Nota_A_As + 1
WHERE NIF_A IN (SELECT NIF_A FROM Alumnos WHERE Edad_A = 18)
AND Cod_As IN (SELECT Cod_As FROM Asignaturas WHERE Area = "Programación")
AND Nota < 10;
```

- **Eliminar los *Alumnos* cuyo *Nombre_A* contengan la 'r'.**

```
DELETE FROM Alumnos WHERE Nombre_A LIKE "%r%" OR Nombre_A LIKE "%R%";
```

- **Mostrar los *Alumnos* matriculados en cada *Asignatura*.**

```
SELECT Nombre_As, A.NIF_A, Nombre_A
FROM Alumnos A, Estudian E, Asignaturas S
WHERE A.NIF_A = E.NIF_A AND E.Cod_As = S.Cod_As
GROUP BY Nombre_As;
```

- **Mostrar los *Alumnos* matriculados en cada *Asignatura* pero mostrando solo las *Asignaturas* que tengan más de un *Alumno*.**

```
SELECT Nombre_As, A.NIF_A, Nombre_A
FROM Alumnos A, Estudian E, Asignaturas S
WHERE A.NIF_A = E.NIF_A AND E.Cod_As = S.Cod_As
GROUP BY Nombre_As
HAVING COUNT(*)>1;
```

- **Mostrar el nombre de las *Asignaturas* con mayor número de créditos.**

```
SELECT Nombre_As
FROM Asignaturas
WHERE Num_Creditos_As = (SELECT MAX(Num_Creditos_As) FROM Asignaturas);
```

- **Mostrar la *Nota_A_As* media de cada *Alumno*.**

```
SELECT A.NIF_A, Nombre_A, AVG(Nota_A_As) AS "Nota Media"
FROM Alumnos A, Estudian E
WHERE A.NIF_A = E.NIF_A
GROUP BY NIF_A;
```

- **Mostrar el número de convocatoria en la que aprobaron los alumnos que viven en "Granada".**

```
SELECT A.NIF_A, Nombre_A, Nombre_As, Num_Convocatoria
FROM Alumnos A, Estudian E, Asignaturas S
WHERE A.NIF_A = E.NIF_A AND E.Cod_As = S.Cod_As AND Ciudad_A = "Granada"
AND Nota_A_AS>=5;
```

- **Mostrar el *Nombre_A* de los *Alumnos* por grupo de *Edad_A*.**

```
SELECT Edad_A, Nombre_A
FROM Alumnos
GROUP BY Edad_A;
```

- **Mostrar los alumnos que no están matriculados en ninguna asignatura.**

```
SELECT NIF_A, Nombre_A
FROM Alumnos
WHERE NIF_A NOT IN (SELECT NIF_A FROM Alumnos A, Estudian E
WHERE A.NIF_A = E.NIF_A);
```

- Mostrar los alumnos que están matriculados en más de dos asignaturas.

```
SELECT A.NIF_A, Nombre_A
FROM Alumnos A, Estudian E
WHERE A.NIF_A = E.NIF_A
GROUP BY Nombre_A
HAVING COUNT(*) > 2;
```

7.15 EJERCICIOS PROPUESTOS

1. El ejercicio consiste en escribir en SQL todas las operaciones y consultas que a continuación se relacionan:

- Crear las tablas *Coche*, *Garaje*, *Cliente*, *Agencia* y *Reserva* con los atributos siguientes:
  Coche (Matrícula_C, Marca_C, Modelo_C, Código_G, precioAlquiler_C)
  Garaje (Código_G, Nombre_G, Dirección_G)
  Cliente (DNI_Cli, Nombre_Cli, Dirección_Cli, Ciudad_Cli, Teléfono_Cli)
  Reserva (Número_R, FechaInicio_R, FechaFin_R, DNI_Cli, precioTotal_R, deVuelta_R)
  Lista_Reserva (Número_R, Matrícula_C, LitrosInicio, precioFinal)
- Insertar diversos campos a cada una de las tablas creadas.
- Eliminar los clientes de "Almería".
- Mostrar el número, fecha de inicio y de fin, coche, marca y modelo de las reservas de coches del garaje con código *G1* realizadas a través de la agencia con el código número 4.

- Código y nombre de garajes donde nunca se ha reservado ningún coche.
- NIF y nombre de los clientes que hayan reservado algún coche de la marca "BMW" más de dos veces.
- Coche que más veces ha sido reservado, indicando cuántas veces ha sido reservado.
- Suma total del precio de las reservas realizadas por clientes que hayan hecho menos de tres reservas.
- Reservas con un solo coche en la lista de reservas.

7.16 TEST DE CONOCIMIENTOS

NOTA

Las consulta 1 - 6 pertenecen a un ejercicio de las **Oposiciones al Cuerpo de Técnicos Auxiliares de Informática de la Administración del Estado (Convocatoria de 2015)**.

1 Señales las propiedades que han de tener las transacciones de los sistemas gestores de bases de datos:
a) Atomicidad, consistencia, cohesión y permanencia
b) Atomicidad, consistencia, polimorfismo y permanencia
c) Atomicidad, consistencia, aislamiento y permanencia
d) Atomicidad, encapsulación, herencia y permanencia

2 En una base de datos relacional se quiere añadir una nueva columna ya existente. ¿Qué sentencia SQL habría que utilizar?
a) ADD COLUMN
b) ALTER TABLE
c) INSERT COLUMN
d) MODIFY TABLE

3 Indica qué resultado se puede esperar de esta consulta SQL sobre una tabla Coches_Venta que mantiene el inventario de automóviles en un pequeño concesionario:

```
SELECT Modelo, Color, COUNT(Bastidor) AS Num
FROM Coches_Venta
GROUP BY Modelo, Color
HAVING COUNT(Bastidor)<=1
```

a) Le indica al dueño del concesionario que quizás debe plantearse adquirir más existencias de un cierto modelo y color de automóvil.
b) La indica al dueño del concesionario todos los modelos distintos del concesionario. Es decir, un inventario organizado por modelos.
c) Le indica al dueño del concesionario todos los modelos distintos del concesionario. Es decir, un inventario organizado por modelos y color.
d) Le indica al dueño todos los modelos cuyo número de Bastidor es menor o igual a 1.

4 ¿Cuál de los siguientes términos se relaciona directamente con el control de integridad en SQL?
a) ROLLBACK
b) GRANT
c) CREATE INDEX
d) REVOKE

5 ¿Qué sentencia de las siguientes pertenece a la categoría de DDL?
a) CREATE
b) UPDATE
c) SELECT
d) DELETE

6 ¿Qué operador de SQL, utilizado junto la cláusula WHERE, permite buscar según un patrón determinado en una columna?
a) LIKE
b) UPDATE
c) BETWEEN
d) HAVING

NOTA

Las consultas 7 y 8 pertenecen a un ejercicio de las **Oposiciones al Cuerpo de Gestión de Sistemas e Informática de la Administración del Estado (Convocatoria de 2015)**.

7 Señale cuál de las siguientes sentencias se cataloga dentro del lenguaje SQL como sentencia de DCL:
a) REVOKE
b) CREATE
c) DELETE
d) SELECT

8 En el ámbito de las bases de datos, ANALYZE:
a) Recopila estadísticas acerca de los objetos del esquema usados por el optimizador.
b) Es una herramienta de la base de datos con la que se realiza el análisis del modelo de datos.
c) Es una herramienta de la base de datos con la que se realiza el diseño del modelo de datos.
d) Es una vista ordinaria.

9 ¿Cuál no es un modificador de tipo en SQL?
a) NULL
b) UNIQUE
c) PRIMARY
d) NOT NULL

10 La instrucción ALTER TABLE utiliza los siguientes comandos:
a) ADD, MODIFY, DELETE
b) ADD, MODIFY, DROP
c) ADD, CHANGE, DELETE
d) ADD, UPDATE, DELETE

11 El operador LIKE utiliza los siguientes caracteres especiales:
a) '%' (cualquier subcadena) y '_' (un carácter)
b) '%' (cualquier subcadena), '_' (un carácter) y '\' (escape)
c) '%' (cualquier subcadena) y '#' (un carácter)
d) '%' (cualquier subcadena), '#' (un carácter) y '\' (escape)

12 Dados los datos introducidos en la siguientes tablas:
ALUMNOS (11111111A – Fran; 22222222B – Amalia; 33333333C – Juan; 44444444D – Antonio)
ESTUDIAN (11111111A – 7 – …; 22222222B – 8 – …; 33333333C – 8 – …; 44444444D – 5 – …)
La consulta "SELECT (*) FROM Alumnos, Estudian WHERE Alumnos.DNI_A = Estudian.DNI_A AND Nota_A_As > ALL (SELECT Nota_A_As FROM Estudian)" devolverá:
a) Amalia
b) Amalia y Juan
c) Todos los alumnos
d) Ningún alumno

13 Los privilegios que se pueden otorgan con el comando GRANT son:
a) ALL PRIVILEGES, USAGE, SELECT, INSERT
b) DELETE, TRIGGER, CONNECT, SELECT, DISCONNECT
c) UPDATE, INDEX, RESOURCE, UNCONNECT
d) Todas las respuestas son ciertas.

14 Los estados de una transacción pueden ser:
a) Activa, error, pausa, inactiva
b) Activa, parcialmente comprometida, fallida, abortada, comprometida
c) Activa, parcialmente comprometida, abortada, comprometida
d) Activa, fallida, abortada, comprometida

15 Para la detección de consultas lentas en MySQL debemos:
a) Activar el log *long-query-time* en el fichero *my.conf*.
b) Activar el log de consultas lentas, activando el fichero *long-query-time.log*.
c) Activar el log *long-query-time* en el fichero *mysql.conf*.
d) Activar el log *long-query-time* en el fichero *mysql-slow-queries.log*.

8 Lenguajes de programación de bases de datos

8.1 ENTORNOS DE DESARROLLO

8.1.1 QUÉ ES UN ENTORNO DE DESARROLLO

Un **entorno de desarrollo integrado** (en inglés *Integrated Developmen Environment*, IDE) lo componen un conjunto de diversas herramientas de programación, entre las que se encuentran, al menos, un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI); todas ellas empaquetadas como un programa de aplicación.

De esta forma, podemos afirmar que el IDE es la herramienta fundamental de trabajo para un programador, facilitándose así su labor y siendo para el programador, lo que un lienzo y pinturas son para un artista. De hecho, un entorno de desarrollo eficiente e inteligente, puede llevar a cabo una importante labor facilitadora en el trabajo de creación de un *software*.

NOTA

Los IDE pueden ser aplicaciones por sí solas o pueden estar incluidas en otras aplicaciones existentes.

Por citar un ejemplo de este último caso, el lenguaje Visual Basic es usado en las herramientas del paquete ofimático Microsoft Office, lo que permite escribir procedimientos y funciones en Visual Basic para la creación de macros.

Las características más reiteradas por la mayor parte de los IDE se pueden resumir en las siguientes:

- Son aplicaciones **multiplataforma,** es decir, tienen versiones para los diferentes entornos y sistemas operativos, o bien se trata de una única versión que puede operar en estas.
- **Ofrecen soporte para más de un lenguaje de programación**, como veremos en el apartado 8.1.3.
- Suelen tener integrado un **sistema de control de versiones**.

NOTA

Según Wikipedia "Los **programas para control de versiones** son un grupo de aplicaciones originalmente ideadas para gestionar ágilmente los cambios en el código fuente de los programas y poder revertirlos, cuyo ámbito ha sido ampliado pasando del concepto control de versiones al de gestión de configuración de software, en el que se engloban todas las actividades que pueden realizarse por un equipo sobre un gran proyecto software u otra actividad que genere ficheros digitales (…)"

WIKIPEDIA
The Free Encyclopedia

- Poseen **Reconocimiento de Sintaxis**, marcándote con colores o formatos distintos las palabras reservadas del lenguaje en el que se programe, incluyendo opciones de autocompletado léxico o, incluso, sintáctico, etc.
- Del mismo modo, los IDE también poseen diversas **extensiones** y **componentes para el IDE**. También llamados *pluggins*, las extensiones y componentes son añadidos de la aplicación IDE con el objetivo de mejorar su rendimiento, aspecto visual, funcionalidad, etc. Podemos encontrar *pluggins* para cambiar un IDE a otro idioma (los nombres de las ventanas, botones, mensajes…) o para incorporarle nuevos lenguajes de programación soportados.
- **Integración con *framework* populares.** Podemos definir *framework* como un marco, un esquema, un esqueleto o un patrón para el desarrollo e implementación de una aplicación o, mejor dicho, tipo de aplicaciones de propósito o características específicos.
- Poseen **depuradores**, para optimizar la implementación de nuestros programas.
- También ofrecen la posibilidad de **importar** y **exportar proyectos**, para hacer más sencillo el trabajo en diversos lugares, estaciones de trabajo, etc.
- Del mismo modo, ofrece **múltiples idiomas** para el entorno IDE, bien intercambiables desde algún apartado de su configuración, bien, como decíamos antes, mediante la instalación de *pluggins*.
- Incorporan **manuales de usuario y ayuda**.

INFORMACIÓN

Os recomendamos la lectura de este interesante *post* titulado "**Los 10 mejores editores de texto para desarrolladores**", donde se analizan los que, en la opinión de los autores de esta obra, son los mejores IDE de la actualidad: *https://hipertextual.com/archivo/2013/10/mejores-editores-de-texto-para-desarrolladores/*

8.1.2 COMPONENTES

A continuación, vamos a analizar los principales componentes de un entorno de desarrollo integrado, realizando un pequeño análisis sobre las mejoras que supone su inclusión en la aplicación de desarrollo.

Un editor de texto

Se trata de un programa con el que crear y modificar los archivos que contienen el texto plano (sin formato) que compone una aplicación o *software* de programación.

El editor de texto no siempre se encuentra integrado en un IDE de programación, sino que, en ocasiones, se puede encontrar como un programa independiente (caso de Notepad++, entre otros).

Tal y como comentábamos en el apartado anterior, los editores de texto suelen reconocer uno o varios lenguajes de programación, ofreciendo opciones que ayudan a la implementación de código, tales como reconocedores de sintaxis, formateos de palabras reservadas del lenguaje de programación, autocompletado de palabras, posibilidad de agrupar bloques de código, entre otras.

Las funciones más características de un editor son:

- Marcar regiones de texto.
- Búsqueda y reemplazo de texto.
- Copiar, cortar y pegar.
- Formatear.
- Deshacer y rehacer.
- Importar y exportar.

Algunos de los editores de texto, no integrados en otros IDE, más extendidos en el mundo de la programación son:

- Bloc de notas o Notepad, para Windows. Otro editor, más completo, también para este sistema operativo, es el mencionado *Notepad++*.
- Vi, para el sistema operativo Unix.
- Junto a otros sistemas multiplataforma, tales como jEdit, Emacs o Kate.

Editores de texto web

Un editor de páginas web es una aplicación que, aunque tiene la misma funcionalidad que un editor de texto normal, está diseñado para facilitar la creación de documentos en formato *html o xhtml*.

A diferencia del resto de editores, en un editor de texto web se puede visualizar el resultado de la página web que se está implementando, colocar distintos elementos sobre dicha vista previa de la página (generándose su código de programación asociado), ayudando, de este modo, en la implementación[12].

Algunos editores de *text web* son Dreamweaver, Microsoft Frontpage o Mozilla Composer.

Un compilador

Un compilador es un *software* que examina el código fuente de un lenguaje de programación para detectar posibles errores en su construcción y traduce dicho código original (escrito en un lenguaje de alto nivel, como Java, C, Delphi, etc.) a un lenguaje intermedio o un lenguaje máquina (dependiente del compilador).

INFORMACIÓN

Tal y como podemos estudiar en el libro ***Teoría, diseño e implementación de Compiladores de Lenguajes***, obra de Francisco Javier Martínez López y Alejandro Ramallo Martínez, la fase de evaluación de la corrección del código fuente, está dividida en tres subfases:

- **Análisis Léxico:** esta subfase se encarga de "leer la secuencia de caracteres del programa fuente, carácter a carácter, y la de agruparlos para formar las unidades con significado lógico, estos son los componentes léxicos o tokens".
- **Análisis Sintáctico**: cuyos objetivos se clasificar en tres:
  - "(…) Comprobar si la cadena de componentes léxicos, proporcionada por el Analizador Léxico, puede ser generada por la gramática que define el lenguaje fuente (Gramática Libre o Independiente del Contexto).
  - Construir el árbol de análisis sintáctico que define la estructura jerárquica de un programa y obtener la serie de derivaciones para generar la cadena de componentes léxicos. El árbol sintáctico se utilizará como representación intermedia en la generación de código.
  - Informar de los errores sintácticos de forma precisa y significativa. Además, deberá estar dotado de un mecanismo de recuperación de errores para continuar con el análisis".
- **Análisis Semántico:** el cual acometerá "dos actuaciones principalmente:
  - Anotar información semántica al árbol de análisis.
  - Añadir posibles errores semánticos detectados por la rutina".

Un intérprete

Un intérprete es un programa que se encarga de, a partir de un programa escrito o traducido a un lenguaje de alto nivel, analizarlo y, posteriormente, lanzarlo a ejecución (es decir, hacer que se ejecute).

Depurador

Un depurador (o *debugger*), es una herramienta de un IDE que permite probar y detectar errores en el código fuente escrito por el programador.

El depurador puede ejecutar totalmente el código fuente y devolver los resultados más significativos al programador, o bien, se puede ejecutar en modo *paso a paso*, para mostrar dichos datos relevantes, después de ejecutar cada instrucción de dicho código de programación (en diferentes puntos de ruptura). De esta forma, el programador podrá observar si la ejecución del programa en el depurador va generando los valores esperados.

12 Esto cada vez es más normal en editores de texto que no son web.

Control de versiones

Como hemos definido anteriormente, citando una definición de la enciclopedia digital Wikipedia, "(…) los **programas para control de versiones** son un grupo de aplicaciones originalmente ideadas para gestionar ágilmente los cambios en el código fuente de los programas y poder revertirlos, cuyo ámbito ha sido ampliado pasando del concepto control de versiones al de gestión de configuración de *software*, en el que se engloban todas las actividades que pueden realizarse por un equipo sobre un gran proyecto *software* u otra actividad que genere ficheros digitales (…)".

Las versiones más empleadas en este contexto son:

- Versión *alfa*: primera versión del programa. Generalmente se trata de una versión inestable y pendiente de depuración de errores. No suele estar completa en cuanto a la funcionalidad final del *software*
- Versión *beta*: primera versión con toda la funcionalidad operativa para comenzar con la primera fase de testeo.
- Versión *rc*: versión más robusta y con los errores depurados. Puede llegar a ser el producto final, a menos que aparezcan errores previos a la distribución del *software*.
- Versión *rtm*: también llamada versión final o definitiva. No suele contemplar muchos cambios con respecto a la *versión rc*.

Interfaz gráfica (GUI)

La interfaz gráfica (o *Graphical User Interface,* GUI) es una parte del IDE que actúa de interfaz de usuario, mediante un conjunto de imágenes y elementos gráficos, para representar la información y acciones disponibles en dicha interfaz. Básicamente proporciona un entorno visual sencillo con el que interactuar con el ordenador.

Refactorización

La refactorización (o *refactoring*) consiste en reestructurar el código fuente que se implementa, de forma que se modifique su estructura interna, evidentemente si modificar su comportamiento final.

8.1.3 LENGUAJES QUE SOPORTAN

Los entornos de desarrollo generalmente dan soporte a la gran mayoría de lenguajes de programación, tales como Java, C, C++, C#, Python, Delphi, Visual Basic, entre otros. No obstante, este aspecto no dejar de ser algo basado en "la oferta y la demanda", por lo que un lenguaje de programación muy extendido dispondrá de decenas de IDE para poder programar en dicho lenguaje, mientras que otros lenguajes poco extendidos o caídos en la obsolescencia, carecerán de tales posibilidades.

Por otra parte, también debemos destacar que un mismo IDE puede estar optimizado para trabajar con más de un lenguaje de programación, bien a través de sistemas de *pluggins* que se instalan, bien a través de diferencias versiones/distribuciones del IDE en cuestión. En este sentido, la herramienta versátil por excelencia es la plataforma *Ecilpse*.

INFORMACIÓN

El IDE Eclipse es uno de los más destacados entornos de desarrollo en cuanto a su calidad y por tener unas características muy valiosas: es totalmente multiplataforma y neutral al lenguaje.

Eclipse Consortium, además de soportar lenguajes como Java, C/C++ y Cobol, también posee otros proyectos para añadir a Eclipse el soporte de lenguajes tan diversos como Python, Eiffel, PHP, Ruby y C#.

Con respecto a las plataformas, el Eclipse Consortium proporciona instalaciones binarias para Windows, Linux, Solaris, HP-UX, AIX, QNX y Mac OS X.

Todo ello, como decíamos anteriormente, se soporta y evoluciona de forma tan rápida gracias al sistema de *pluggins*, con el que cientos de proyectos de desarrollo están siendo abarcados por un gran número de importantes empresas del sector.

Os recomendamos la lectura de este artículo, del que hemos obtenido los datos anteriores:
http://programacion.net/articulo/eclipse_i_historia_y_toma_de_contacto_288

8.2 ENTORNOS DE DESARROLLO EN EL ENTORNO DE LA BASE DE DATOS

A la hora de programar aplicaciones dependientes de una base de datos, los desarrolladores pueden plantear su trabajo desde dos paradigmas bien diferenciados:

- Trabajar con lenguajes de propósito general (como Java, C, Delphi, etc.), IDE para su desarrollo (como Eclipse o Netbeans) y *embeber* las sentencias SQL en el código fuente generado en ese determinado lenguaje de programación.
- Utilizar lenguajes de programación incrustados en los DBMS (como el lenguaje PL/SQL de Oracle).

La forma más extendida en el mundo de la informática para crear programas con bases de datos es la primera. Así pues, nuestra propuesta, por ejemplo, para hacer un *software* de aplicación (es decir, un programa de escritorio, no portal web ni una aplicación móvil), sería la de instalar una versión de Eclipse que ya incluya los *pluggins* para la elaboración de entornos gráficos e implementar la aplicación en Java. Un DBMS que recomendaríamos sería MySQL, por su versatilidad, por la gran comunidad de usuarios que aseguran una gran cantidad de documentación, manuales, soporte y, en general, ayuda, y porque tiene una distribución *open source* muy robusta y versátil.

NOTA

Para conectar Java y MySQL es necesario instalar una serie de librerías. Esta tarea, con el IDE Eclipse es tan sencilla como instalar un paquete con la API de conexión (generalmente archivo con extensión *jar*).

Podéis aprender a hacer esta conexión siguiendo el manual de este *post*:
http://www.hermosaprogramacion.com/2014/07/mysql-java-conectar-como/

No obstante, dado que esta obra está totalmente orientada a las bases de datos y sus sistemas de gestión de bases de datos, a lo largo de este capítulo vamos a proponer opciones para la segunda alternativa. Por ello, en adelante vamos a trabajar con las siguientes ideas:

- Utilizaremos bases de datos relacionales y, como DBMS, propondremos el uso de MySQL. Seguidamente explicaremos cómo instalar dicho sistema de gestión de base de datos.
- Como IDE específico para el uso de base de datos emplearemos MySQL Workbench, el que también explicaremos cómo instalar y su operativa básica.

- El lenguaje de programación que utilizaremos en sucesivos apartados será el propio de MySQL para definir bucles, sentencias condicionales, procedimientos, etc.

Tras comentar el estado del arte de la programación con bases de datos, proseguiremos definiendo el concepto de **entorno de desarrollo en el entorno de la base de datos.**

8.2.1 DEFINICIÓN DE ENTORNO DE DESARROLLO EN EL ENTORNO DE LA BASE DE DATOS

Un IDE de desarrollo en el entorno de la base de datos se diferencia fundamentalmente de un IDE de desarrollo de propósito general en la existencia de una potente herramienta (de hecho, en algunos IDE es la única, en otros la principal) para diseñar la base de datos. Esta herramienta, en la mayoría de los casos, suele ser gráfica y muy intuitiva, y permite la exportación de dicha definición a su código fuente SQL.

Del mismo modo, el IDE en el entorno de la base de datos también posee su editor, compilador e intérprete, generalmente más orientado a la edición, compilación y ejecución de consultas (aunque también, en algunos IDE, se puede programar en su lenguaje propio).

Estos son los componentes más extendidos en esta clase de entornos de desarrollo.

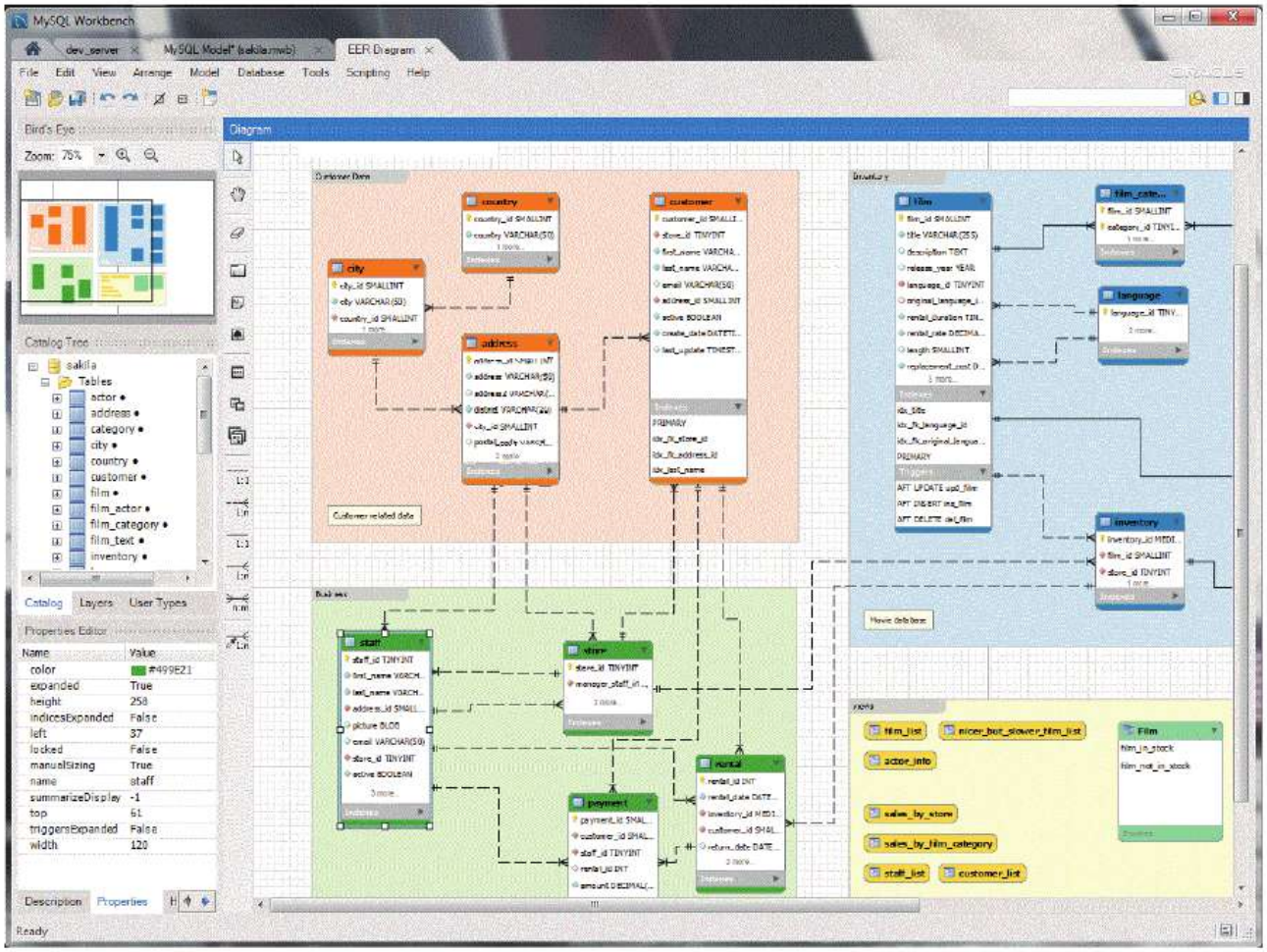


Figura 8.1. *Ejemplo de diagrama Entidad-Relación de MySQL Workbench para el diseño de una base de datos relacional*

8.2.2 INSTALACIÓN DE MYSQL Y MYSQL WORKBENCH

Desde hace algún tiempo, Oracle, empresa poseedora de MySQL y los productos derivados del DBMS, implementó un servicio en el que los usuarios de MySQL pueden comprobar los productos que tienen instalados y llevar a cabo su gestión de descargas, sin tener que visitar la web de la compañía. Este instalador de productos se denomina **MySQL Installer.**

Así pues, lo primero que explicaremos será cómo descargar este instalador y, posteriormente, a través de él, descargaremos y seguidamente instalaremos las dos herramientas con las que trabajaremos a lo largo de este capítulo.

8.2.2.1 Descargar e Instalar MySQL Installer

MySQL Installer se descarga de la página de descargas de MySQL:

http://dev.mysql.com/downloads/

A simple vista, en esta URL aparecen los productos susceptibles de ser descargamos pero, al hace clic sobre cualquier de ellos, el siguiente enlace nos derivará a la descarga e instalación de MySQL Installer en lugar del producto elegido.

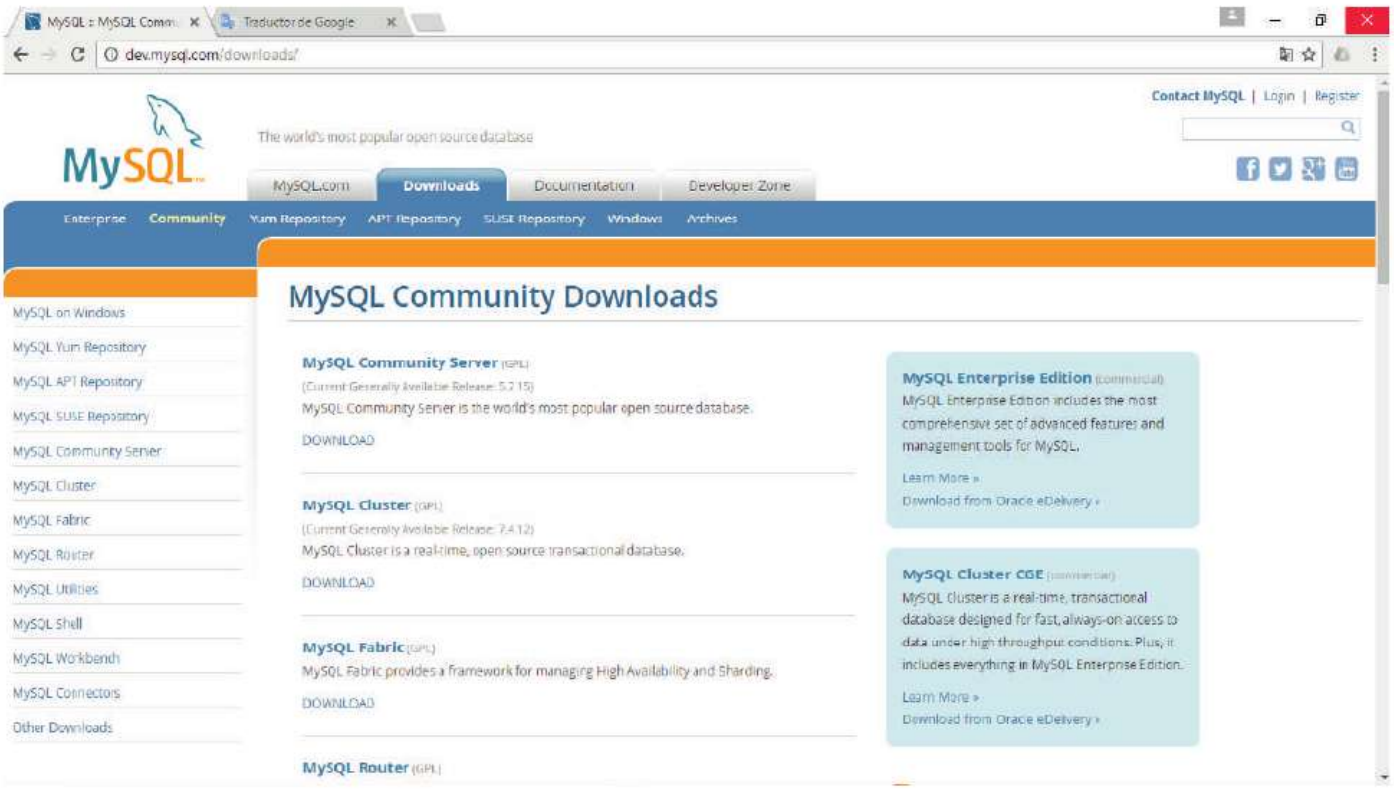


Figura 8.2. *Vista de la página de descargas de MySQL*

Esto puede generar, *a priori*, algún tipo de confusión, pero cuando estás familiarizado con la forma de trabajar, el hacer uso del instalador, sin tener que buscar el producto MySQL por Internet, todo resulta más sencillo.

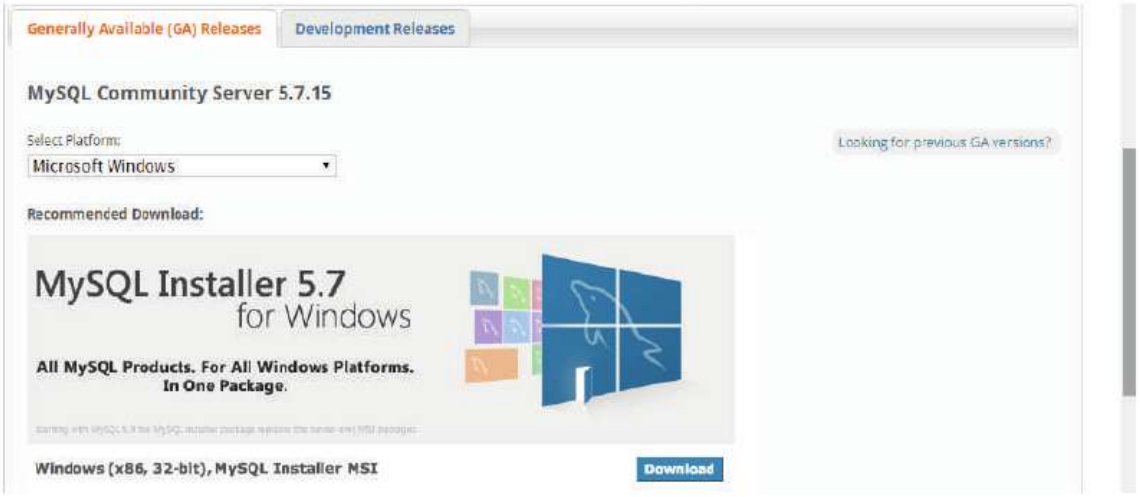


Figura 8.3. *Vista del enlace para descargar* ***MySQL Installer***

Como se puede comprobar, la descargas es muy intuitiva, simplemente debemos seleccionar el sistema operativo y pulsar en el botón de *Download* (ver Figura 8.3). Seguidamente, se abrirá una nueva página para seleccionar las características del sistema operativo y volvemos a pulsar *Download* en la opción escogida.

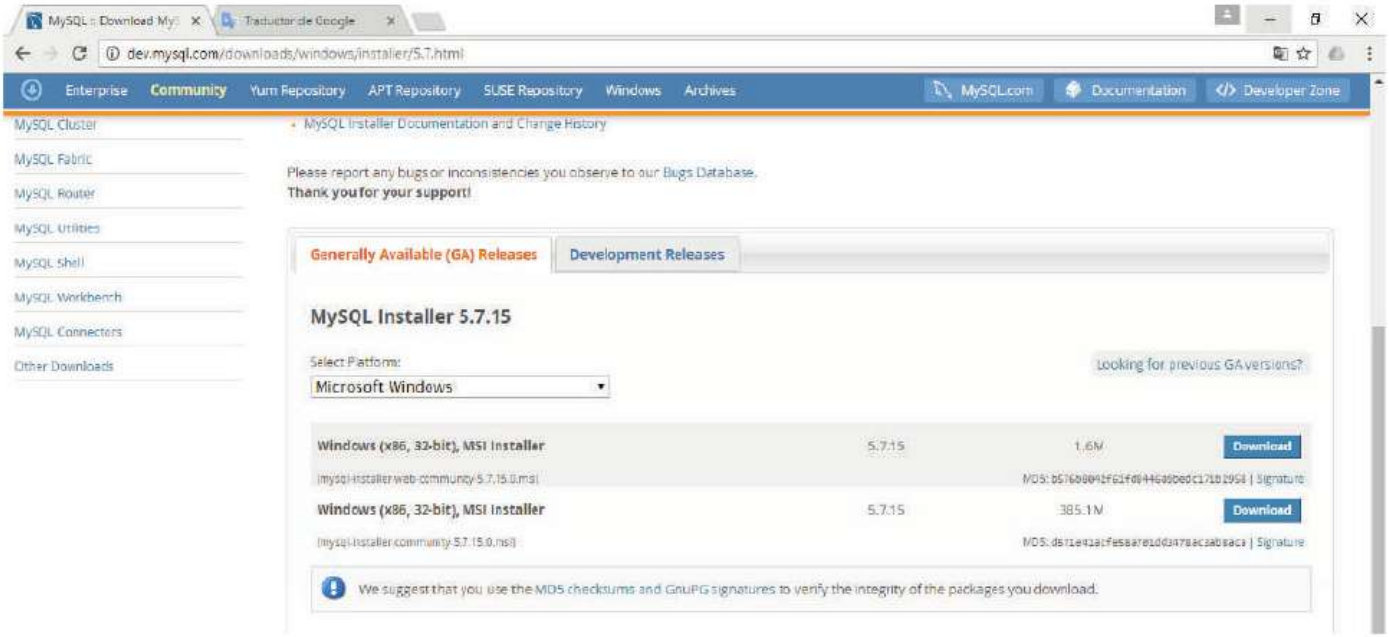


Figura 8.4. *Segunda ventana del proceso de descarga de* ***MySQL Installer***

Y, por último, antes de iniciar la descarga, nos pide que nos demos de alta en la comunidad (o autenticarnos si ya somos miembros).

También es posible saltarnos este paso, como vemos en la siguiente figura.

Figura 8.4. *Último paso del proceso de descarga de* ***MySQL Installer***

Una vez descargado el *software*, el proceso de instalación comienza con la aceptación de los términos de la licencia de uso del producto, que tendremos que aceptar.

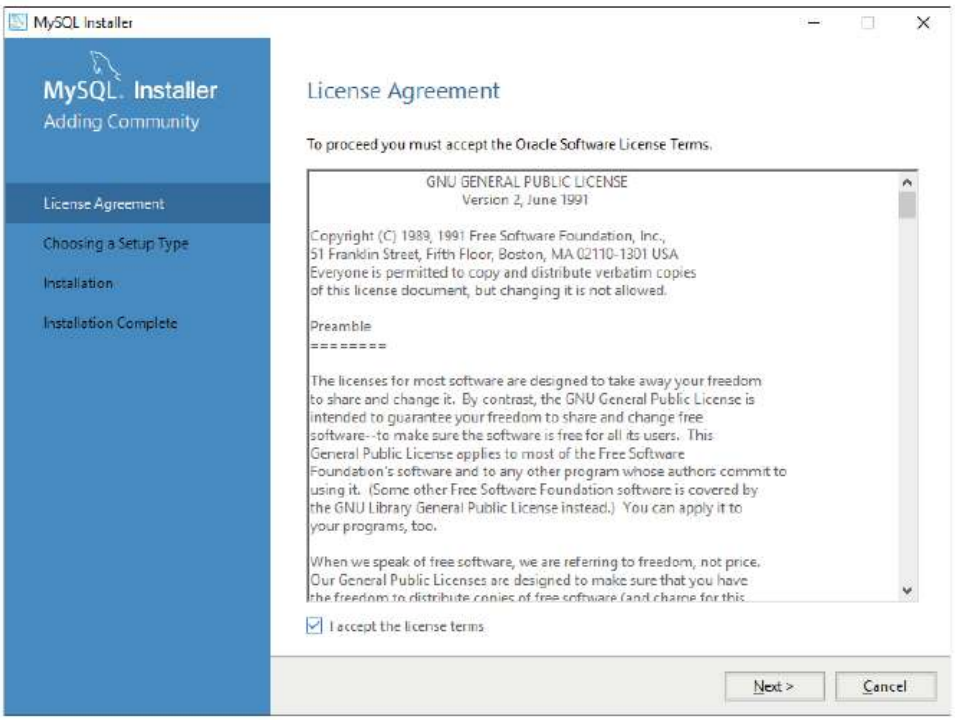


Figura 8.5. *Instalación de* ***MySQL Installer:*** *aceptación de los términos de la licencia*

La primera vez que instalemos el *software*, tendremos que, a su vez, instalar alguno de los productos de MySQL. Así pues, en esta primera ocasión, nosotros instalamos MySQL Server, por lo que el resto del proceso de instalación lo explicaremos en la sección siguiente.

8.2.2.2 Instalación de MySQL Server

Continuando con MySQL Installer, nos aparece una ventana con las opciones de productos a instalar (nótese que esta ventana aparecerá solo en esta primera instalación, ya que el entorno de MySQL Installer es diferente).

Nosotros seleccionaremos lo que nos interese, en este caso, el servidor, como podemos ver en la Figura 8.6.

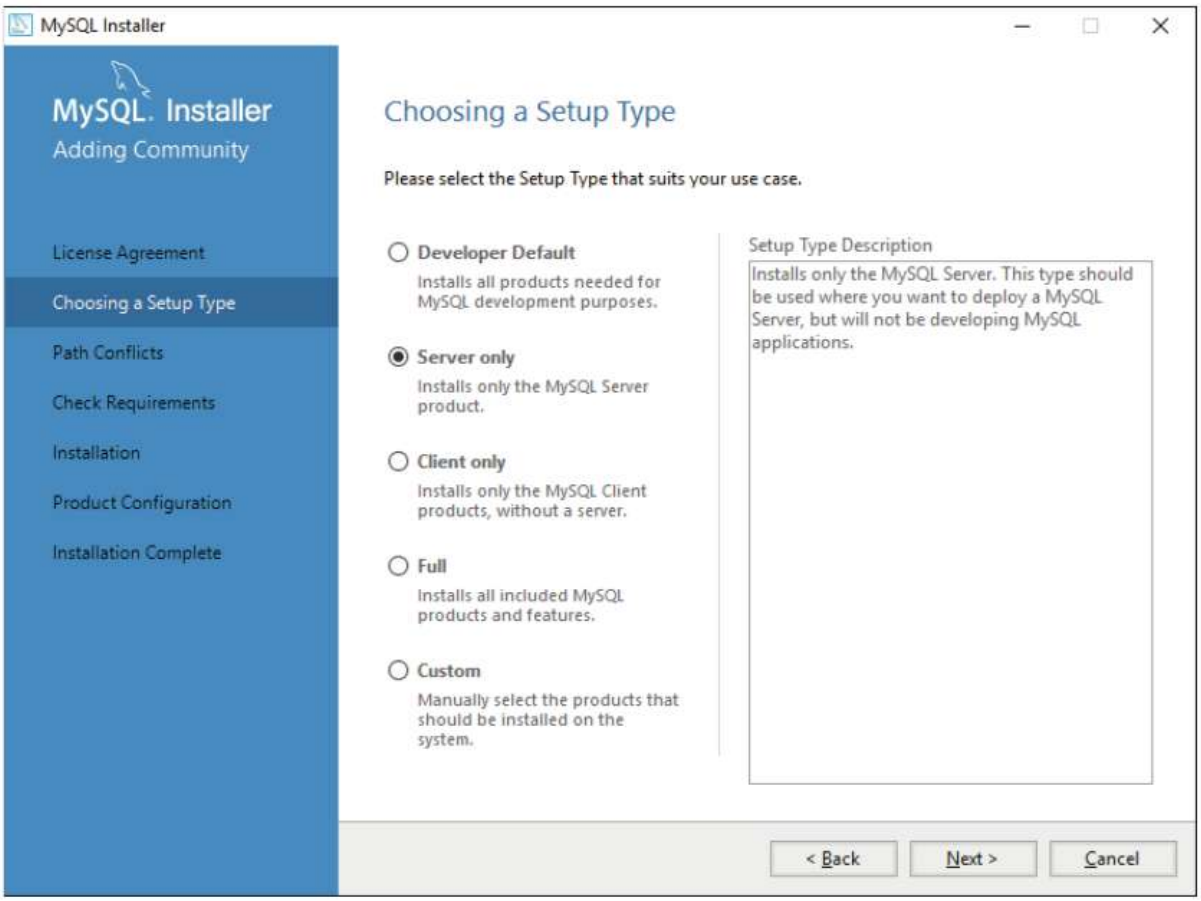


Figura 8.6. *Selección del* software *a instalar*

Tras ello, el instalador nos pedirá la ubicación donde queremos que se instale el producto, notificándonos si estamos incurriendo en algún tipo de conflicto.

Para finalizar, el resto de pasos son de notificación de que empieza el proceso de instalación, que debemos validar pulsando en *Execute* y, posteriormente, de notificación de instalación.

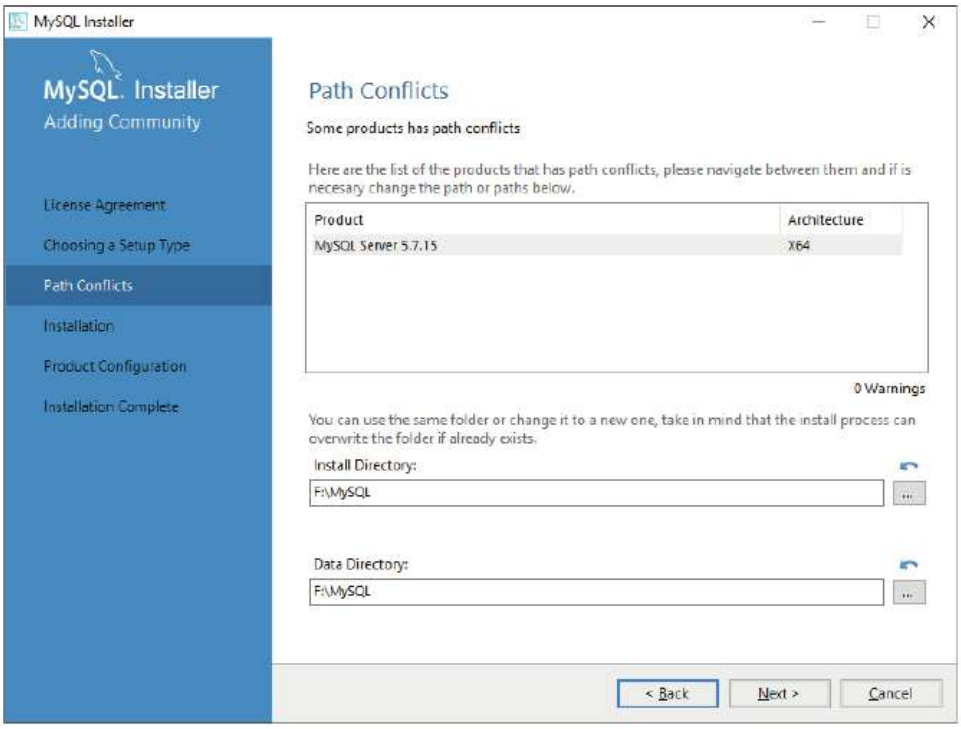


Figura 8.7. *Selección del lugar donde instalar el producto*

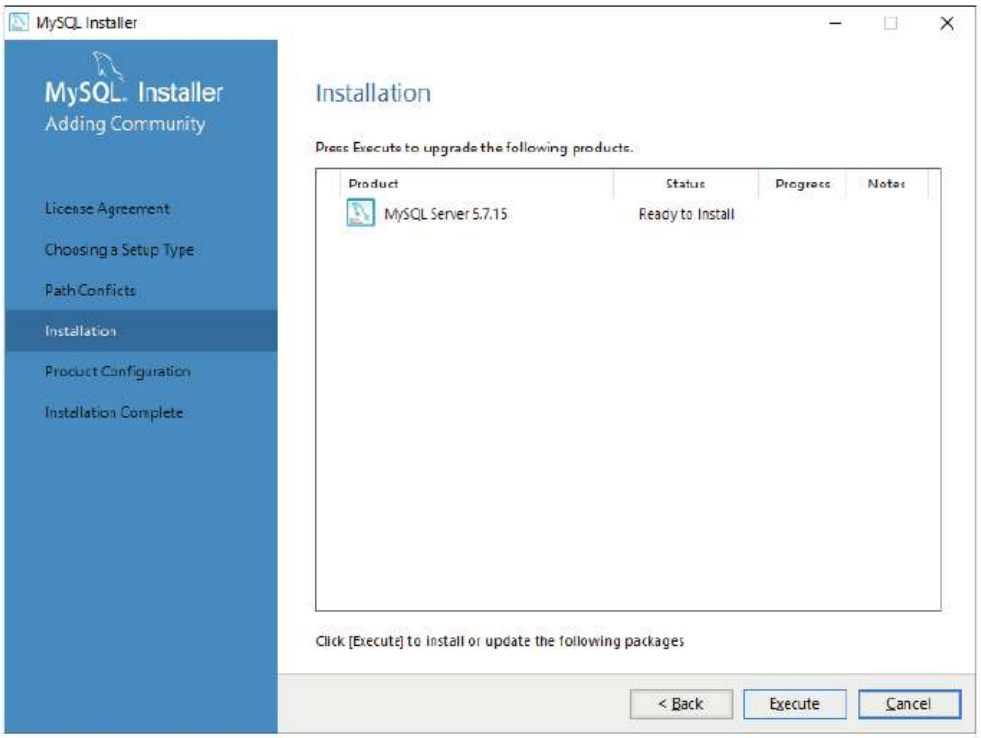


Figura 8.8. *Notificación para confirmar el inicio de la instalación*

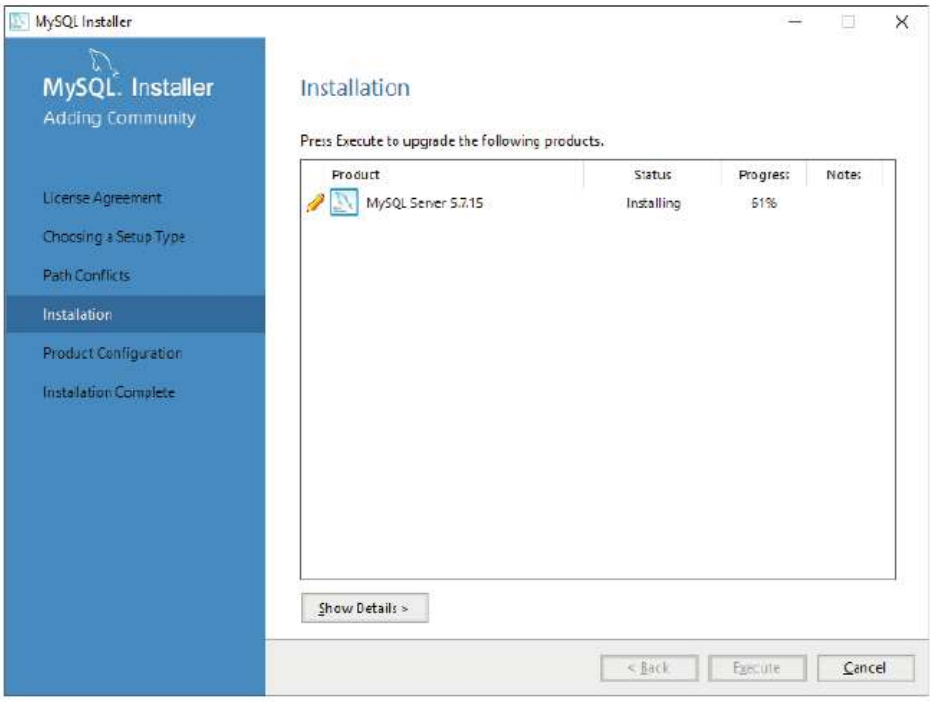


Figura 8.9. *Proceso de instalación en curso*

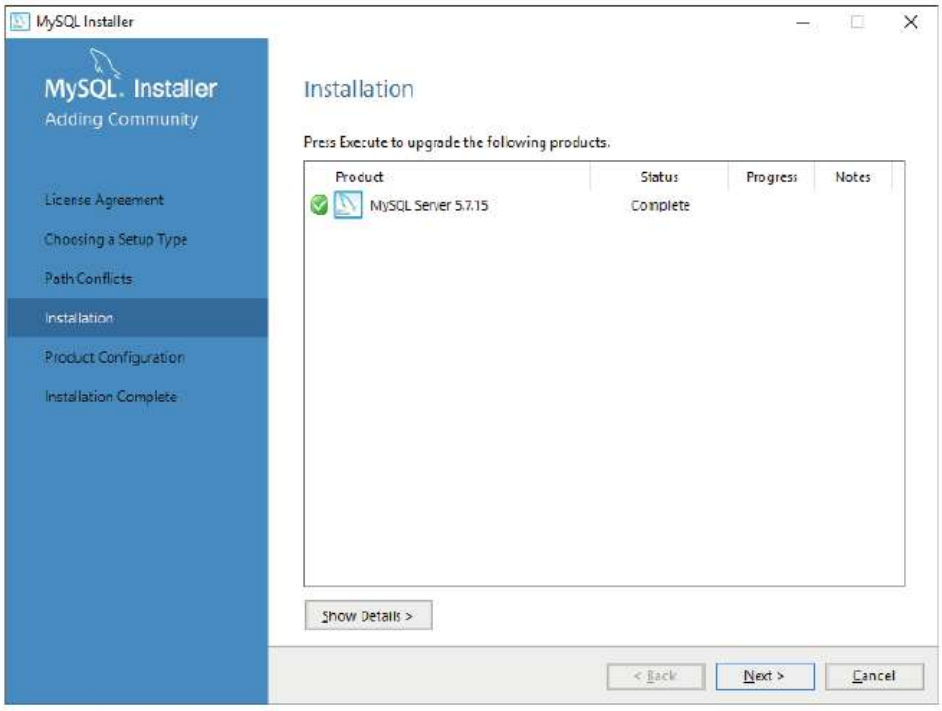


Figura 8.10. *Notificación de final del proceso de instalación*

Una vez instalado MySQL Server, nos pedirá que lo configuremos, para los cual se seguirán sucediendo las diferentes pantallas para finalizar satisfactoriamente la configuración del producto. Veremos las opciones de configuración en las siguientes figuras.

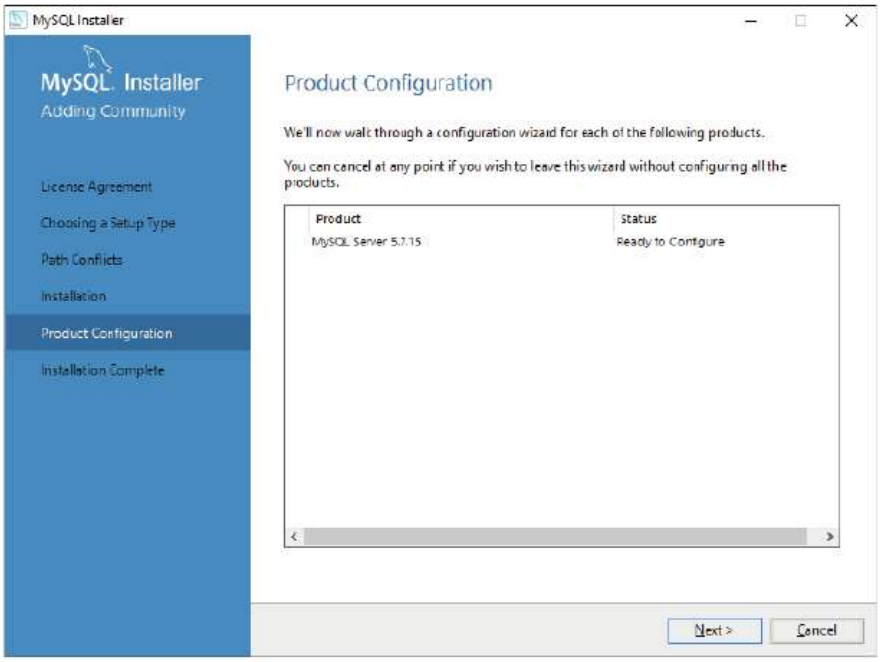


Figura 8.11. *Inicio del proceso de configuración*

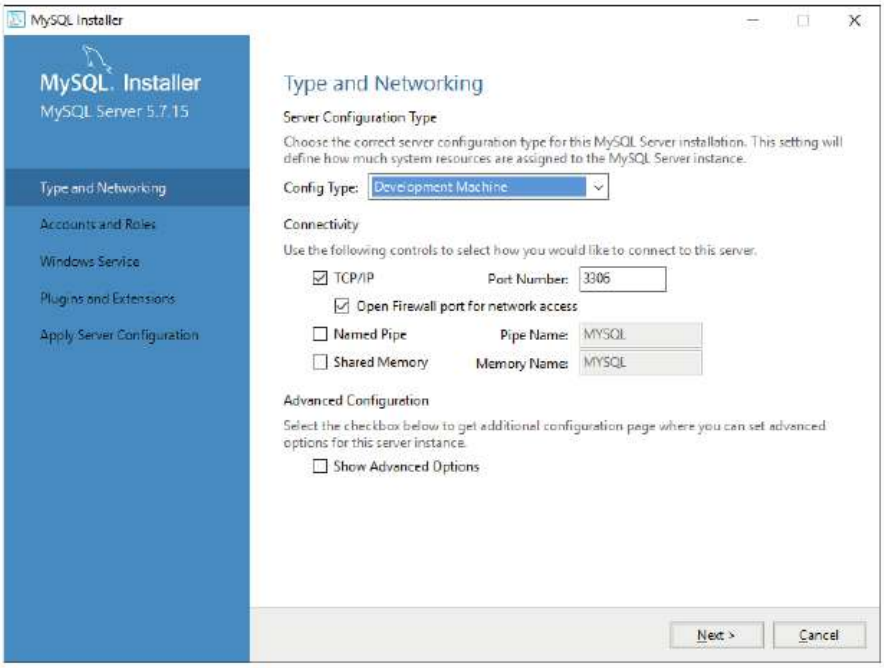


Figura 8.12. *En esta ventana es importante especificar el puerto (por defecto el 3306 aunque se puede cambiar)*

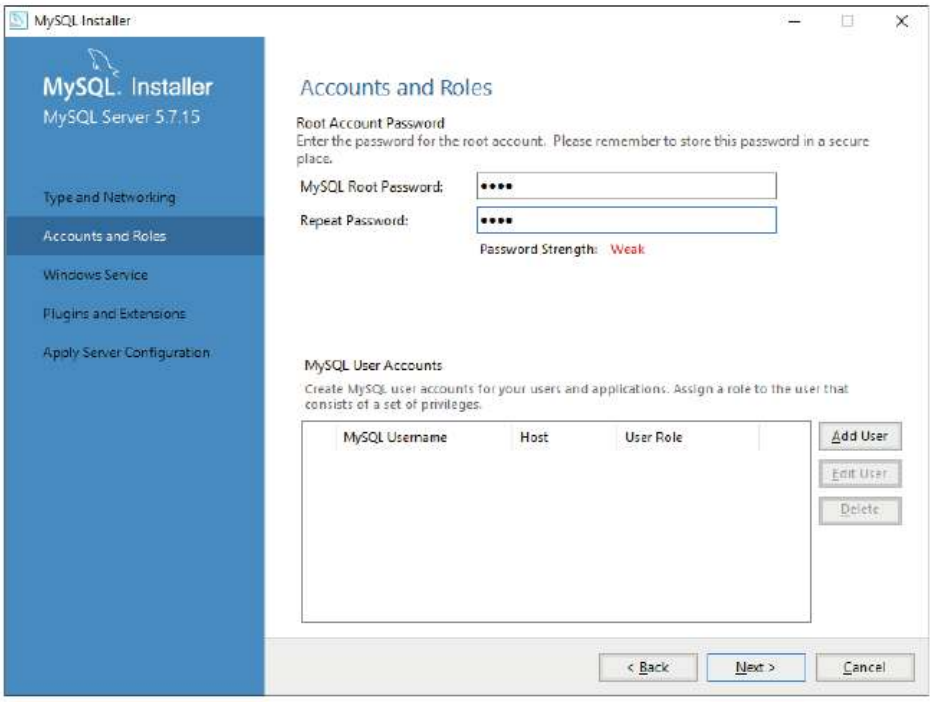


Figura 8.13. *Muy importante la selección de una buena contraseña, que será necesaria para acceder al servidor*

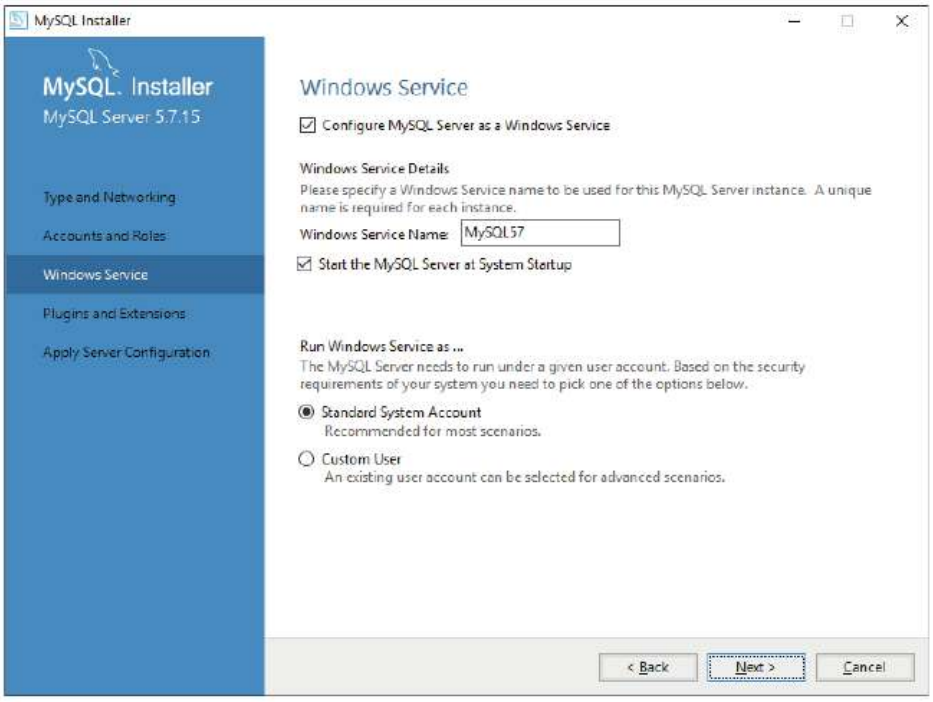


Figura 8.14. *En esta ventana recomendamos mantener la configurar por defecto*

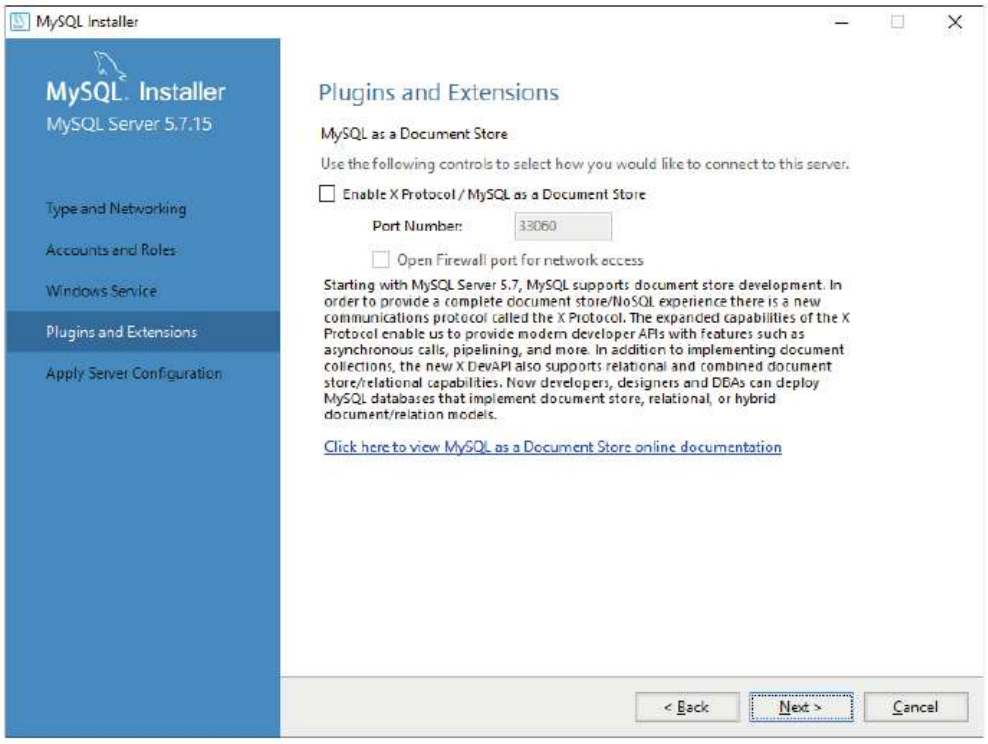


Figura 8.15. *Igualmente, en esta ventana, también pulsaremos solo la opción de "Siguiente"*

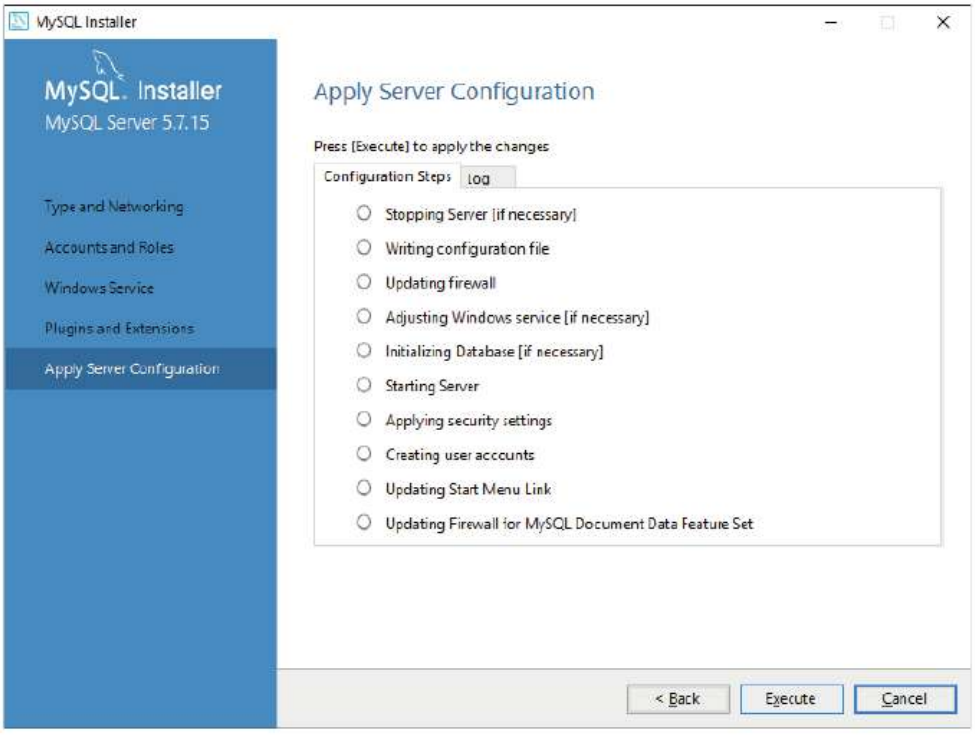


Figura 8.16. *Ordenamos que se ejecute la configuración establecida*

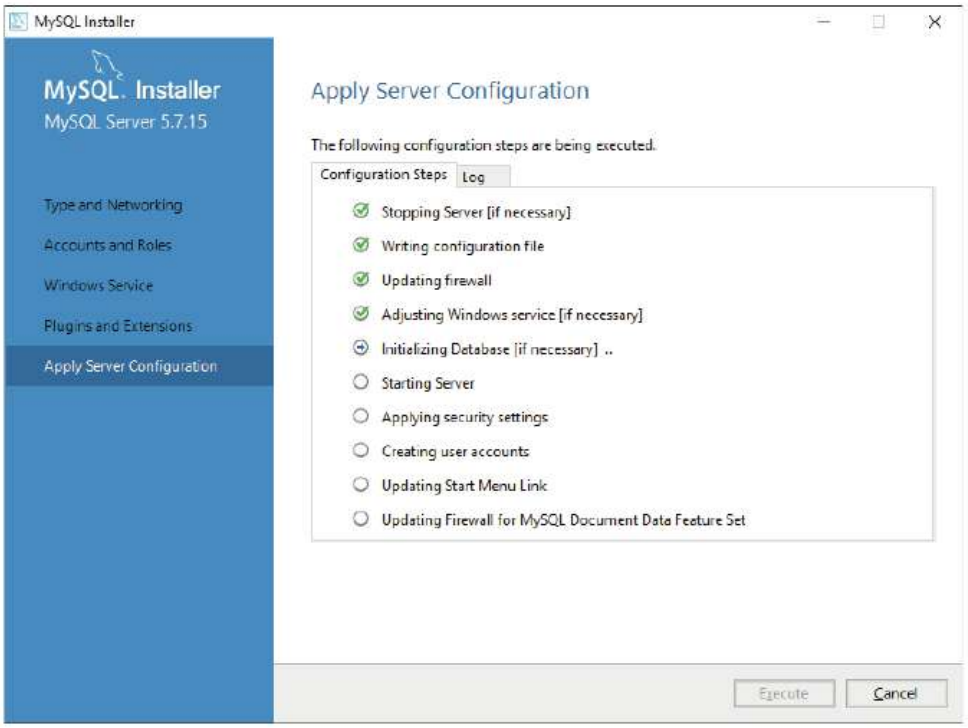


Figura 8.17. *Proceso de configuración*

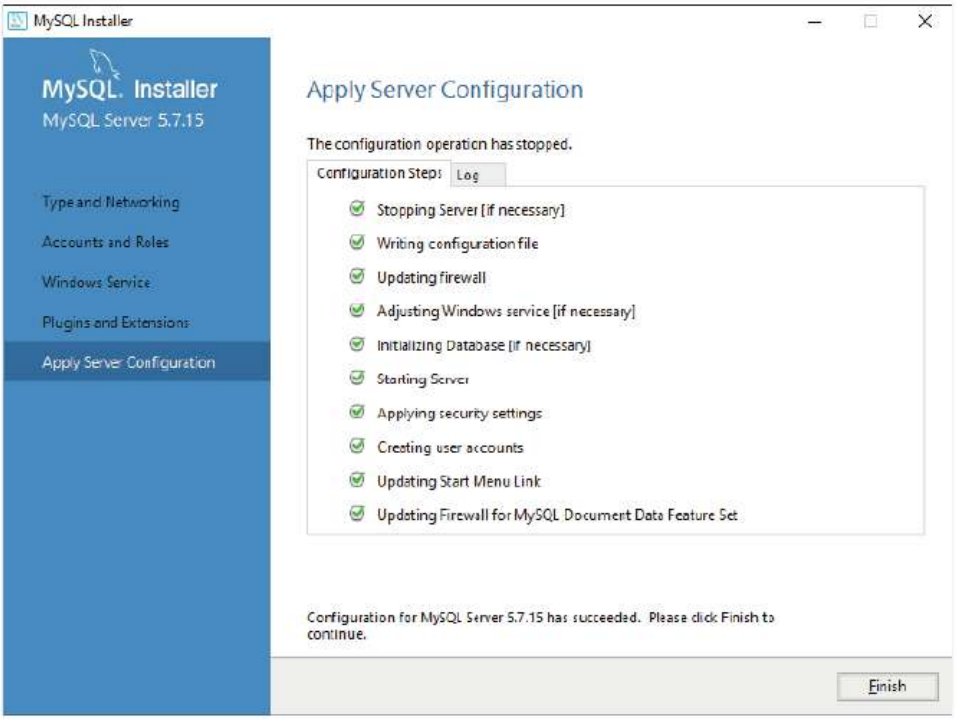


Figura 8.18. *Configuración finalizada*

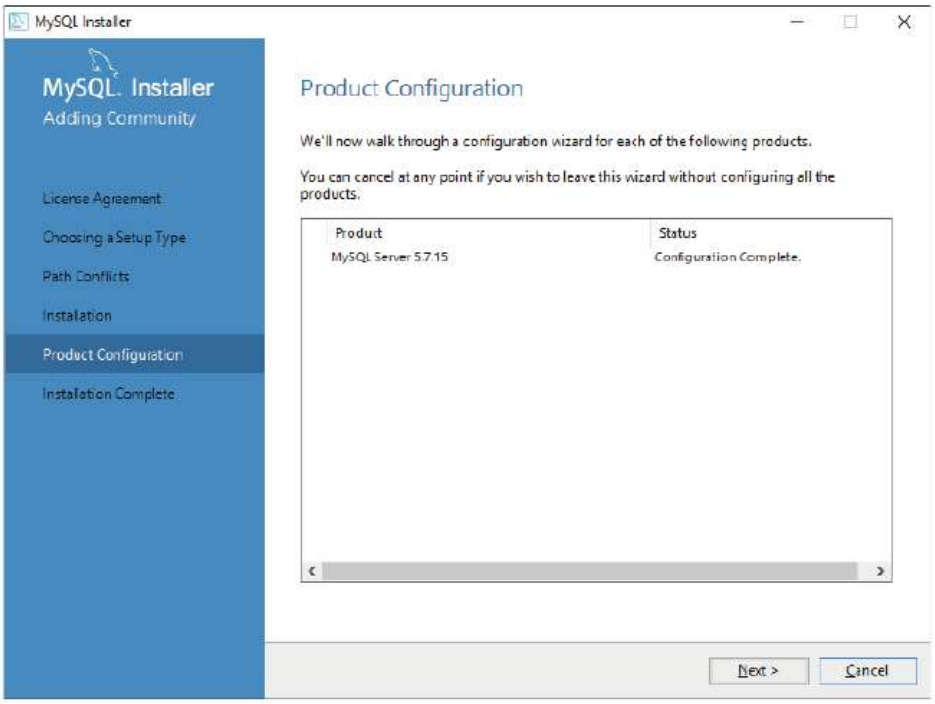


Figura 8.19. *Última ventana del proceso de configuración*

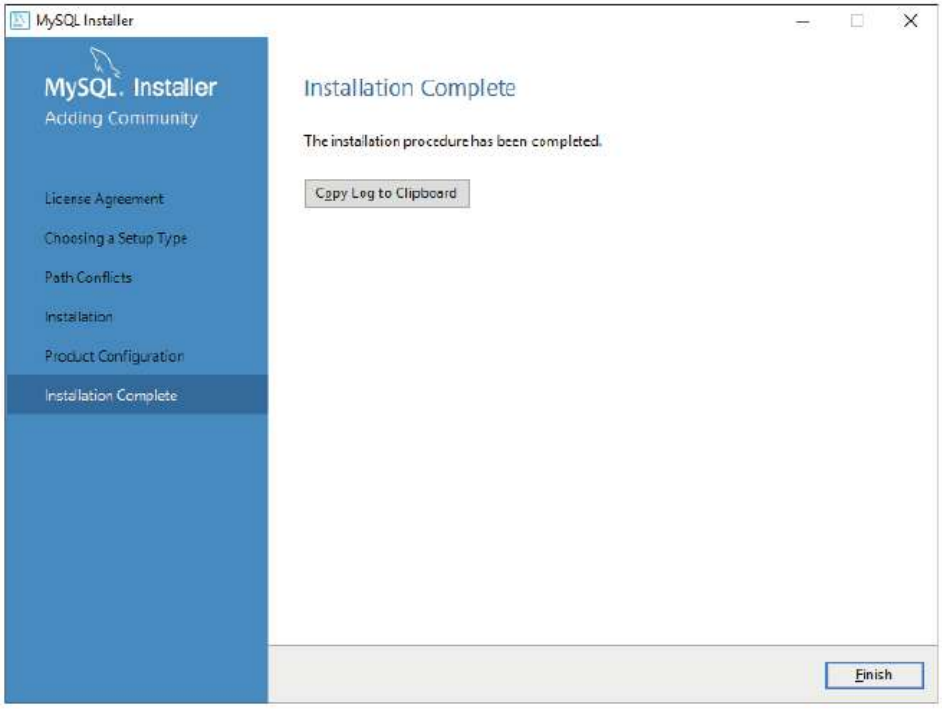


Figura 8.20. *Última ventana del proceso de instalación*

Con el proceso de instalación finalizado, cada vez que abramos MySQL Installer nos aparecerán los productos instalados en nuestro ordenador, podremos reconfigurarlos, eliminarlos e ir añadiendo cuantos queramos utilizar.

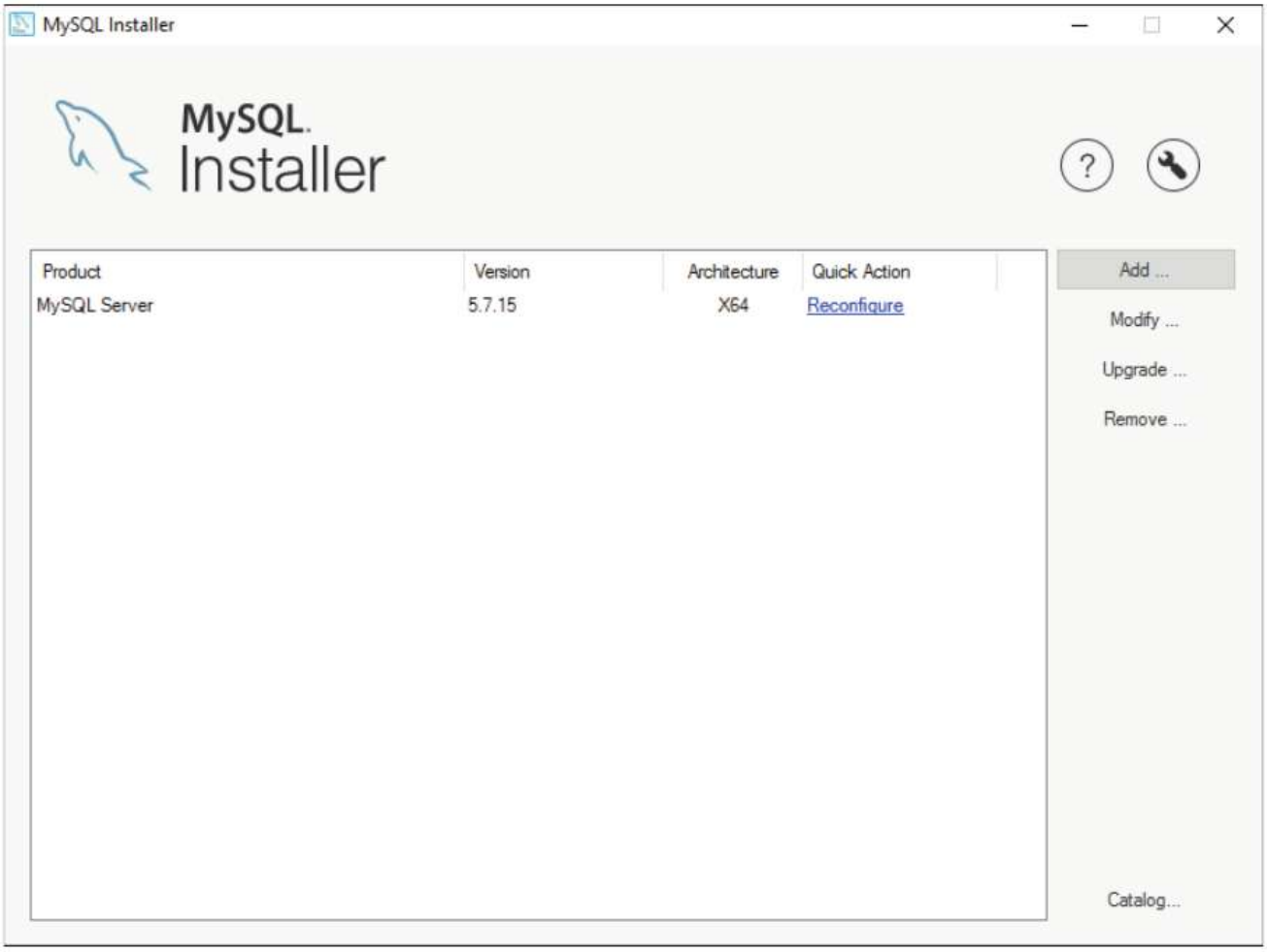


Figura 8.21. *Ventana principal de* ***MySQL Installer***

8.2.2.3 Instalación de MySQL Workbench

Como acabamos de comentar en la sección anterior, para poder instalar este *software* tendremos que entrar en la herramienta MySQL *Installer* y añadirla a la lista de aplicaciones que ya tengamos (pulsando el botón *Add…*).

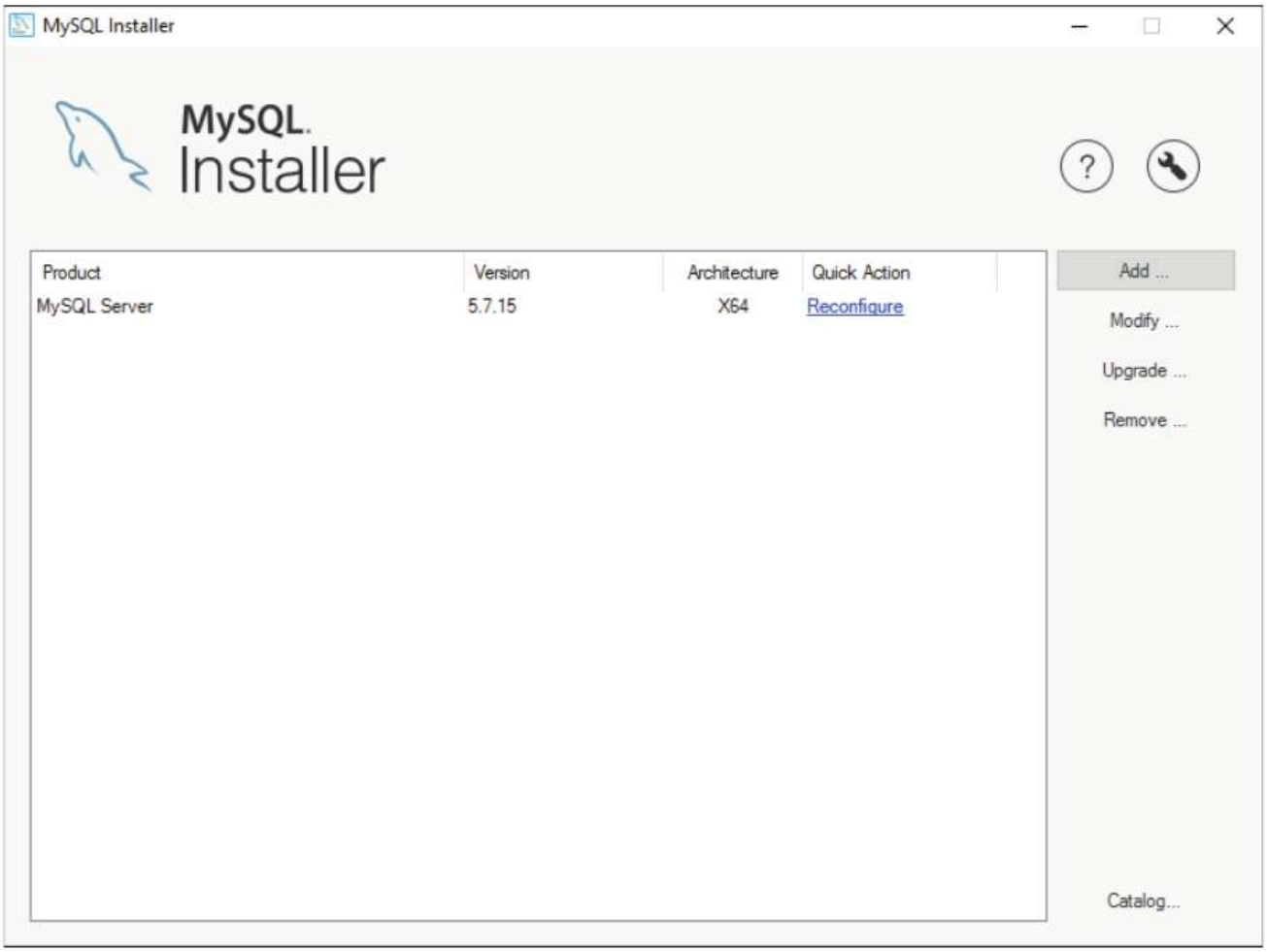


Figura 8.22. *Ventana principal de* ***MySQL Installer*** *con la opción marcada de añadir nuevo* software

Seguidamente se mostrará una nueva ventana donde podremos elegir el nuevo *software* a incorporar. En este caso, seleccionaremos la última versión del producto, MySQL Workbench 6.3.7 en su versión para 64 bits.

Como se puede ver en la Figura 8.23 y 8.24, para ir añadiendo productos bastará con seleccionarlos y pulsar sobre el botón para que se sitúen en el cuadro de diálogo de la parte derecha de la ventana.

Por otra parte, si queremos deseleccionar un producto, lo quitaremos de dicho cuadro de diálogo del lado derecho, pulsando el botón .

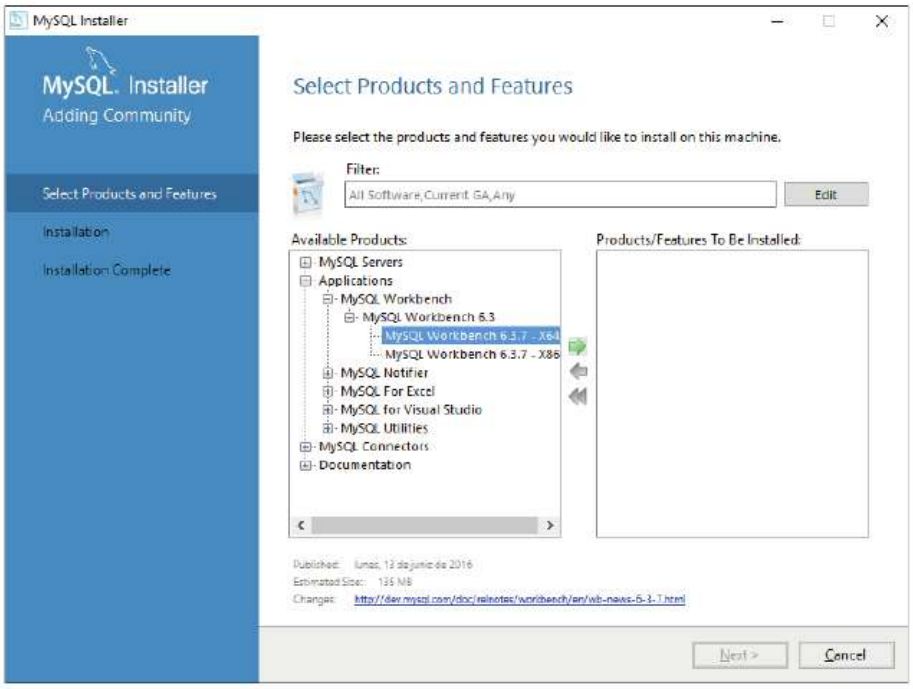


Figura 8.23. *Ventana de selección de nuevos productos de* ***MySQL Installer***

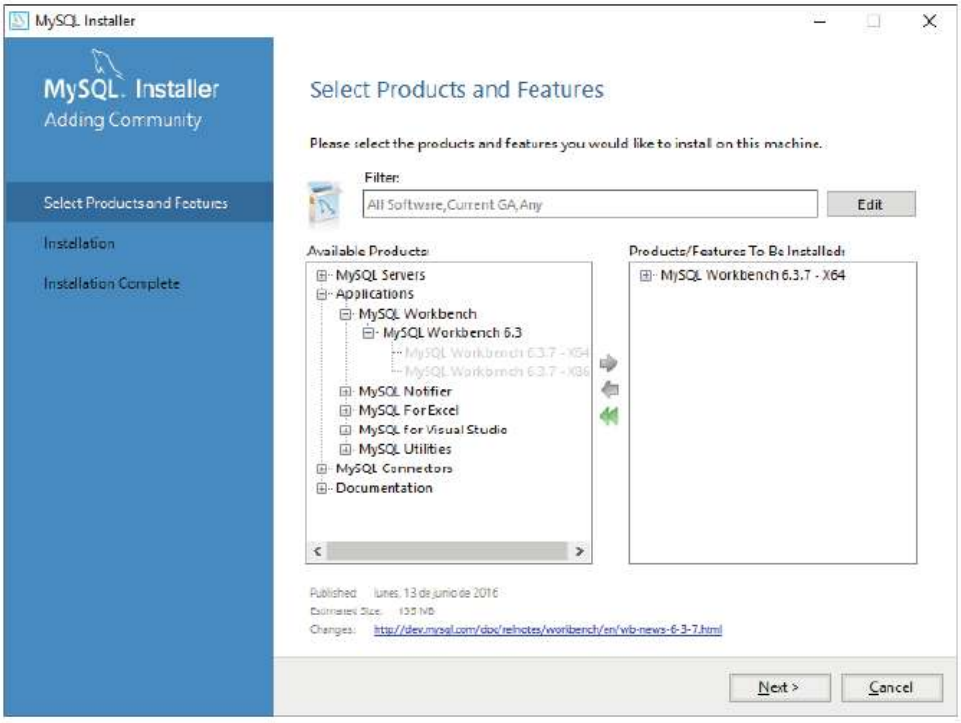


Figura 8.24. *Ventana de selección de nuevos productos de* ***MySQL Installer***

Seguidamente, comenzará el proceso de instalación de MySQL Workbench, siguiendo las mismas ventanas que veíamos anteriormente con la instalación MySQL Server. Las vemos en las figuras sucesivas.

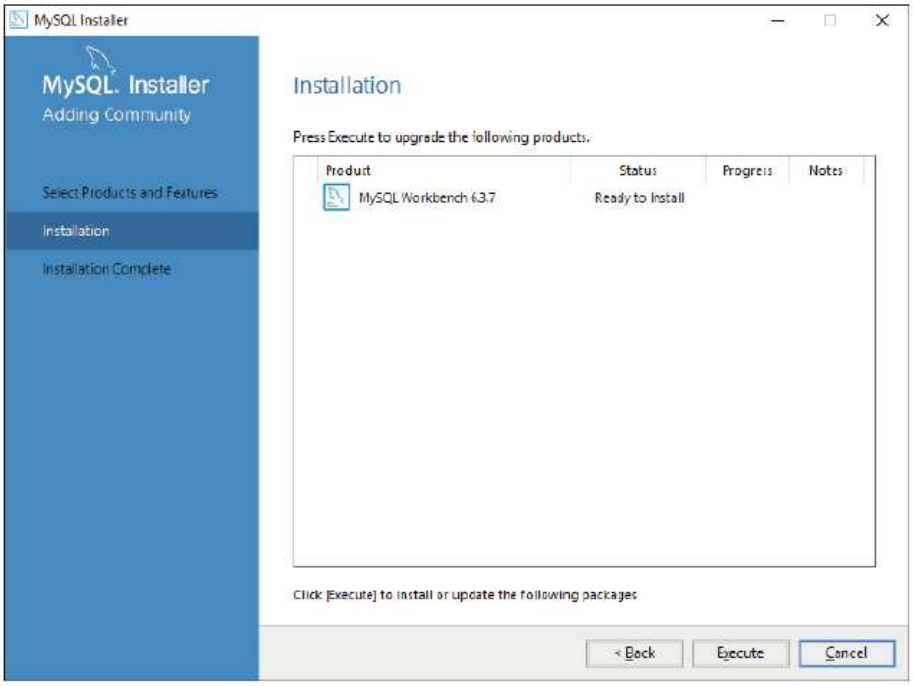


Figura 8.25. *Comienzo de instalación de* ***MySQL Workbench***

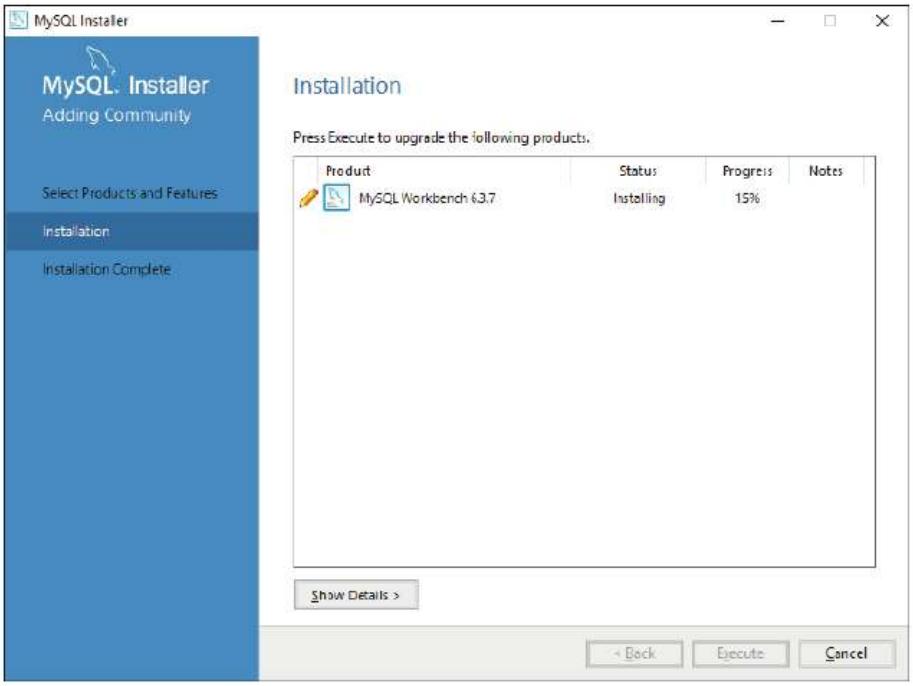


Figura 8.26. *Proceso de instalación de* ***MySQL Workbench***

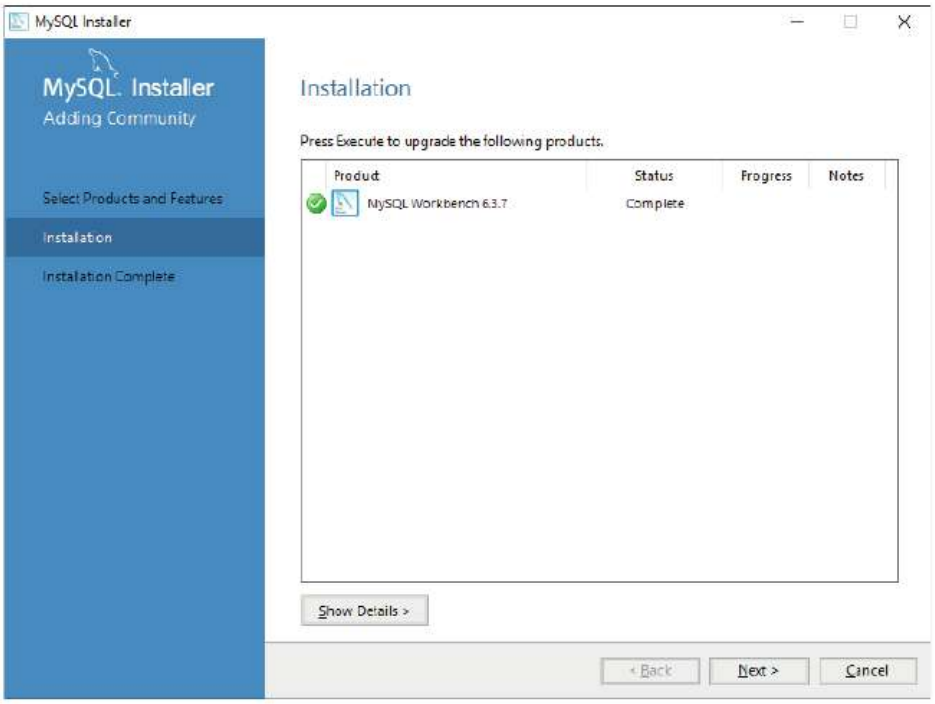


Figura 8.27. *Finalización de la instalación de* ***MySQL Workbench***

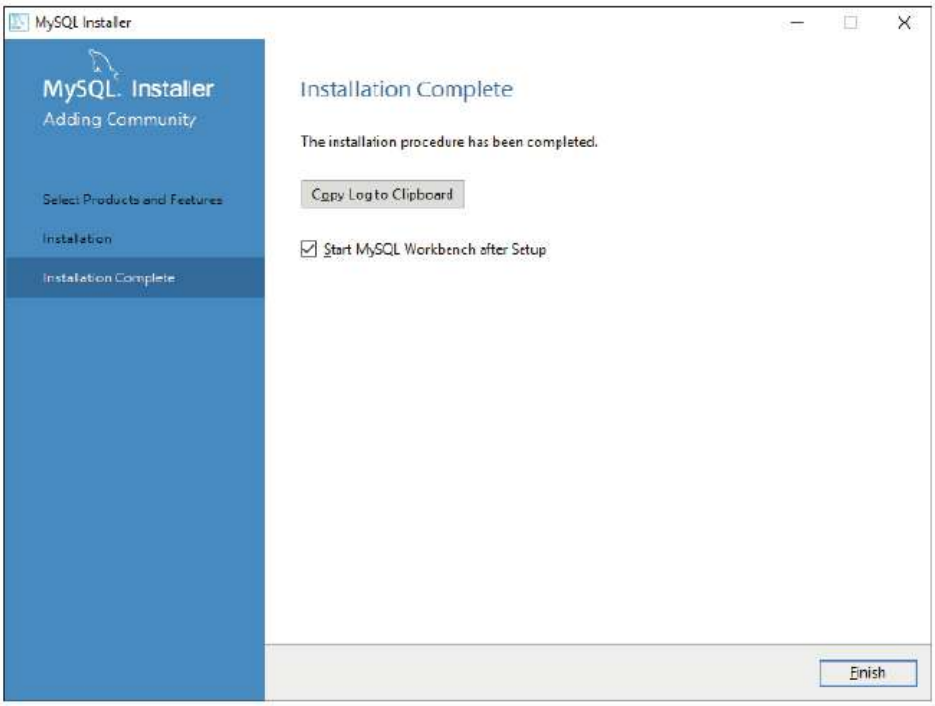


Figura 8.28. *Una vez terminada la instalación de* ***MySQL Workbench*** *podemos entrar en el programa o finalizar el proceso, volviendo a la ventana principal de* ***MySQL Installer***

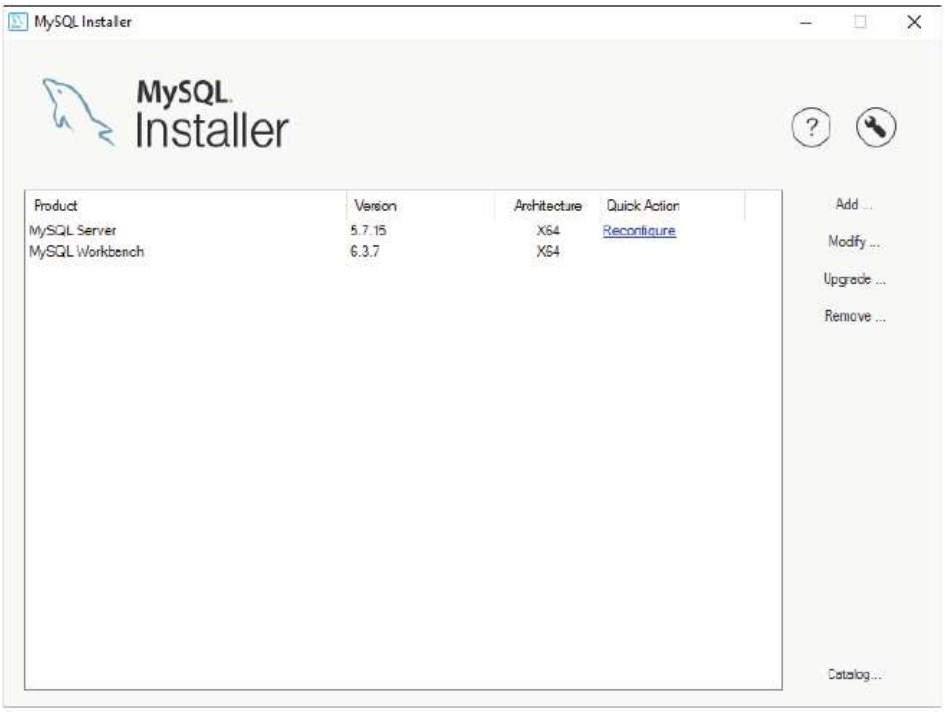


Figura 8.29. *Ventana inicial de* ***MySQL Installer*** *con los programas instalados*

8.2.2.4 Mis primeros pasos con MySQL Workbench

Con la realización de este apartado pretendemos poner a la disposición del lector un pequeño manual con las dos tareas principales que podemos llevar a cabo en pos de cubrir las necesidades de este libro:

- Elaborar un diagrama Entidad-Relación como se ha diseñado en capítulos anteriores y poder exportarlo a un *script* en SQL para poder embeberlo en cualquier lenguaje de propósito general o directamente trabajar con la base de datos creada.
- Utilizar el editor de código para implementar y probar nuestras consultas y códigos de programación de MySQL que comentaremos en sucesivos apartados.

Para ello, a este breve manual le acompañaremos un ejercicio guiado en el que crearemos la base de datos con la que hemos estado trabajando anteriormente (la definida en el Entidad-Relación del Ejemplo 3.1).

Mi primera base de datos

El primer paso consistirá en crear una nueva base de datos (o modelo, como es definido en la herramienta). Para ello, seleccionaremos la opción *File* → *New Model* o pulsaremos en el botón de añadir modelo tal y como vemos en la siguiente figura.

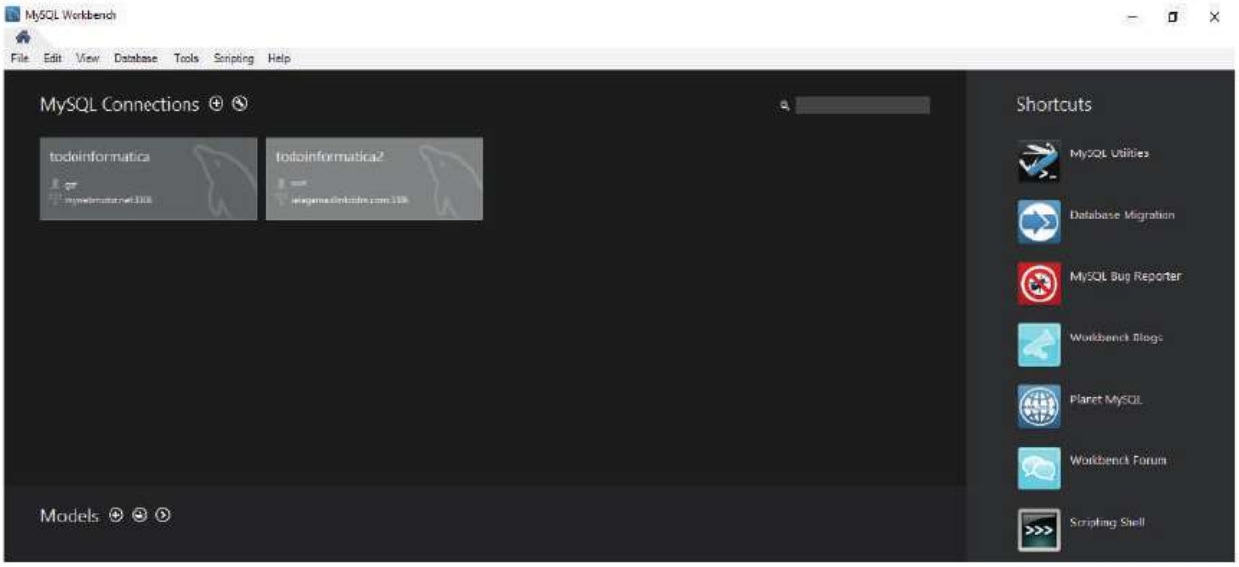


Figura 8.30. *Ventana inicial de* ***MySQL Workbench*** *para la gestión de los modelos almacenados y la creación de nuevos modelos*

Como podemos ver en la Figura 8.30, además de gestionar nuevo modelos locales, también se pueden gestionar y añadir bases de datos ya creadas y ubicadas en servidores externos, como las dos que vemos en la figura, cuyos nombres son *todoinformatica* (en el dominio *mywebmotor.net*) y *todoinformatica2* (base de datos ubicada en *kelagama.dlinkdns.com*).

Una vez pulsada la opción de añadir modelo, se genera un nuevo esquema de base de datos (llamado *MySQL Schema*), con un nombre por defecto (*mydb*). Dicho nombre puede ser cambiado haciendo doble clic en el citado nombre, donde hemos indicado en la Figura 8.31. Para nuestro ejemplo, el nombre seleccionado será *academia*, como hemos indicado en la Figura 8.32.

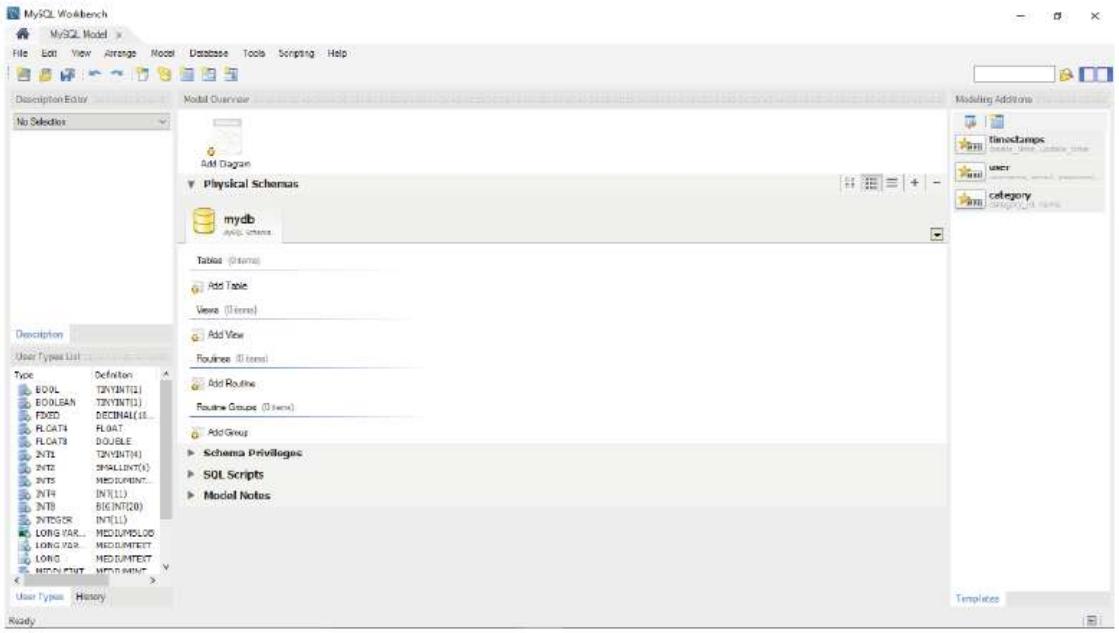


Figura 8.31. *Ventana para la creación de un nuevo esquema de base de datos*

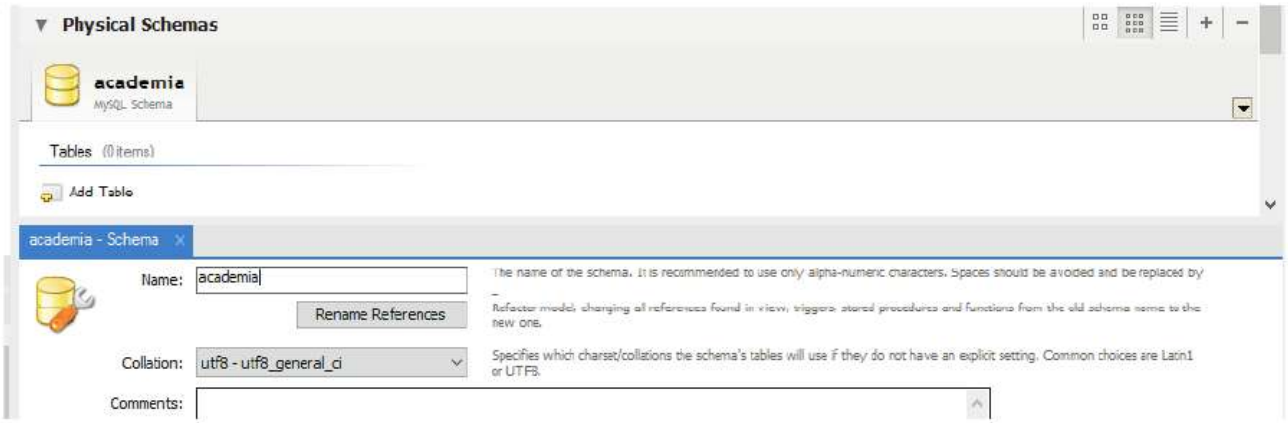


Figura 8.32. *Cambiando el nombre a nuestro esquema de base de datos*

Si volvemos a la pantalla principal de definición de esquemas de la Figura 8.31, podemos observar que nos aparecen cuatro opciones fundamentales, con diversas acciones en cada una de ellas:

- **Physical Schemas.** Podemos, además de cambiar el nombre al esquema de base de datos, realizar las siguientes tareas:
  - Añadir tablas al esquema.
  - Añadir vistas al esquema.
  - Añadir rutinas, que es el código de programación que veremos en los siguientes apartados y que hemos estado mencionando a lo largo de este capítulo.
  - Añadir grupos de usuarios con roles, aunque esta última opción no está aún muy conseguida.

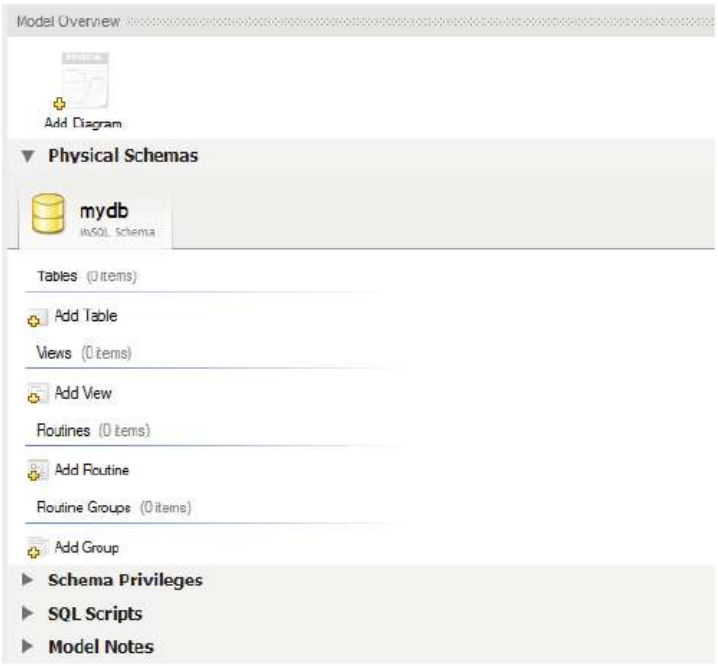


Figura 8.33. *Acciones posibles de la pestaña "Physical Schemas"*

- **Schema Privileges:** para definir los diferentes usuarios que van a operar sobre la base de datos y definir sus roles para con esta.

Figura 8.34. *Acciones posibles de la pestaña "Schema Privileges"*

- **SQL Scripts:** opción con la que podremos añadir y, con ayuda de su editor, escribir *script* sobre la base de datos.

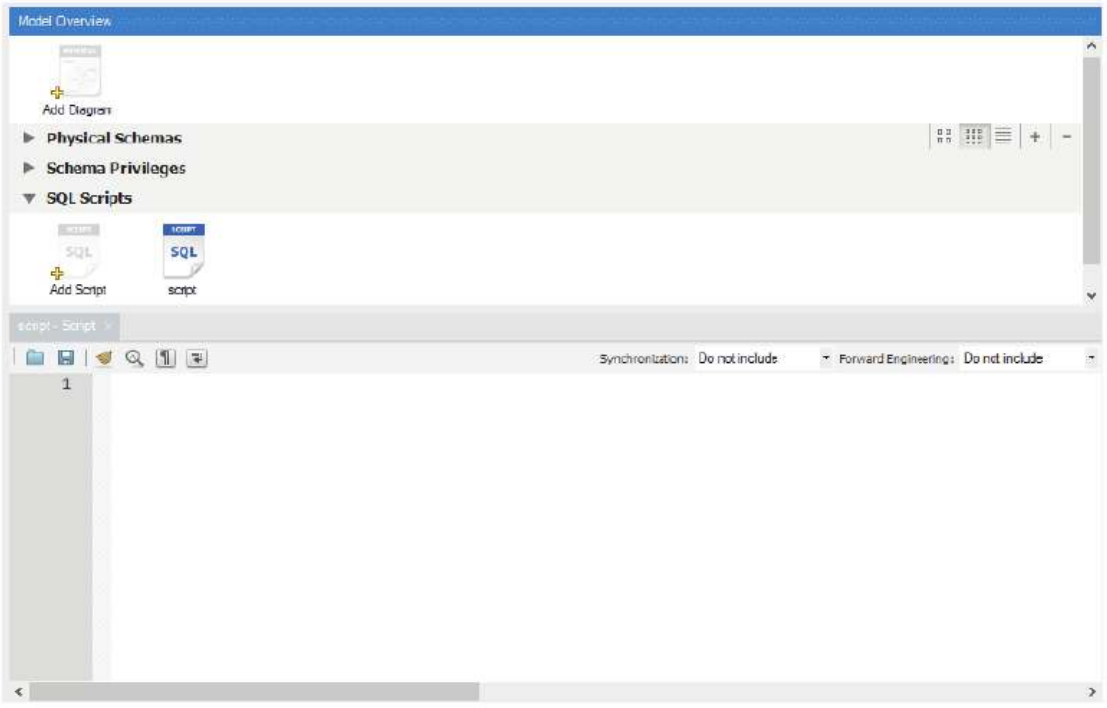


Figura 8.35. *Acciones posibles de la pestaña "SQL Script"*

- **Model Notes:** herramienta para crear y editar notas sobre el esquema creado.

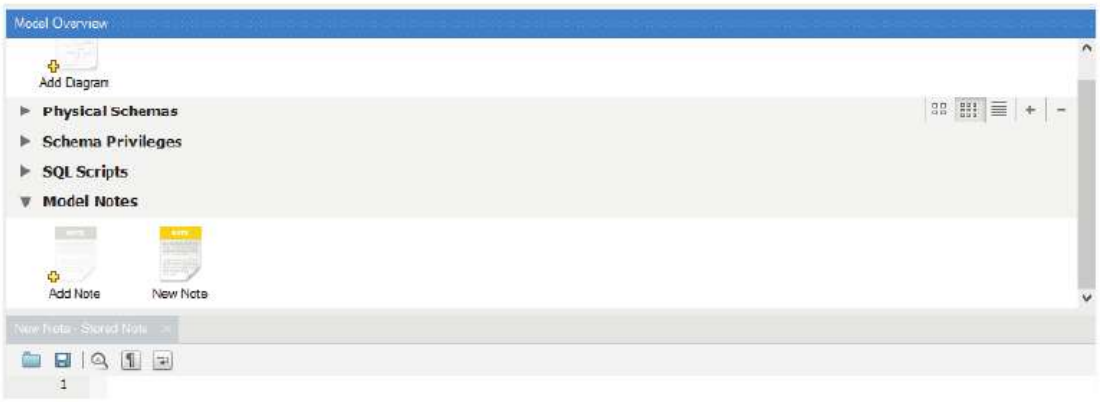


Figura 8.36. *Acciones posibles de la pestaña "Model Notes"*

Junto a estos grupos de acciones, no podemos olvidar un acceso directo denominado *Add Diagram* con el que crearemos el modelado Entidad-Relación de nuestra base de datos, quizás uno de los objetivos fundamentales de muchos programadores.

Definir tablas en la base de datos

El proceso de creación de una tabla comienza pulsando el botón *Add Table* y, tal como vemos en la Figura 8.37, a continuación, escribiremos el nombre la tabla e iremos añadiendo las columnas de dicha tabla junto a sus restricciones, tipo de datos, etc. En la parte inferior izquierda de la ventana, podemos consultas los tipos de datos a emplear.

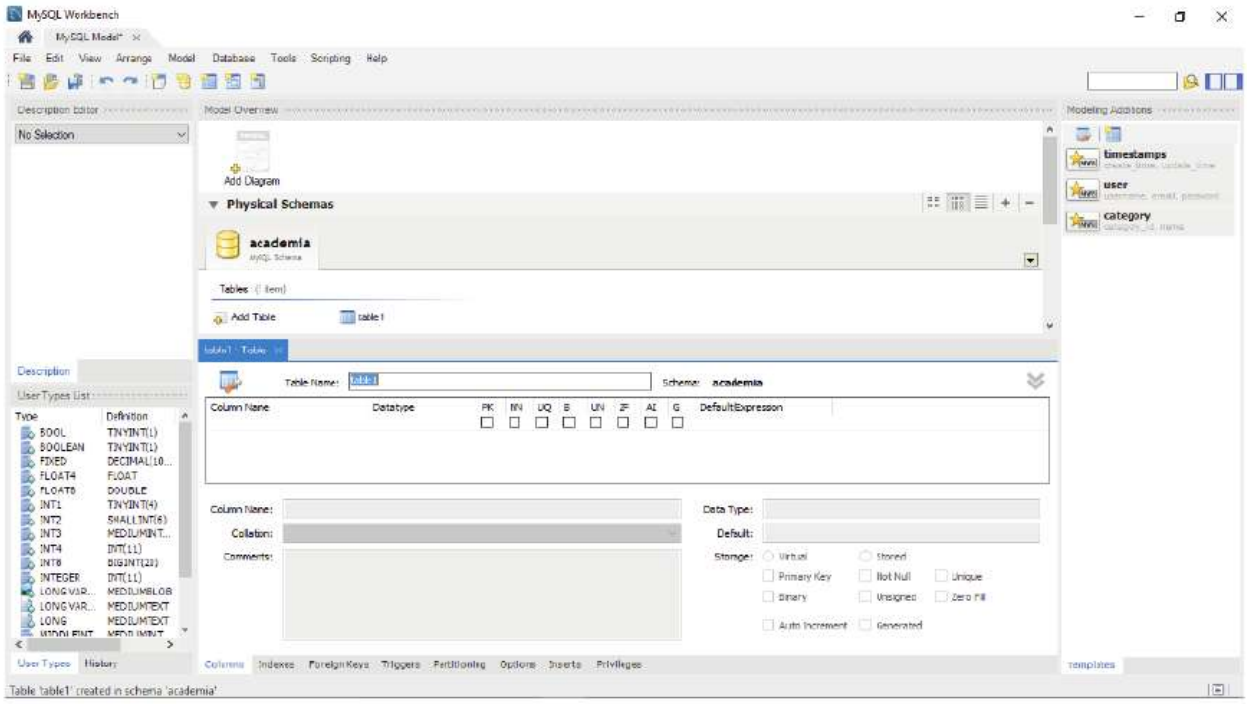


Figura 8.37. *Proceso de creación de una nueva tabla*

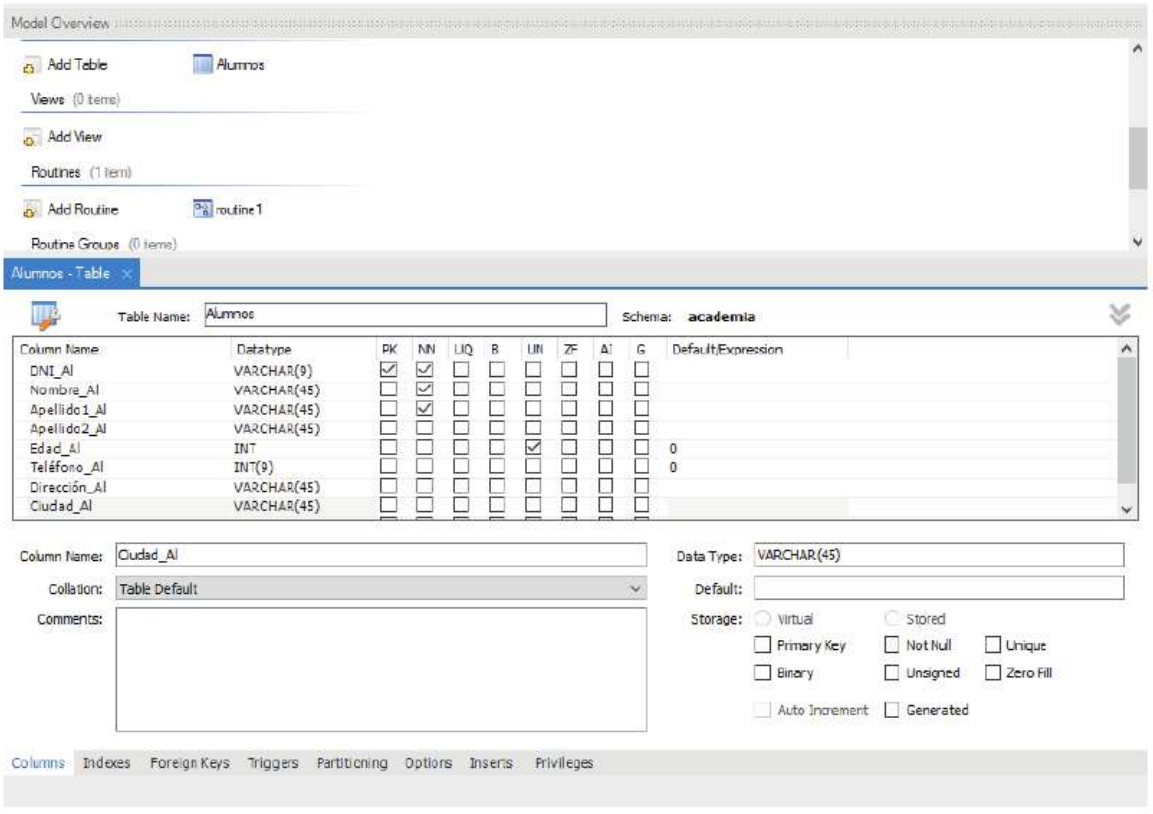
EJEMPLO 8.1

Siguiendo las especificaciones del diagrama Entidad-Relación del Ejemplo 3.1 y su paso a tablas, vamos a crear la tabla *Alumnos*.

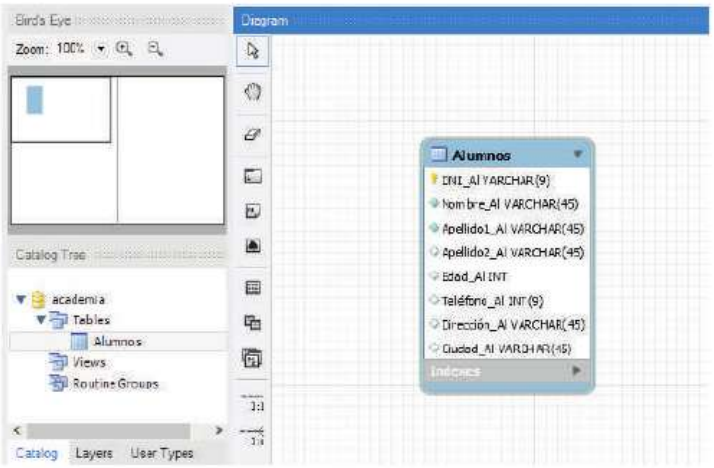
Tal y como hemos comentado en la Figura 8.37, añadiremos una nueva tabla, la renombraremos como se indica y le añadiremos las diferentes columnas, especificando su nombre, tipo de datos a almacenar y sus restricciones.

En la imagen inferior, podemos ver una vista del modo de diseño de la tabla y la definición de las siguientes columnas:

- **DNI_Al:** cadena de 9 caracteres (VARCHAR), clave primaria (PK) y no acepta el valor *Null* (NN).
- **Nombre_Al:** Cadena de 45 caracteres (VARCHAR) y no acepta el valor *Null* (NN).
- **Apellido1_Al:** cadena de 45 caracteres (VARCHAR) y no acepta el valor *Null* (NN).
- **Apellido2_Al:** cadena de 45 caracteres (VARCHAR) y si acepta el valor *Null*, ya que los nombres extranjeros no tienen segundo apellido (al menos algunos).
- **Edad_Al:** entero (INT) y sin signo (UN), con valor por defecto "0".
- **Teléfono_Al:** entero (INT) limitado a 9 dígitos y con valor por defecto "0".
- **Dirección_Al** y **Ciudad_Al:** cadenas de 45 caracteres como máximo.

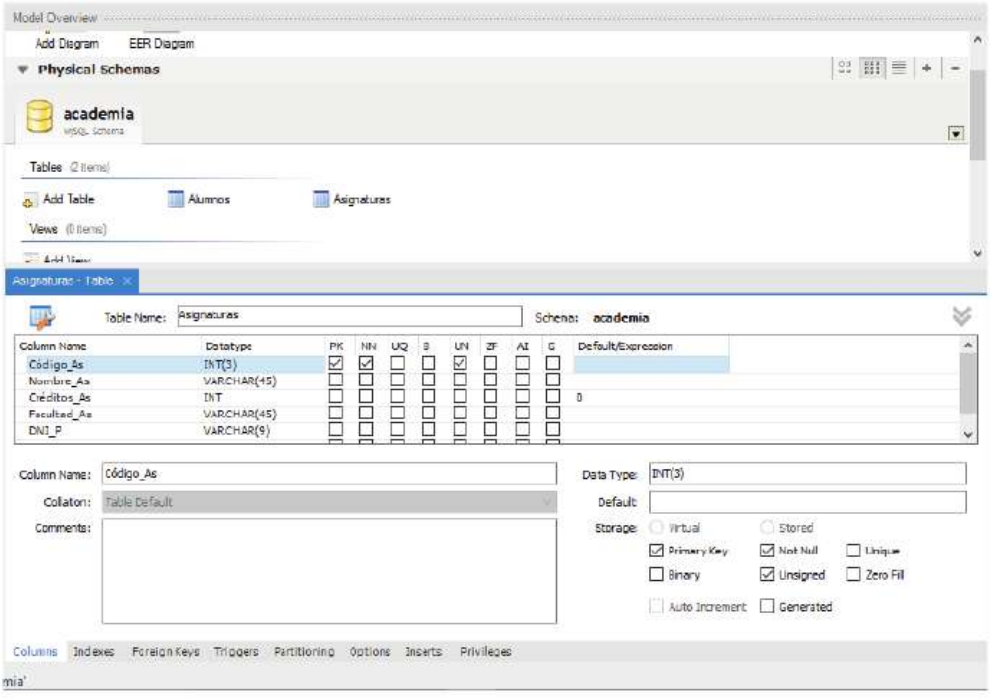


De esta forma, cuando queramos elaborar el diagrama Entidad-Relación, ya dispondremos de esta tabla:

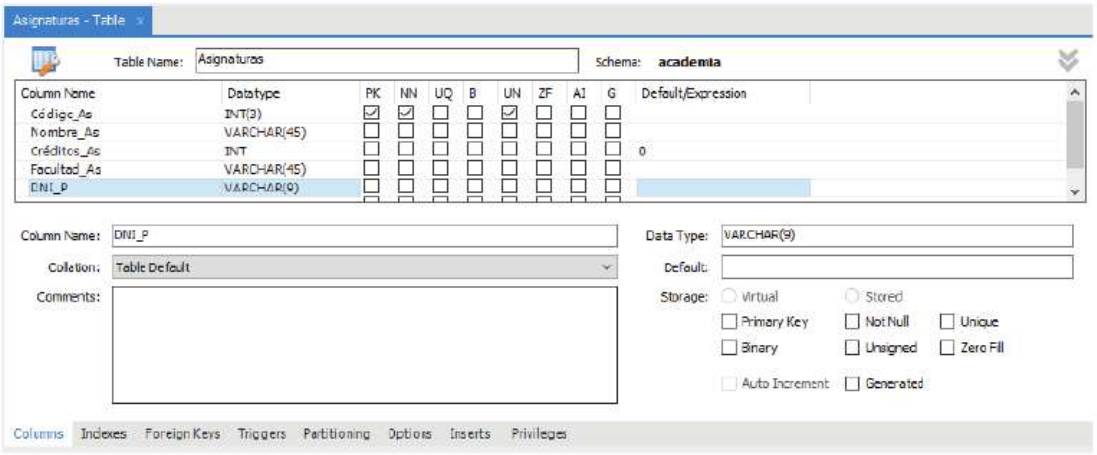


EJEMPLO 8.2

Siguiendo nuevamente las especificaciones del diagrama Entidad-Relación del Ejemplo 3.1 y su paso a tablas, vamos a crear la tabla *Asignaturas*.



En este caso, dado que la tabla contiene la columna *DNI_P*, que es una clave foránea de la clave primaria de la tabla *Profesores*, tenemos que indicar este hecho en la pestaña *"Foreign Keys"* situada en la parte inferior del cuadro de diálogo (cuando tengamos seleccionada la columna afectada).



No obstante, dado que esa columna hace referencia a una tabla que aún no hemos creado no podremos completar esta tarea hasta no haber dado de alta la tabla *Profesores*.

Definir relaciones en la base de datos

Las relaciones se definen en el editor denominado *EER Diagram*. Para ello, en primer lugar tendremos que presentar en el editor las tablas involucradas en dicho diagrama (concretamente las que queramos relacionar) ayudándonos del listado de elementos que vemos en el margen izquierdo de la MySQL Workbench. Posteriormente, ayudándonos de los tipos de relaciones situados a la izquierda del editor, escogeremos la más adecuada y marcaremos las tablas que involucra dicha relación.

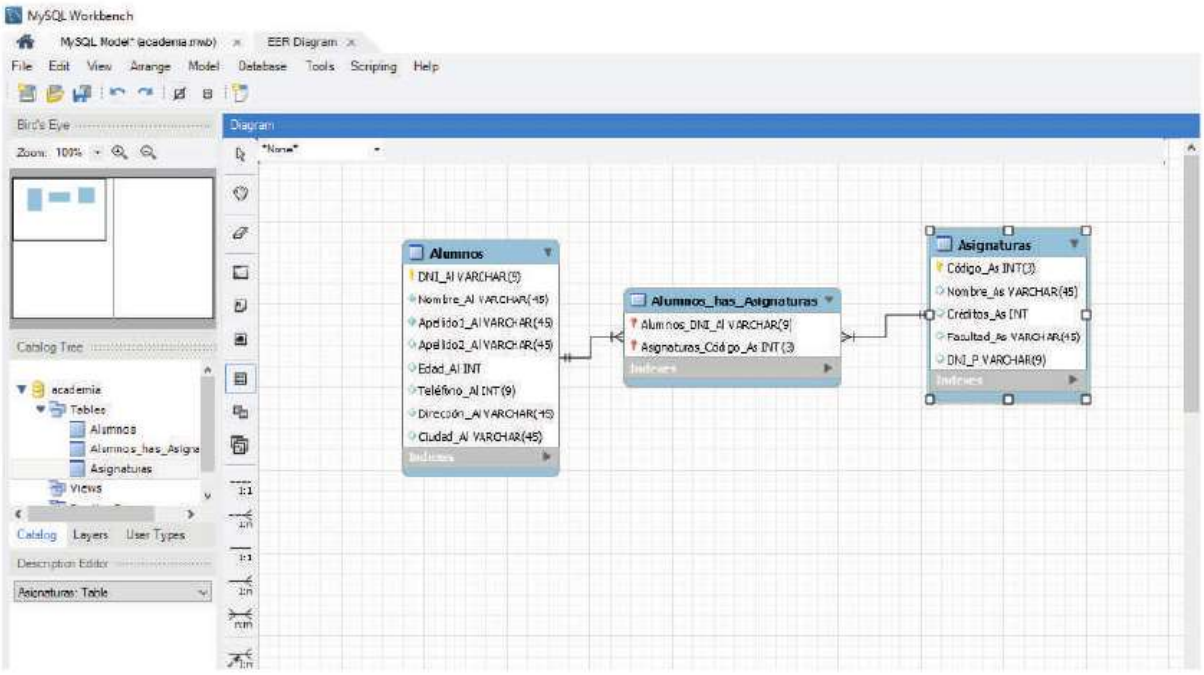


Figura 8.38. *Proceso de creación de una relación entre dos tablas*

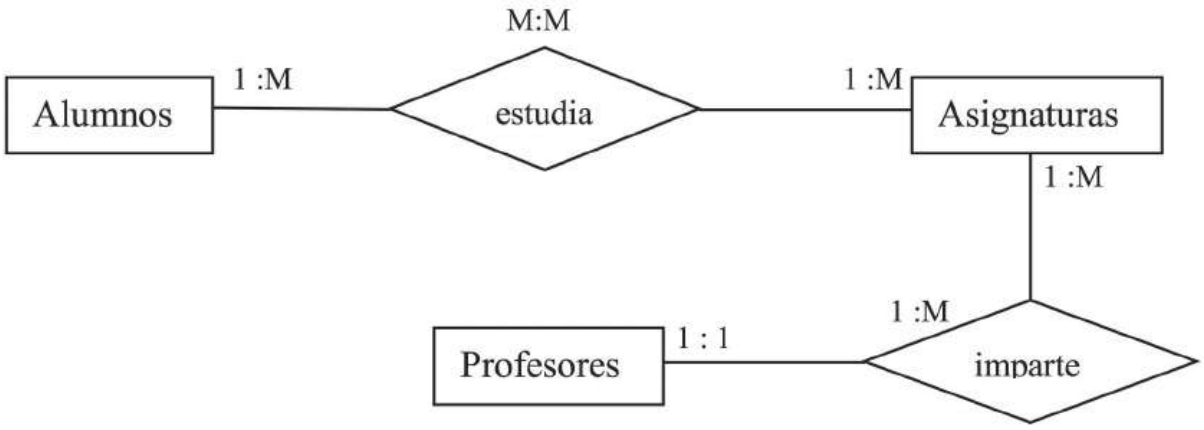
Es necesario aclarar que, aunque coloquialmente denominamos a este diagrama como "diagrama Entidad-Relación", en este programa no estamos haciendo un diseño conceptual de la base de datos, sino que estamos diseñando la estructura definitiva de la misma.

Esta es la razón por la que, aunque previamente disponíamos de dos tablas, al realizar la relación *N:M* (de muchos a muchos), *MySQL Workbench* ha creado una nueva tabla, ya que, como estudiamos en capítulos anteriores, las relaciones de muchos a muchos pasan a tablas, siendo la clave de la misma, las claves primarias de ambas tablas relacionadas. Así lo ha llevado a cabo el editor, como vemos en la Figura 8.38.

Del mismo modo, en la Figura 8.38, podemos ver todo los tipos de relaciones existentes.

EJERCICIO GUIADO 8.1

Tomando como referencia el diseño conceptual y lógico del Ejemplo 3.1 que seguidamente mostramos.



Alumnos (DNI_Al, Nombre_Al, Apellido1_Al, Apellido2_Al,Edad_Al, Teléfono_Al, Dirección_Al, Ciudad_Al)

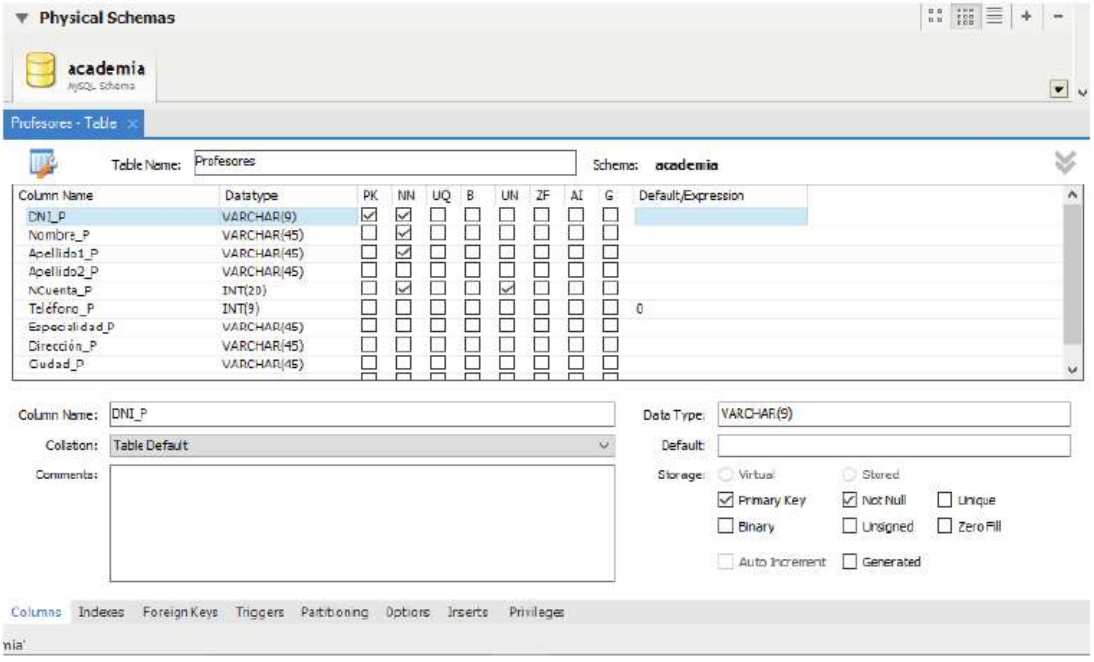
Estudian (DNI_Al, Cód_As, Nota_Al_As, Convocatoria_Al_As)

Asignaturas (Cód_As, Nombre_As, Créditos_As, Facultad_As, DNI_P)

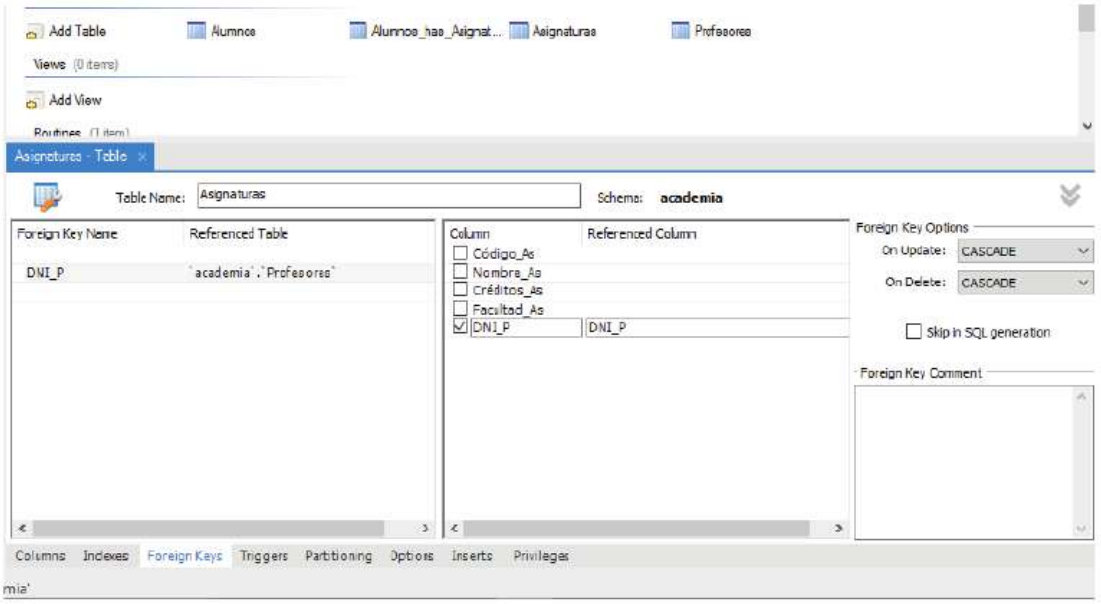
Profesores (DNI_P, Nombre_P, Apellido1_P, Apellido2_P, NCuenta_P, Teléfono_P, Especialidad_P, Dirección_P, Ciudad_P)

Vamos a crear, con ayuda de MySQL Workbench, la base de datos resultante, partiendo de los elementos ya creados en el Ejemplo 8.1 y el Ejemplo 8.2:

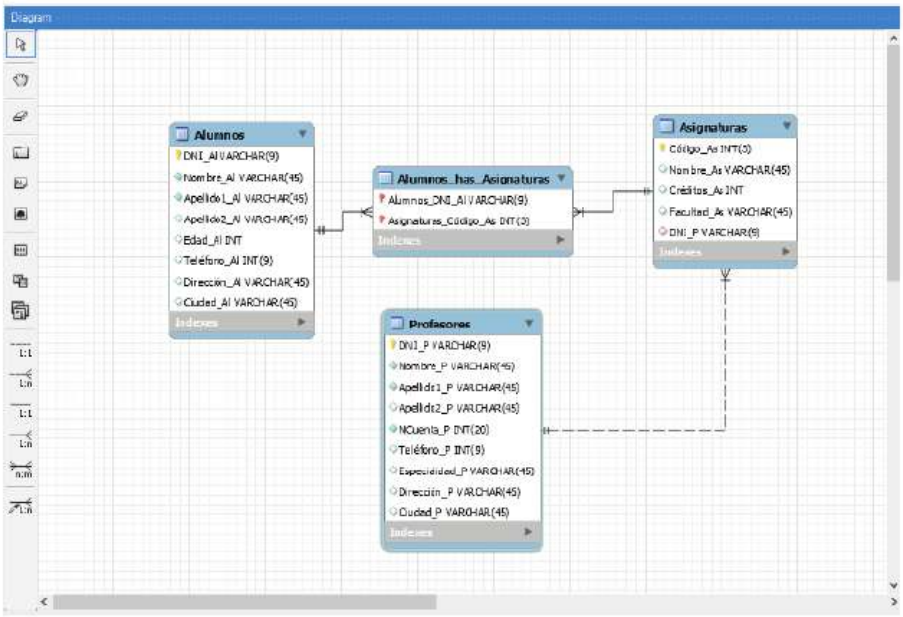
Comenzaremos creando la tabla restante, es decir, la tabla *Profesores:*



Una vez dado este paso, ya podemos terminar la tabla *Asignaturas* con el aspecto que nos faltó en el Ejemplo 8.2, esto es, con la especificación de la clave foránea:



Y, por último, tenemos que crear la relación entre las tablas *Asignaturas* y *Profesores*. Dado que ya hemos especificado la clave foránea, nada más incluir la nueva tabla, se genera automáticamente la relación.



Exportar el modelo de la base de datos a código SQL

Por otra parte, si el único objetivo para usar MySQL Workbench es poder definir bases de datos con una herramienta gráfica, para poder exportar el modelo a SQL (con el objetivo de embeber dicho código SQL y trabajar con un IDE y lenguaje de programación no orientado a bases de datos en exclusividad), tenemos que exportar la base de datos creada con la opción que nos proporciona MySQL Workbench.

Para ello, seleccionaremos el botón *File → Export → Forward Enginer SQL CREATE Script…*

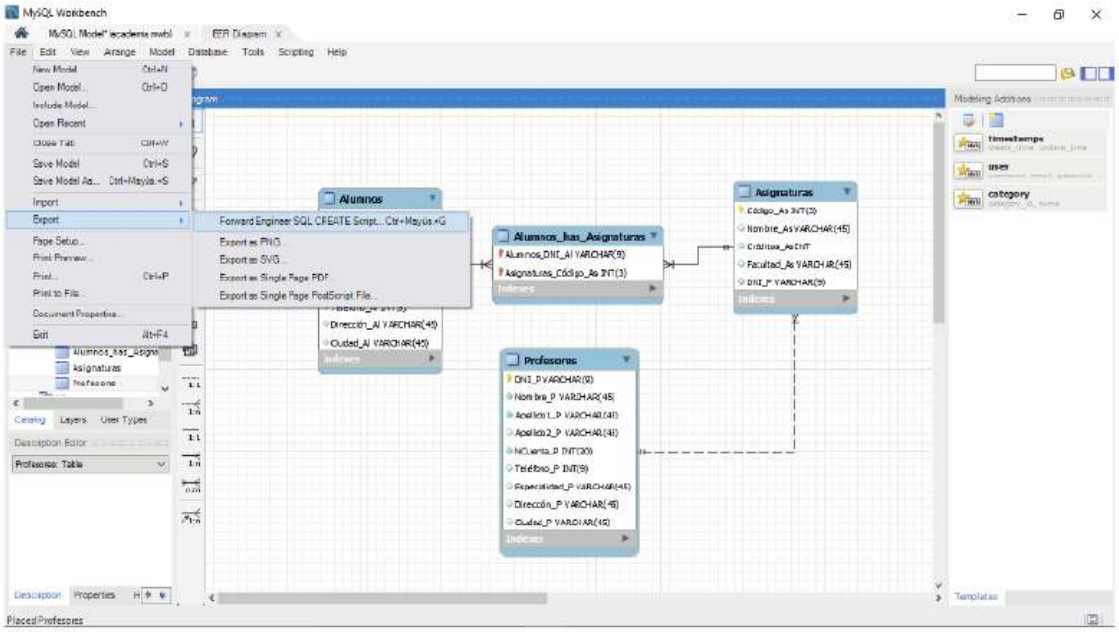


Figura 8.39. *Vista de la ruta para exportar una base de datos a un Script SQL*

Seleccionada esta opción, aparecerá un cuadro de diálogo para definir los parámetros del proceso de exportación de la base de datos.

En primer lugar, se mostrará un cuadro de diálogo donde estableceremos el nombre del archivo *sql* que contendrá la base de datos y las diferentes alternativas sobre los elementos a exportar.

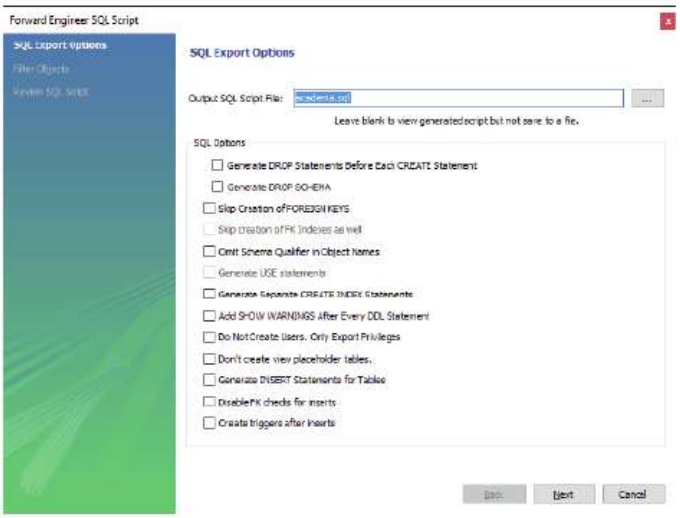


Figura 8.40. *Primer paso para la exportación de un modelo de base de datos a un Script SQL*

Tras esto, debemos seleccionar qué elementos del modelo de base de datos queremos exportar, escogiendo entre:

- Tablas y relaciones
- Vistas
- Rutinas (programas escritos en MySQL)
- *Triggers*
- Usuarios con sus correspondientes privilegios

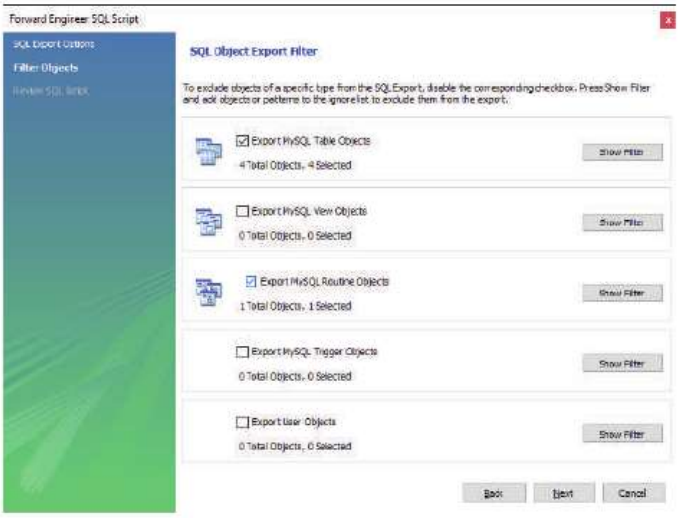


Figura 8.41. *Segundo paso para la exportación de un modelo de base de datos a un Script SQL*

Y, por último, aparecerá un último cuadro de diálogo en el que se mostrará el código SQL correspondiente al modelo que hemos exportado, según la configuración del proceso de exportación que hayamos definido en los dos pasos anteriores.

En esta última ventana podremos:

- Editar, copiar, cortar y pegar en otro lugar el código anteriormente comentado.
- Guardar el código en otro archivo.
- Copiar el código SQL en el portapapeles para, posteriormente, pegarlo en algún otro editor.

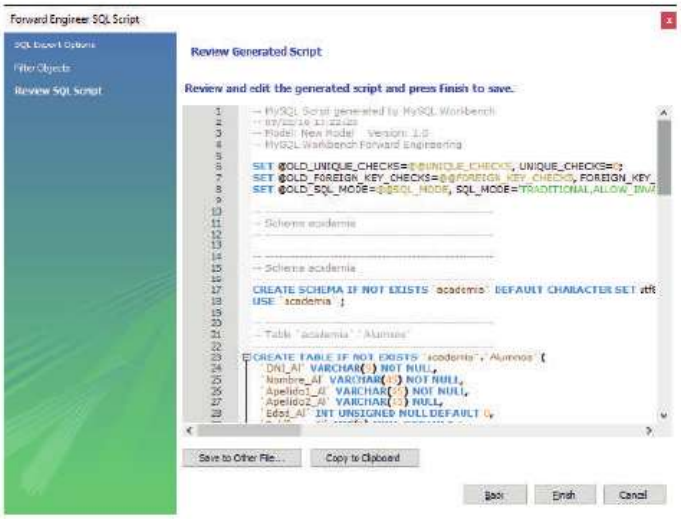
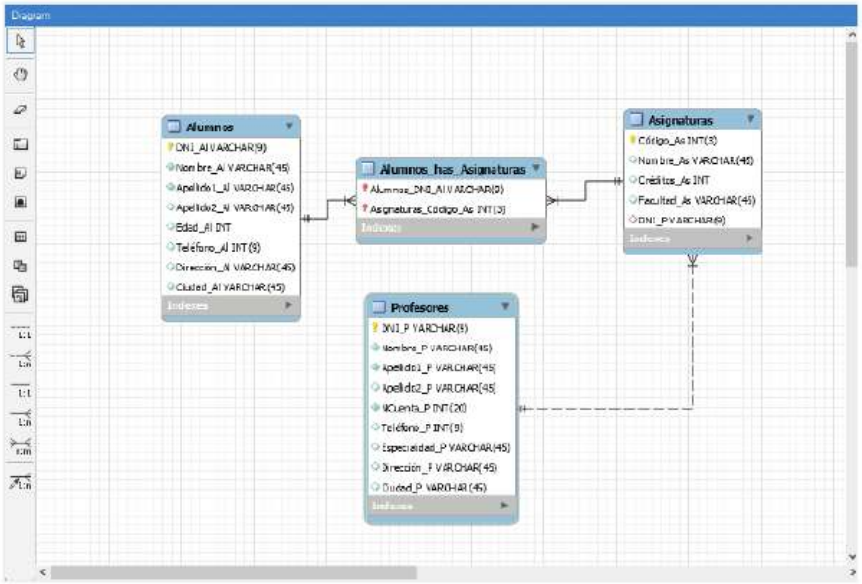


Figura 8.42. *Tercer y último paso para la exportación de un modelo de base de datos a un Script SQL*

EJERCICIO GUIADO 8.2

Tomando como referencia el modelo de base de datos que generamos en el Ejercicio guiado 8.1 y que, a continuación, se muestra:



Generemos el Script SQL resultante de exportar el anterior modelo:

- -- MySQL Script generated by MySQL Workbench
- -- 09/25/16 13:23:52
- -- Model: New Model Version: 1.0
- -- MySQL Workbench Forward Engineering

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS,
FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE,
SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';
-- -----------------------------------------------------
-- Schema academia
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema academia
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `academia` DEFAULT CHARACTER SET utf8 ;
USE `academia` ;

-- -----------------------------------------------------
-- Table `academia`.`Alumnos`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `academia`.`Alumnos` (
`DNI_Al` VARCHAR(9) NOT NULL,
  `Nombre_Al` VARCHAR(45) NOT NULL,
  `Apellido1_Al` VARCHAR(45) NOT NULL,
  `Apellido2_Al` VARCHAR(45) NULL,
  `Edad_Al` INT UNSIGNED NULL DEFAULT 0,
  `Teléfono_Al` INT(9) NULL DEFAULT 0,
  `Dirección_Al` VARCHAR(45) NULL,
  `Ciudad_Al` VARCHAR(45) NULL,
PRIMARY KEY (`DNI_Al`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `academia`.`Profesores`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `academia`.`Profesores` (
  `DNI_P` VARCHAR(9) NOT NULL,
`Nombre_P` VARCHAR(45) NOT NULL,
  `Apellido1_P` VARCHAR(45) NOT NULL,
  `Apellido2_P` VARCHAR(45) NULL,
`NCuenta_P` INT(20) UNSIGNED NOT NULL,
  `Teléfono_P` INT(9) NULL DEFAULT 0,
`Especialidad_P` VARCHAR(45) NULL,
```

```
  `Dirección_P` VARCHAR(45) NULL,
`Ciudad_P` VARCHAR(45) NULL,
  PRIMARY KEY (`DNI_P`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `academia`.`Asignaturas`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `academia`.`Asignaturas` (
  `Código_As` INT(3) UNSIGNED NOT NULL,
  `Nombre_As` VARCHAR(45) NULL,
  `Créditos_As` INT NULL DEFAULT 0,
  `Facultad_As` VARCHAR(45) NULL,
  `DNI_P` VARCHAR(9) NULL,
  PRIMARY KEY (`Código_As`),
  INDEX `DNI_P_idx` (`DNI_P` ASC),
  CONSTRAINT `DNI_P`
    FOREIGN KEY (`DNI_P`)
    REFERENCES `academia`.`Profesores` (`DNI_P`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `academia`.`Alumnos_has_Asignaturas`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `academia`.`Alumnos_has_Asignaturas` (
`Alumnos_DNI_Al` VARCHAR(9) NOT NULL,
`Asignaturas_Código_As` INT(3) UNSIGNED NOT NULL,
PRIMARY KEY (`Alumnos_DNI_Al`, `Asignaturas_Código_As`),
  INDEX `fk_Alumnos_has_Asignaturas_Asignaturas1_idx` (`Asignaturas_Código_As` ASC),
  INDEX `fk_Alumnos_has_Asignaturas_Alumnos1_idx` (`Alumnos_DNI_Al` ASC),
  CONSTRAINT `fk_Alumnos_has_Asignaturas_Alumnos1`
    FOREIGN KEY (`Alumnos_DNI_Al`)
    REFERENCES `academia`.`Alumnos` (`DNI_Al`)
ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_Alumnos_has_Asignaturas_Asignaturas1`
    FOREIGN KEY (`Asignaturas_Código_As`)
REFERENCES `academia`.`Asignaturas` (`Código_As`)
ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

Utilizar el editor para escribir rutinas SQL

Una vez pulsemos en la opción para crear nuevos archivos SQL, tendremos que dar de alta procedimientos y lanzarlos, tal y como vemos en las siguientes figuras.

Nótese que para el uso y lanzamientos de estos Scripts SQL tendremos que crear una conexión.

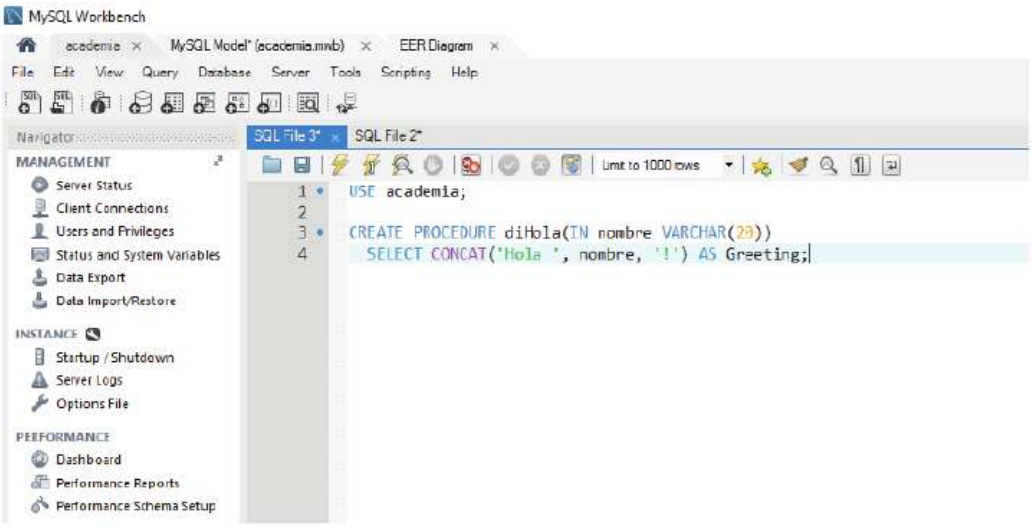


Figura 8.43. *Ejemplo de código MySQL*

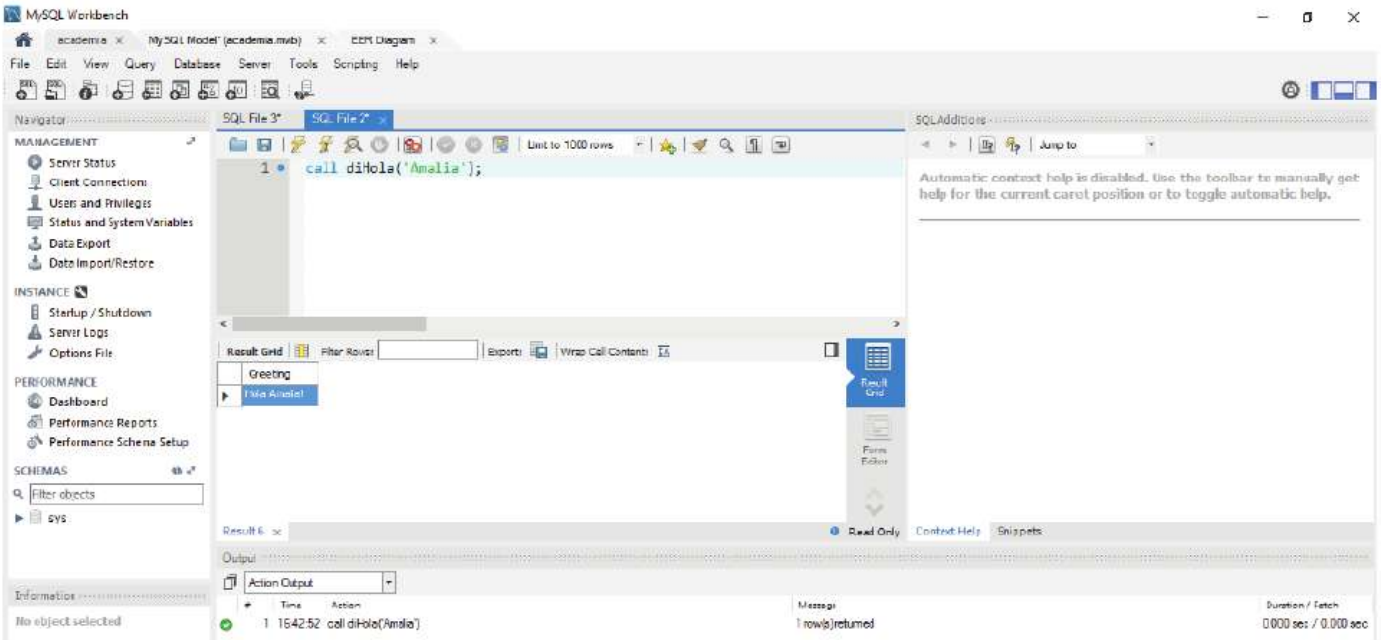


Figura 8.44. *Lanzamiento del ejemplo de código MySQL anterior*

INFORMACIÓN

Para adquirir un conocimiento más ampliado de este IDE de desarrollo orientado a bases de datos, os recomendamos el manual de referencia oficial que nos ofrece la página web de Oracle: *http://dev.mysql.com/doc/workbench/en/*

8.3 LA SINTAXIS DEL LENGUAJE DE PROGRAMACIÓN

8.3.1 VARIABLES

Una variable, tanto en SQL como en cualquier lenguaje de programación, es un espacio de memoria donde se almacena información de un determinado tipo (que se debe especificar en el momento de la declaración de la citada variable). El uso de variables es muy importante a la hora de trabajar con procedimientos y funciones, ya que pueden almacenar valores intermedios entre las diferentes instrucciones de la subrutina.

En el caso de las variables de SQL se consideran que son variables locales, es decir, que son eliminadas en el momento en el que finaliza el procedimiento o la función donde han sido declaradas.

A continuación, estudiaremos las principales operaciones que se pueden llevar a cabo con una variable.

Declaración

Se trata de la operación fundamental, ya que es necesario declarar una variable para, posteriormente, poder emplearla en cualquier otra instrucción.

Para declarar una variable se empleará la palabra reservada DECLARE, siguiendo la siguiente *sintaxis:*

DECLARE nombre_variable tipo_variable;

EJEMPLO 8.3

En este ejemplo vamos a declara una variable denominada *nombre_Al* con un tipo de datos adecuado:

```
DECLARE nombre_Al VARCHAR(45);
```

Inicialización

Esta operación consiste en almacenar un valor inicial a la variable, probablemente con la intención de usar dicho valor al comienzo del código de programación pero, posteriormente, modificar dicho valor almacenado.

Para la inicialización se empleará la palabra reservada SET, con la siguiente *sintaxis*:

```
SET nombre_variable = valor;
```

EJEMPLO 8.4

Vamos a declarar una variable para almacenar la edad de un alumno y la vamos a inicializar a 15 años:

```
DECLARE Edad_Al INT;
SET Edad_Al = 15;
```

Inicialización y declaración de una variable en una sola línea

Las dos operaciones anteriores se pueden combinar en una sola instrucción, como en prácticamente todos los lenguajes (como Java, C, C++, etc.).

Para ello seguiremos la siguiente *sintaxis*:

```
DECLARE nombre_variable DEFAULT valor;
```

EJEMPLO 8.5

Tomando como referencia el Ejemplo 8.4, en este ejemplo escribiremos una sola instrucción para el mismo objetivo:

```
DECLARE Edad_Al DEFAULT 15;
```

Imprimir el contenido de una variable

Para llevar a cabo dicha operación se empleará la palabra reservada SELECT, con la siguiente *sintaxis*:

```
SELECT nombre_variable;
```

EJEMPLO 8.6

Vamos a mostrar los contenidos de las variables declaradas en el Ejemplo 8.3 y 8.4.

```
SELECT Nombre_Al;
SELECT Edad_Al;
```

Asignar el resultado de una consulta en una variable

Esta operación consiste en almacenar el resultado que se devuelve de una consulta (en caso de que dicha consulta devuelva un solo registro como resultado) en una variable.

Para llevar a cabo dicha operación se empleará la palabra reservada INTO. Explicaremos la *sintaxis* mediante el siguiente ejemplo:

EJEMPLO 8.7

Vamos a almacenar la edad del alumno llamado "Francisco Javier Martínez López" en una variable que tendremos que declarar. Finalmente mostraremos el resultado por pantalla.

```
DECLARE Edad_Fran INT;
SELECT Edad_Al INTO Edad_Fran FROM Alumnos WHERE Nombre_Al = "Francisco
Javier" AND Apellido1_Al = "Martínez" AND Apellido2_Al = "López";
SELECT Edad_Fran;
```

8.3.2 TIPOS DE DATOS

INFORMACIÓN

La información para poder desarrollar esta sección la hemos obtenido el manual de referencia que nos ofrece MySQL, en este caso, el de su versión5.6. Os recomendamos su consulta para cualquier duda acerca del lenguaje: *https://dev.mysql.com/doc/refman/5.6/en/*.

Los principales **tipos de datos numéricos** de MySQL son los siguientes:

Tipo	Descripción
BIT[(longitud)]	Campo numérico de *longitud* bits. El valor de *longitud* está comprendido entre 1 y 64 (por defecto 1, si *longitud* está omitida)
TINYINT[(longitud)] [UNSIGNED] [ZEROFILL]	Campo entero muy pequeño, comprendido entre -128 y 127 (con signo) o entre 0 y 255 (sin signo). El parámetro *ZEROFILL*indica que se debe completar el campo con ceros a la izquierda hasta su longitud máxima
BOOL, BOOLEAN	Equivale a *TINYINT(1)*. Se usa como valores booleanos (0 equivale a *false*, 1 equivale a *true*)
SMALLINT[(longitud)] [UNSIGNED] [ZEROFILL]	Campo entero pequeño, comprendido entre -32.768 y 32.767 (con signo) o entre 0 y 65.535 (sin signo). El parámetro *ZEROFILL*indica que se debe completar el campo con ceros a la izquierda hasta su longitud máxima
MEDIUMINT[(longitud)] [UNSIGNED] [ZEROFILL]	Campo entero mediano, comprendido entre -8.388.608 y 8.388.607 (con signo) o entre 0 y 16.777.215 (sin signo). El parámetro *ZEROFILL*indica que se debe completar el campo con ceros a la izquierda hasta su longitud máxima
INT[(longitud)] [UNSIGNED] [ZEROFILL]	Campo entero estándar, comprendido entre -2.147.483.648 y 2.147.483.647 (con signo) o entre 0 y 4.294.967.295 (sin signo). El parámetro *ZEROFILL* indica que se debe completar el campo con ceros a la izquierda hasta su longitud máxima
INTEGER[(longitud)] [UNSIGNED] [ZEROFILL]	Sinónimo de *INT*.
BIGINT[(longitud)] [UNSIGNED] [ZEROFILL]	Campo entero grande, comprendido entre -9.223.372.036.854.775.808 y 9.223.372.036.854.775.807 (con signo) o entre 0 y 18.446.744.073.709.551.615 (sin signo). El parámetro *ZEROFILL* indica que se debe completar el campo con ceros a la izquierda hasta su longitud máxima
DECIMAL [(longitud [,decimales])] [UNSIGNED] [ZEROFILL]	Número con coma flotante con *longitud* dígitos en la parte entera (valor máximo de 65 dígitos, por defecto 10) y *decimales* dígitos tras la coma decimal (valor máximo de 30 dígitos, por defecto 0)
DEC [(longitud [,decimales])] [UNSIGNED] [ZEROFILL] **NUMERIC [(longitud [,decimales])] [UNSIGNED] [ZEROFILL]** **FIXED [(longitud [,decimales])] [UNSIGNED] [ZEROFILL]**	Sinónimos de *DECIMAL*
FLOAT [(longitud,decimales)] [UNSIGNED] [ZEROFILL]	Número en coma flotante (simple precisión). El rango de valores permitidos es: De -3,402823466 x 10^{38} hasta -1,175494351 x 10^{-38} 0 De 1,175494351 x 10^{-38} hasta 3,402823466 x 10^{38}
DOUBLE [(longitud,decimales)] [UNSIGNED] [ZEROFILL]	Número en coma flotante (doble precisión). El rango de valores permitidos es: De -1,7976931348623157 x 10^{308} hasta -2,2250738585072014 x 10^{-308} 0 De 2,2250738585072014 x 10^{-308} hasta 1,7976931348623157 x 10^{308}

DOUBLE PRECISION [(longitud,decimales)] [UNSIGNED] [ZEROFILL] **REAL [(longitud,decimales)] [UNSIGNED] [ZEROFILL]**	Sinónimos de *DOUBLE*
FLOAT (longitud) [UNSIGNED] [ZEROFILL]	Número con coma flotante. *Longitud* representa la precisión (puede ir de 0 a 24 para números de coma flotante de precisión sencilla y de 25 a 53 para números de coma flotante con doble precisión. Estos tipos son como los tipos *FLOAT* y *DOUBLE* descritos anteriormente. *FLOAT(p)* tiene el mismo rango que los tipos correspondientes *FLOAT* y *DOUBLE*, pero la anchura de muestra y el número de decimales no están definidos.

Los principales **tipos de datos relacionados con fechas y horas** de MySQL son los siguientes:

Tipo	Descripción
DATE	Fecha bajo el formato 'YYYY-MM-DD'
DATETIME[(precisión)]	Fecha y hora bajo el formato 'YYYY-MM-DD HH:MM:SS.SSSSSS'. El parámetro *precisión* define la precisión de la parte decimal de los segundos
TIMESTAMP(precisión)	*precisión* representa el número de dígitos que se utilizarán para representar un valor de fecha y hora comprendido desde el inicio de 1970 hasta algún momento del año 2037
TIME[(precisión)]	Hora, bajo el formato 'HH:MM:SS[.*precisión* valores de S]'
YEAR[(2|4)]	Tipo que representa el año, con cuatro dígitos (por defecto), es decir 'YYYY' o con dos dígitos 'YY'

Los principales **tipos de datos relacionados con cadenas de caracteres** de MySQL son los siguientes:

Tipo	Descripción
CHAR[(longitud)]	Admite caracteres alfanuméricos. La *longitud* varía entre 1 y 255 caracteres (por defecto 1). Si se le asigna una cadena de longitud inferior, se completará con espacios
VACHAR(longitud)	Muy similar al *CHAR* salvo por la obligatoriedad de especificar la *longitud* y porque no se autorrellena con espacios
TEXT, TINYTEXT, MEDIUMTEXT, LONGTEXT	En todos los tipos se admiten cadenas alfanuméricas de longitud variable, diferenciándose en su capacidad: TINYTEXT: hasta 255 caracteres TEXT: hasta 65.535 caracteres MEDIUMTEXT: hasta 16.777.215 caracteres LONGTEXT: hasta 4.294.967.295 caracteres

Los principales **tipos de datos binarios** de MySQL son los siguientes:

Tipo	Descripción
BLOB	Objeto binario que puede almacenar cualquier tipo de información, desde un archivo de texto con su formato (a diferencia del tipo *TEXT*), hasta imágenes, archivos de audio o video, etc. Admite hasta 65.535 caracteres
TINYBLOB, MEDIUMBLOB, LONGBLOB	Similares al tipo *BLOB* pero con los siguientes tamaños: TINYBLOB: hasta 255 caracteres MEDIUMBLOB: hasta 16.777.215 caracteres LONGBLOB: hasta 4.294.967.295 caracteres

8.3.3 ESTRUCTURAS DE CONTROL

Al igual que los lenguajes de programación más extendidos, tales como Java, C, Delphi, etc., MySQL también dispone de estructuras de control; son las sentencias condicionales y las sentencias repetitivas o bucles.

Sentencias condicionales

Las sentencias condicionales son estructuras predefinidas por un lenguaje de programación para permitir que una instrucción o grupo de instrucciones se ejecuten o no en función de una o varias condiciones establecidas en tiempo de programación.

Estas sentencias condicionales pueden ser simples, es decir, si se cumple una condición o condiciones se ejecutan las instrucciones y, en caso de no cumplimiento, no se ejecutan, o múltiples. En las sentencias condicionales múltiples, se pueden ir seleccionando bloques de código que se ejecuten uno u otro en función de condiciones.

Del mismo modo, las sentencias condicionales se pueden encadenar entre sí. Lo veremos en algunos ejemplos en este apartado.

Las sentencias condicionales más representativas en MySQL son *IF* Y *CASE* (más conocido como *SWITCH* en otros lenguajes).

Sentencia condicional IF

Sintaxis:

IF *condición* THEN
    Instrucciones;
END IF;

También puede presentar una doble alternativa, mediante el uso de la palabra reservada *ELSE*, siguiendo la siguiente *sintaxis*:

IF *condición* THEN
    $Instrucciones_1$*;*
ELSE
    $Instrucciones_2$;
END IF;

EJEMPLO 8.8

Vamos a crear un procedimiento para que devuelva como salida si el alumno con DNI *75248143D* está "aprobado" o "suspenso" en la asignatura denominada "Programación":

```
CREATE PROCEDURE verNota()
BEGIN
DECLARE notaAlumno INT;
DECLARE salida VARCHAR(8);
SELECT Nota_Al_As INTO notaAlumno FROM Notas, Asignaturas WHERE Notas.Código_As = Asignaturas.
Código_As AND Notas.DNI_Al = '75248143D' AND Asiganturas.Nombre_As = 'Programación';
IF notaAlumno < 5 THEN
    SET salida = 'Suspenso';
ELSE
    SET salida = "Apto";
END IF;
SELECT salida;
END;
```

NOTA

Como antes anticipábamos, es posible encadenar sucesivas estructuras IF, como por ejemplo en la siguiente función: **Dado un número del 1 al 7 como parámetro de entrada, devolver el día de la semana asociado a dicho número.**

```
CREATE FUNCTION diaSemana(num INT) RETURNS VARCHAR(10)
BEGIN
DECLARE dia VARCHAR(10);
IF num== 1 THEN
        SET dia = 'Lunes';
ELSE
        IF num== 2 THEN
                SET dia = 'Martes';
        ELSE

                IF num== 3 THEN

                        SET dia = 'Miércoles';

                ELSE

                        IF num== 4 THEN

                                SET dia = 'Jueves';

                        ELSE
```

```
                    IF num== 5 THEN
                        SET dia = 'Viernes';
                    ELSE
                        IF num== 6 THEN
                            SET dia = 'Sábado';
                        ELSE
                            SET dia = 'Domingo';
                        END IF;
                    END IF;
                END IF;
            END IF;
        END IF;
    END IF;
    RETURN dia;
    END;
```

Sentencia condicional SWITCH

Sintaxis:

CASE *expresión/variable*
WHEN valor$_1$ THEN *Instrucciones$_1$;*
WHEN valor$_2$ THEN *Instrucciones$_2$;*
WHEN valor$_3$ THEN *Instrucciones$_3$;*
…
WHEN valor$_n$ THEN *Instrucciones$_n$;*
[ELSE *Instrucciones$_{n+1}$*;]
END CASE;

EJEMPLO 8.9

Podemos implementar el mismo ejemplo del día de la semana con esta nueva estructura condicional. Tendría la siguiente forma:

```
CREATE FUNCTION diaSemana(num INT) RETURNS VARCHAR(10)
BEGIN
DECLARE dia VARCHAR(10);
```

```
CASE num
    WHEN 1 THEN SET dia = 'Lunes';
    WHEN 2 THEN SET dia = 'Martes';
    WHEN 3 THEN SET dia = 'Miércoles';
    WHEN 4 THEN SET dia = 'Jueves';
    WHEN 5 THEN SET dia = 'Viernes';
    WHEN 6 THEN SET dia = 'Sábado';
    ELSE SET dia = 'Domingo';
END CASE;
RETURN dia;
END;
```

Sentencias repetitivas o bucles

Mediante un bucle, un programador puede definir que un determinado bloque de instrucciones se ejecuten en más de una ocasión (o en ninguna), mediante la definición de una condición de repetición o salida (dependiendo del bucle que se esté utilizando).

Bucle WHILE

Sintaxis:

```
WHILE condición DO
    Instrucciones;
END WHILE;
```

EJEMPLO 8.10

Vamos a generar el factorial de un número dado como parámetro de entrada:

```
CREATE PROCEDURE calcularFactorial(IN num INT)
BEGIN
DECLARE factorial DEFAULT 1;
WHILE num>1 DO
    SET factorial = factorial * num;
    SET num = num - 1;
END WHILE;
SELECT CONCAT('El factorial del número es ', factorial);
END;
```

NOTA

Los bucles pueden contener, a su vez, otros bucles o sentencias condicionales. Para ejemplificar el uso de estas estructuras anidadas poder implementar una función que lleve a cabo la sumatoria de los términos de la ***sucesión ULAM*** a partir de un número inicial introducido como parámetro de entrada.

```
CREATE FUNCTION sucesionULAM(num INT) RETURNS INT
BEGIN
DECLARE ulam DEFAULT 1;
WHILE num > 1 DO
        IF (num%2) == 0 THEN
                SET ulam = ulam + num;

                num = num / 2;
        ELSE
                SET ulam = ulam + num;

                num = num * 3 + 1;
        END IF;
END WHILE;
RETURN ulam;
END;
```

Bucle REPEAT

La diferencia entre los bucles *WHILE* y *REPEAT* radica en que el primer bucle puede no dar ninguna vuelta (es decir, no ejecutar el grupo de instrucciones ninguna vez) mientras que con el bucle *REPEAT* al menos una ejecución de las instrucciones del bucle se ejecutarán.

Sintaxis:

```
REPEAT
    Instrucciones;
UNTIL condición
END REPEAT;
```

EJEMPLO 8.11

Vamos a implementar el Ejemplo 8.10 mediante el bucle *REPEAT*:

```
CREATE PROCEDURE calcularFactorial(IN num INT)
BEGIN
DECLARE factorial DEFAULT 1;
REPEAT
    SET factorial = factorial * num;
    SET num = num - 1;
UNTIL num == 1;
END REPEAT;
SELECT CONCAT('El factorial del número es ', factorial);
END;
```

8.3.4 LIBRERÍAS DE FUNCIONES

MySQL nos ofrece una gran cantidad de funciones relacionadas con diferentes tipos de datos y operaciones sobre la base de datos, por lo que tan solo vamos a comentar las más empleadas.

INFORMACIÓN

La información para poder desarrollar esta sección la hemos obtenido del manual de referencia que nos ofrece MySQL, en este caso, el de su versión 5.6. Os recomendamos su consulta para obtener más información sobre las funciones expuestas en este capítulo y para aprender más funciones además de las aquí expuestas: *https://dev.mysql.com/doc/refman/5.6/en/*.

Las principales **funciones matemáticas** en MySQL son las siguientes:

Función	Operativa
ABS(valor)	Retorna el valor absoluto de *valor*
ACOS(valor)	Retorna el arcocoseno de *valor*. La función retorna *NULL* si *valor* no está en el rango -1 a 1
ASIN(valor)	Retorna el arcoseno de *valor*. La función retorno *NULL* si *valor* no está en el rango -1 a 1
ATAN(valor)	Retorna la arcotangente de *valor*
CEILING(valor), CEIL(valor)	Retorna el entro más pequeño no menor a *valor*. El valor retornado se convierte a *BIGINT*
COS(valor)	Retorna el coseno de *valor* (en radianes)
COT(valor)	Retorna la cotangente de *valor*
DEGREES(valor)	Retorna *valor* convertido de radianes a grados
EXP(valor)	Retorna el valor de e^{valor}
FLOOR(valor)	Retorna el valor entero más grandes pero no mayor a *valor*. El valor retornado se convierte a *BIGINT*
LN(valor), LOG(valor)	Retorna el logaritmo natural (de base *e*) de *valor*
LOG(base,valor)	Retorna el logaritmo en base *base* de *valor*
LOG2(valor)	Retorna el logaritmo en base 2 de *valor*
LOG10(valor)	Retorna el logaritmo en base 10 de *valor*

Función	Operativa
MOD(num1,num2), num1%num2, num1 MOD num2	Retorna el resto de num1 dividido entre num2
PI()	Retorna el valor de PI (por defecto muestra siete decimales)
POW(num1,num2), POWER(num1,num2)	Retorna el valor de num1num2
RADIANS(valor)	Retorna *valor* convertido de grados a radianes
RAND(), RAND(valor)	Retorna un valor aleatorio en coma flotante en el rango de 0 a 10. Si se añade el argumento *valor*, este argumento sirve como "semilla" para la generación del número
ROUND(num1), ROUND(num1,num2)	Retorna el valor de *num1* redondeado al entero más cercano. En caso de tener segundo parámetro, redondea el valor *num1* al decimal *num2*
SIGN(valor)	Retorna el signo del *valor* como -1, 0 o 1, en función de que *valor* sea negativo, cero o positivo
SIN(valor)	Retorna el seno de *valor* dado en radianes
SQRT(valor)	Retorna la raíz cuadrada de un *valor* no negativo
TAN(valor)	Retorna la tangente de *valor* dado en radianes
TRUNCATE(num1,num2)	Retorna el valor de *num1* truncado a *num2* decimales

Las principales **funciones sobre fecha y hora** en MySQL son las siguientes:

Función	Operativa
ADDDATE(fecha, INTERVAL *exp* **tipo_ fecha)**	Retorna una nueva fecha como resultado de sumar a la *fecha* un intervalor *exp* expresado en *tipo_fecha* unidades (días, meses…)
ADDTIME(exp1,exp2)	Añade *exp2* a *exp1* y devuelve el resultado en formato fecha-hora
CURDATE(), CURRENT_ DATE()	Retorna la fecha actual como valor en formato 'YYYY-MM-DD' o YYYYMMDD, en función del contexto
CURTIME(), CURRENT_ TIME()	Retorna la hora actual como valor en formato 'HH:MM:SS' o HHMMSS, en función del contexto
DATE(exp)	Extrae la fecha de la expresión fecha-hora dada por *exp*
DATEIFF(exp1,exp2)	Retorna el número de días entre una fecha inicial y otra final
DAY(fecha), DAYOFMONT(fecha)	Retorna el día del mes (entre 1 y 31)
DAYNAME(fecha)	Retorna el nombre del día de la semana
DAYOFWEEK(fecha)	Retorna el día de la semana (entre 1 y 7)
DAYOFYEAR(date)	Retorna el día del año (entre 1 y 366)
EXTRACT(tipo_fecha FROM fecha)	Extrae parte de *fecha* en función del *tipo_fecha* indicado
FROM_DAYS(num)	Retorna un valor de fecha asociado con el parámetro *num*
HOUR(hora)	Retorna las horas de una *hora* (entre 0 y 23)
LAST_DAY(fecha)	Retorna el último día del mes de una *fecha* dada

Función	Operativa
MAKEDATE(año,diaAño)	Retorna una fecha dado un año y un día del año
MAKETIME(hora, minuto,segundo)	Retorna un valor tipo hora a partir de los argumentos
MICROSECOND(exp)	Retorna el valor en microsegundos de hora hora o fecha-hora dada (entre 0 y 999999)
MINUTE(hora)	Retorna los minutos de una *hora* (entre 0 y 59)
MONTH(fecha)	Retorna el mes de una *fecha* dada (entre 1 y 12)
MONTHNAME(fecha)	Retorna el mes de una *fecha* dada (entre 'Enero' y 'Diciembre')
NOW()	Retorna la fecha y hora actual con formato 'YYYY-MM-DD'
QUARTER(fecha)	Retorna el cuarto del año correspondiente a *fecha* (entre 1 y 4)
SECOND(hora)	Retorna los minutos de una *hora* (entre 0 y 59)
SEC_TO_ TIME(segundos)	Retorna la hora (HH:MM:SS) a partir de una cantidad de *segundos*
STR_TO_DATE(cadena, tipo_formato)	Retorna una fecha expresada como *cadena* en un formato fecha indicado en *tipo_ formato*
TIME(exp)	Extrae la hora de la hora o fecha-hora dada por *exp*
TIFEDIFF(exp1,exp2)	Retorna el tiempo entre una hora de inicio y otra de fin
TIME_TO_SEC(hora)	Retorna los segundos de una *hora*
TO_DAYS(fecha)	Retorna el número de días desde el año 0 hasta *fecha*
WEEK(fecha[,modo])	Retorna el número de semana para *fecha*. Consultar los modos en el manual de referencia de MySQL
WEEKDAY(fecha)	Retorna el índice de días de la semana para una *fecha* (entre 0=lunes y 6=domingo)
WEEKOFYEAR(fecha)	Retorna la semana de la fecha (entre 1 y 53)
YEAR(fecha)	Retorna el año de la *fecha* (entre 1000 y 9999)
YEARWEEK(fecha)	Retorna el año y semana de una *fecha*

Las principales **funciones sobre cadenas de caracteres** en MySQL son las siguientes:

Función	Operativa
ASCII(cadena)	Retorna el valor numérico (entre 0 y 255) del carácter más a la izquierda de *cadena* (0 si la cadena está vacía)
BIN(num)	Retorna una representación de cadena de caracteres del valor binario *num*
BIT_LENGHT(cadena)	Retorna la longitud de *cadena* en bits
CHAR(num1,num2…)	Retorna la cadena de caracteres resultado de concatenar los caracteres por parámetro (expresados por su valor numérico o por el char)
CHAR_LENGTH(cadena), CHARACTER_LENGTH (cadena)	Retorna la longitud de *cadena* medida en número de caracteres
CONCAT(cad1,cad2,…)	Retorna la concatenación de las cadenas de caracteres aportadas como parámetros de entrada
CONCAT_WS(separador, cad1, cad2, …)	Se trata de una forma especial de *CONCAT()*. Concatena las cadenas añadiendo el separador indicado en el primer parámetro de entrada

Función	Operativa
CONV(num,from_base, to_base)	Convierte el número*num* de la base *from_base* a la base *to_base*. Lo retorna como cadena de caracteres
ELT(num, cad1,cad2,...)	Retorna la cadena de caracteres ubicada en la posición dada por *num*
FIELD(cad, cad1, cad2...)	Retorna el índice en que se encuentra *cad* de la lista {*cad1,cad2…*}
INSERT(cad1, pos, long,cad2	Retorna la cadena *cad1* con la subcadena en la posición *pos* y de longitud *long* de caracteres reemplazados por la cadena *cad2*
INSTR(cad1,cad2)	Retorna la posición de la primera ocurrencia de *cad2* en *cad1*
LEFT(cadena,long)	Retorna los *long* caracteres empezando por la izquierda de la *cadena*
LENGTH(cadena)	Retorna la longitud de *cadena* en bytes
LOAD_FILE(nombre_arc)	Leer el fichero y retorna su contenido como cadena de caracteres
LOCATE(cad1,cad2), LOCATE(cad1,cad2,pos)	Retorna la posición de la primera ocurrencia de *cad2* en *cad1.* En caso de llevar el argumento *pos* se buscará la primera ocurrencia a partir de la posición comentada
LOWER(cadena), LCASE(cadena)	Retorna la cadena *cadena* con todos los caracteres cambiados a minúsculas
REPLACE(cad,cadorigen, caddestino)	Reeplaza de *cad* todas las ocurrencias de *cadorigen* con el texto de *caddestino*
REVERSE(cadena)	Retorno *cadena* con el orden de sus caracteres invertidos
RIGHT(cadena, long)	Retorna los *long* caracteres empezando por la derecha de la *cadena*
SPACE(num)	Retorna la cadena consistente en *num* caracteres en blanco
SUBSTRING(cadena,pos)	Retorna una subcadena de *cadena* a partir de la posición *pos*
STRCMP(cad1,cad2)	Retorna 0 si las cadenas son idénticas, -1 si el *cad1* es menor que *cad2*, y, 1 si *cad1* es mayor que *cad2*

8.4 PROGRAMACIÓN DE MÓDULOS DE MANIPULACIÓN DE LA BASE DE DATOS: PAQUETES, PROCEDIMIENTOS Y FUNCIONES

En el contexto de la programación se denomina **subprograma** al conjunto de instrucciones que persiguen un propósito específico y que son *encapsuladas* formando, de esta forma, una única unidad operativa, que pueda ser invocada en sucesivas ocasiones para su ejecución.

De hecho, de los anteriormente comentados conceptos de subprograma y encapsulación nacen paradigmas de la programación tan extendidos como ***"reutilización de código"***, ***"divide y vencerás"***, ***"modularidad"***, ***"abstracción"*** y un largo etcétera.

Todo subprograma, al igual que sucede en el resto de lenguajes de programación, tendrá:

- Un nombre.
- Puede tener una lista de parámetros.
- Un contenido, que puede estar compuesto de instrucciones SQL, estructuras de control, declaración de variables y, en definitiva, todo cuanto hemos estado comentando en este capítulo.
- Incluso, como veremos, llamadas o invocaciones a otros subprogramas (funciones o procedimientos).

Podemos justificar la existencia de las funciones y procedimiento citando algunas de las principales ventajas de su uso. Son estas:

- **Seguridad:** las funciones y procedimientos permiten ocultar la topología de la base de datos con la que se está trabajando (nombre de tablas, relaciones, etc.) a usuarios que tienen privilegios para manipular dicha base de datos. Esto es posible ya que estos usuarios podrán manipular la base de datos a través de la invocación de estas funciones y procedimientos sin ver su contenido.
- **Reutilización de código:** si nos enfrentamos a un proyecto grande en el que participen varios equipos de desarrollo, la encapsulación de rutinas permite que estos equipos de desarrollo compartan el trabajo ya hecho, optimicen esfuerzos, no incurran en duplicidades de código (redundancia), etc. Esto será posible si se lleva a cabo una buena documentación del código generado.
- **Optimización del tiempo:** siempre es mucho más sencillo reutilizar un subprograma que implementarlo de nuevo.

En cuanto a los procedimientos y las funciones, formalmente se pueden definir como subprogramas que se diferencian entre sí por una característica concreta. Mientras que las **funciones** son subprogramas que, al finalizar sus instrucciones generan un nuevo valor (variable), que devolverán al subprograma que las ha invocado, los **procedimientos** son subprogramas cuyas instrucciones no generan y devuelven una nueva variable, sino que, o bien modifican valores anteriormente definidos (y comunicados mediante *parámetros* o *argumentos*), o bien realizar otras tareas que no implica la modificación o creación de variables.

NOTA

El lenguaje de programación con el que estamos desarrollando este capítulo, esto es, el que ofrece el DBMS MySQL, no da soporte para la utilización de **paquetes**.

Si el lector quiere familiarizarse con este concepto, le proponemos la lectura de la siguiente página web, del portal ***desarrolloweb.com***, donde explica, con detenimiento y variados ejemplos, su utilización para la DBMS Oracle:
http://www.desarrolloweb.com/articulos/paquetes-oracle.html

Funciones

Sintaxis:

CREATE FUNCTION *nombre_función* ([$parámetro_1$, $parámetro_2$, …, $parámetro_N$])
RETURNS *tipo_dato*

Características

BEGIN
Instrucciones;
RETURN *nombre_variable*;
END;

Donde:

- Las *características* pueden contener los siguientes valores:
  - LANGUAGE SQL
  - [NOT] DETERMINISTIC
  - {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
  - SQL SECURITY {DEFINER | INVOKER}
  - COMMENT 'Comentarios'

NOTA

Un procedimiento o función se dice **DETERMINISTIC** (determinista) si siempre genera el mismo resultado ante los mismos parámetros de entrada, si no es así, será **NOT DETERMINISTIC** (opción por defecto).

Estas son las características que proporcionan información sobre la naturaleza de los datos usados en la rutina:

- **CONSTAINS SQL** indica que la rutina no contiene comandos que leen o escriben datos (Valor por defecto).
- **NO SQL** denota que la rutina no contiene comandos SQL.
- **READS SQL DATA** informa que la rutina contiene comandos que leen datos, pero no comandos que escriben.
- **MODIFIES SQL DATA** indica que la rutina contiene comandos que pueden escribir datos.

En cuanto a la característica **SQL SECURITY** sirve para indicar si la rutina debe ser ejecutada usando los permisos del usuario que la creó o los del usuario que la invoca:

- **DEFINER** indica que el creador debe tener permisos para acceder a la base de datos con la que la rutina trabaja.
- **EXECUTE** indica que el invocador debe tener permisos para acceder a la base de datos con la que la rutina trabaja.

La cláusula **COMMENT** puede emplearse para describir el procedimiento o la función.

- Los *parámetros* siguen la siguiente sintaxis:

  [modo] nombre_variable tipo_variable

  El atributo *modo*, que es opcional, puede tener los siguientes valores:

  - **IN:** indica que el parámetro es únicamente de entrada (si no especifica modo, este será el que se tome por defecto).
  - **OUT:** indica que ese parámetro va a ser modificado por el procedimiento.
  - **INOUT:** es una combinación de los anteriores. Indica que ese parámetro va a ser útil como información de entrada en el procedimiento pero que, a su vez, también va a almacenar un nuevo valor como resultado del procedimiento.

EJEMPLO 8.12

Implementar una función que devuelva la edad de un alumno cuyo DNI viene dado por parámetro de entrada:

```
CREATE FUNCTION consultarEdad(dni VARCHAR(9))
RETURNS INT
BEGIN
DECLARE edadINT;
SELECT Edad_Al INTO edad FROM Alumnos WHERE DNI_Al = dni;
RETURN edad;
END;
```

Procedimientos

Sintaxis:
CREATE PROCEDURE *nombre_procedimiento* ([$parámetro_1$, $parámetro_2$, …, $parámetro_N$])

Características
BEGIN
Instrucciones;
END;

Donde:

- Las características pueden contener los siguientes valores:
  - LANGUAGE SQL
  - [NOT] DETERMINISTIC
  - {CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
  - SQL SECURITY {DEFINER | INVOKER}
  - COMMENT 'Comentarios'

NOTA

Estas características son las mismas y con el mismo significado que en ***funciones.***

- Los parámetros siguen la siguiente *sintaxis*:

  [modo] nombre_variable tipo_variable

  El atributo *modo*, que es opcional, puede tener los mismos valores anteriormente comentados:
  - IN
  - OUT
  - INOUT

EJEMPLO 8.13

Implementar un procedimiento que actualice (ya que ha pasado un año) la edad de un alumno cuyo DNI viene dado por parámetro de entrada:

```
CREATE PROCEDURE actualizarEdad(dni VARCHAR(9))
BEGIN
UPDATE Alumnos
SET Edad_Al = Edad_Al + 1
WHERE DNI_P = dni;
END;
```

Utilización de las funciones y procedimientos

Llamada o invocación de subprogramas

Para llamar a la ejecución de una función o procedimiento utilizaremos la palabra reservada CALL siguiendo la siguiente *sintaxis*:

CALL *nombre_subprograma*(*nombre_variable1, nombre_variable2, …, nombre_variableN*);

Donde *nombre_variable1, nombre_variable2, …, nombre_variableN* son los argumentos que recibirá dicha función o procedimiento.

Modificación de subprogramas

Para modificar las características de los subprogramas seguiremos la siguiente *sintaxis*:
ALTER {PROCEDURE | FUNCTION} *nombre_subprograma* [*Características*];

Donde las *características* son las indicadas anteriormente.

Borrado de subprogramas

Para borrar subprogramas seguiremos la siguiente *sintaxis*:
DROP {PROCEDURA | FUNCTION}[IF EXISTS] *nombre_subprograma*;

Esta instrucción elimina la rutina indicada del servidor, pero, para ello, se debe tener permiso *ALTER ROUTINE*. Dicho permiso se concede automáticamente al creador de la rutina.

Por su parte la cláusula **IF EXISTS** evita que se genere una incidencia si el procedimiento o función no existe.

8.5 HERRAMIENTAS DE DEPURACIÓN Y CONTROL DE CÓDIGO

Generalmente, los IDE de propósito general y algunos IDE en el entorno de la base de datos, como Visual Studio, tienen su propia herramienta de depuración y control de código. No obstante, dado que no todos los IDE en el entorno de la base de datos poseen dicha herramienta y, además, no todos los programadores en el entorno de las bases de datos emplean IDE, para desarrollar este apartado hemos optado por explicar una herramienta independiente, muy fácil de utilizar y con un entorno gráfico muy intuitivo y similar al del resto de entornos de depuración: **Debugger for MySQL**.

INFORMACIÓN

Aconsejamos la lectura de la página oficial de **Debugger for MySQL**, donde nos hemos documentado para la realización de esta sección: *http://mydebugger.com/*

Como comentario interesante de este artículo, queremos destacar las principales características de la herramienta, según dicho portal:

- Depura y gestiona de manera eficiente los procedimientos y funciones almacenadas en MySQL y MariaDB.
- Depura procedimientos y funciones anidadas, así como el *código anónimo* (similar a los *bloques anónimos* en Oracle).
- Podemos definir puntos de interrupción condicionales (más adelante hablaremos de ellos).
- Ofrece información sobre herramientas de evaluación de expresiones.
- Ejecuta comandos SQL en el contexto de depuración.
- Su elegante editor de código proporciona opción de autocompletar palabras reservadas del lenguaje de programación.
- Ofrece soporte completo Unicode.

8.5.1 INSTALACIÓN DE LA HERRAMIENTA

Para descargar una versión de prueba debemos entrar en la web del producto: *http://mydebugger.com/*

Y, en la parte superior derecha de dicha web (tal y como se indica en la Figura 8.45) pulsar sobre el botón *Download*. También es posible, si se desea, comprar dicha herramienta, para lo cual se tendría que pulsar el botón *Buy now*.

No obstante, la versión de evaluación contiene toda la operativa para poder trabajar y evaluar si el producto es o no del agrado del programador. Una vez evaluado el producto, ya podremos plantearnos si adquirir una licencia por uno o por tres años.

Figura 8.45. *Página web oficial de* ***Debbuger for MySQL***

Por último, el proceso de instalación es muy intuitivo, ya que tan solo se debe ir aceptando en los diferentes cuadros de diálogo de instalación de la herramienta.

8.5.2 ENTRADA EN LA APLICACIÓN Y CONEXIÓN CON LA BASE DE DATOS

Cada vez que queramos entrar en la aplicación, se nos abrirá un cuadro de diálogo para la conexión con la base de datos, tal y como observamos en la Figura 8.46.

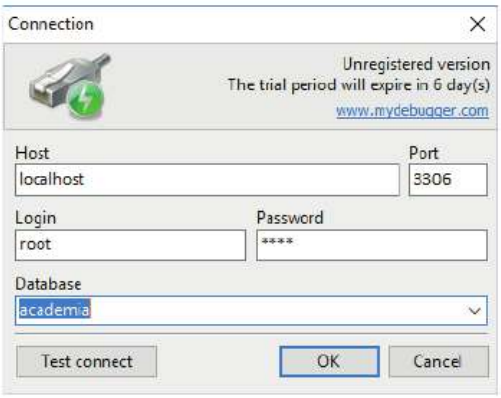


Figura 8.46. *Cuadro de diálogo para establecer la conexión con la base de datos*

Los elementos fundamentales de conexión son los siguientes:

- **Host:** se establece la dirección IP del servidor donde esté instalado el DBMS MySQL con nuestra base de datos. En nuestro caso, como instalamos MySQL en nuestro *host*, escribiremos ***localhost***.
- **Port:** puerto a través del cual se conectará con el DBMS. Como podemos recordar en la Figura 8.12, el más usual (y el que hemos utilizado) es el puerto número ***3306***.
- **Login:** por defecto, se establece ***'root'***.
- **Password:** contraseña de acceso al sistema de gestión de base de datos. Como vemos en la Figura 8.12, nosotros establecimos como contraseña la cadena ***'root'***.
- **Database:** nombre del esquema de base de datos donde tenemos implantada nuestra base de datos. En el ejemplo de este libro, el esquema se denomina ***'academia'***.

Es recomendable, antes de aceptar la configuración y entrar en la herramienta, hacer una prueba de conectividad (*Test connect*), pulsando el botón para ello, como se muestra en la figura anterior.

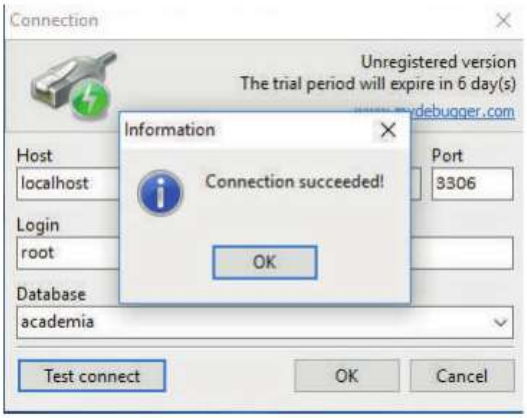


Figura 8.47. *Mensaje que ofrece la prueba de conectividad*

8.5.3 VENTANA PRINCIPAL DE LA HERRAMIENTA

Al entrar en la aplicación, vemos las siguientes áreas sobre la ventana principal:

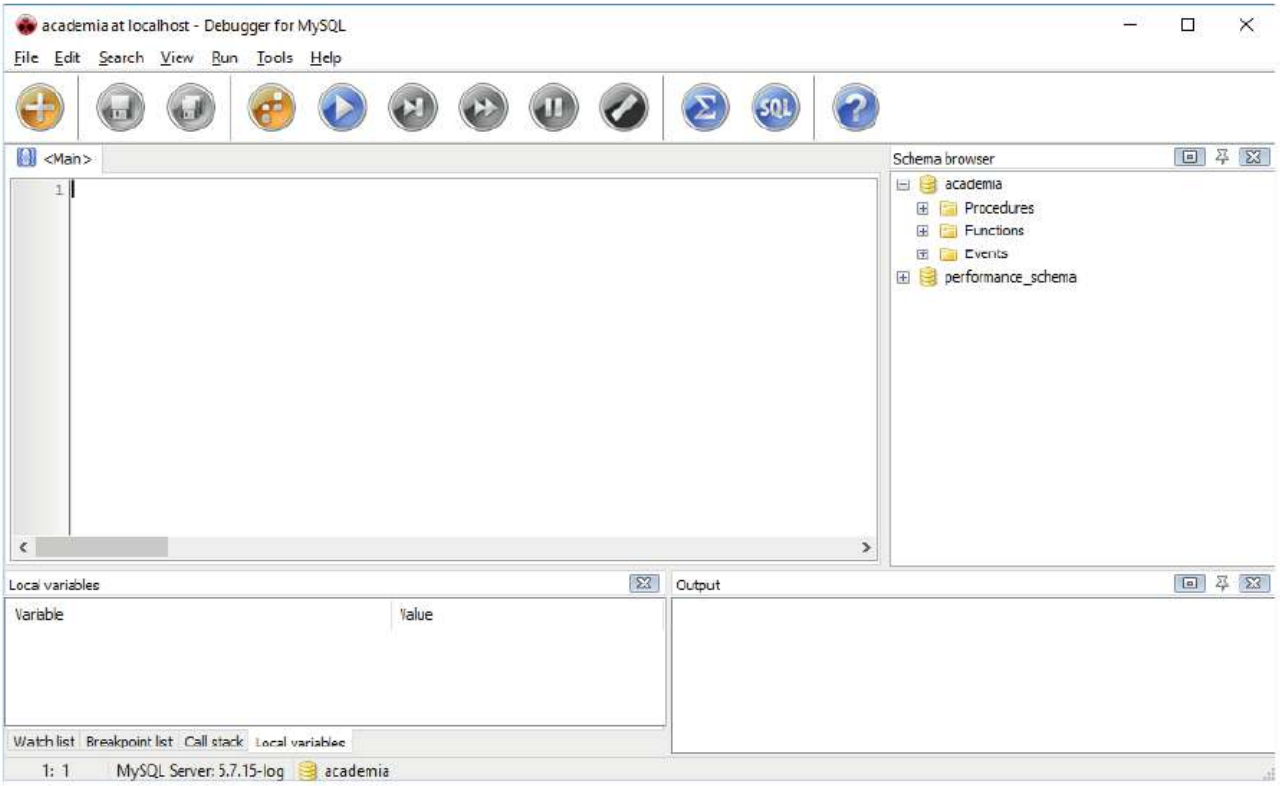


Figura 8.48. *Ventana principal de Debbuger for MySQL*

- **Área de menús**, donde se sitúan todas las opciones que nos ofrece la herramienta, junto a botones con accesos directos a las más destacadas (se sitúa en la parte superior de la ventaja).
- **Editor de código fuente**, donde se puede editar las diferentes llamadas a los procedimientos y funciones, en el denominado *<Main>,* como editar los diferentes subprogramas ya creados en MySQL Workbench o incluso crear otros nuevos y escribirlos desde su inicio (como también veremos).
- El área situada a la derecha y denominada **Schema browser** es un explorador donde podemos navegar con todos los elementos de la base de datos a la que nos hemos vinculado (tablas, procedimientos, funciones, etc.). Además, podemos añadir nuevos elementos.
- El área inferior izquierda puede mostrar diferentes opciones seleccionables a través de diferentes pestañas:
  - **Local variables** (seleccionada por defecto) muestra las variables que se han declarado en el subprograma, así como la evolución de sus valores.
  - **Call stack** almacena las sucesivas llamadas de un subprograma a otro (tiene utilidad cuando tenemos varias funciones o procedimientos que se llaman a ejecución entre sí).
  - **Breakpoint list** muestra los puntos de interrupción que hemos marcado en el código fuente a evaluar, destacando la línea de código donde se encuentra y, si se trata de un *breakpoint condicional*, cuál es la condición.

- **Whatch list** o *lista de vigilancia* es una opción para ver ciertas variables o expresiones que entendemos relevantes y las añadimos a la citada lista. Para añadir a la lista una variable tan solo tenemos que seleccionarla y pulsar *Ctrl + F5*.

- La salida del subprograma al usuario se ofrece en el cuadro inferior derecho, denominado **Output**.

8.5.4 NUESTRA PRIMERA DEPURACIÓN SENCILLA

Para mostrar cómo se depura vamos a proceder a la depuración de nuestro ejemplo más sencillo, el denominado *diHola* que hemos mostrado anteriormente.

En primer lugar, localizaremos el procedimiento en el *Schema browser*, tras lo cual pulsaremos con el botón derecho y elegiremos la opción *Configure environment for debug this routine*, como vemos en la figura.

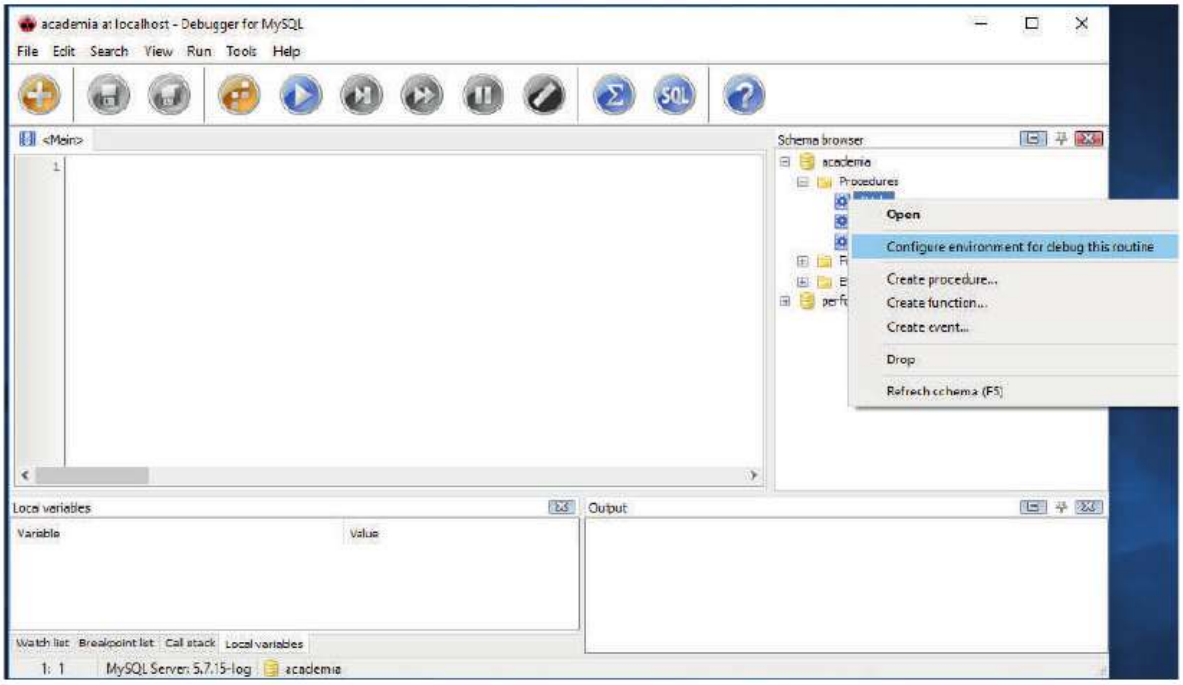


Figura 8.49. *Inicio de configuración para efectuar una depuración de código con Debbuger for MySQL*

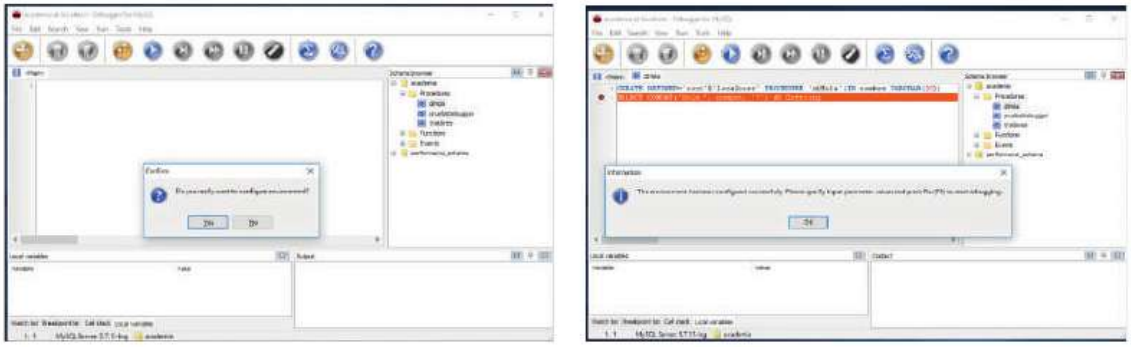


Figura 8.50. *Ventanas emergentes del proceso de configuración de una depuración de código*

Finalmente, **Debbuger for MySQL** nos genera un <*Main*>, es decir, una propuesta de llamada al procedimiento pasándole los parámetros de entrada que ha detectado como necesarios (por defecto siempre le asigna valores *Null*), como podemos ver en la Figura 8.51, y, además, una propuesta de punto de interrupción sobre el código a depurar, como se aprecia en la Figura 8.52.

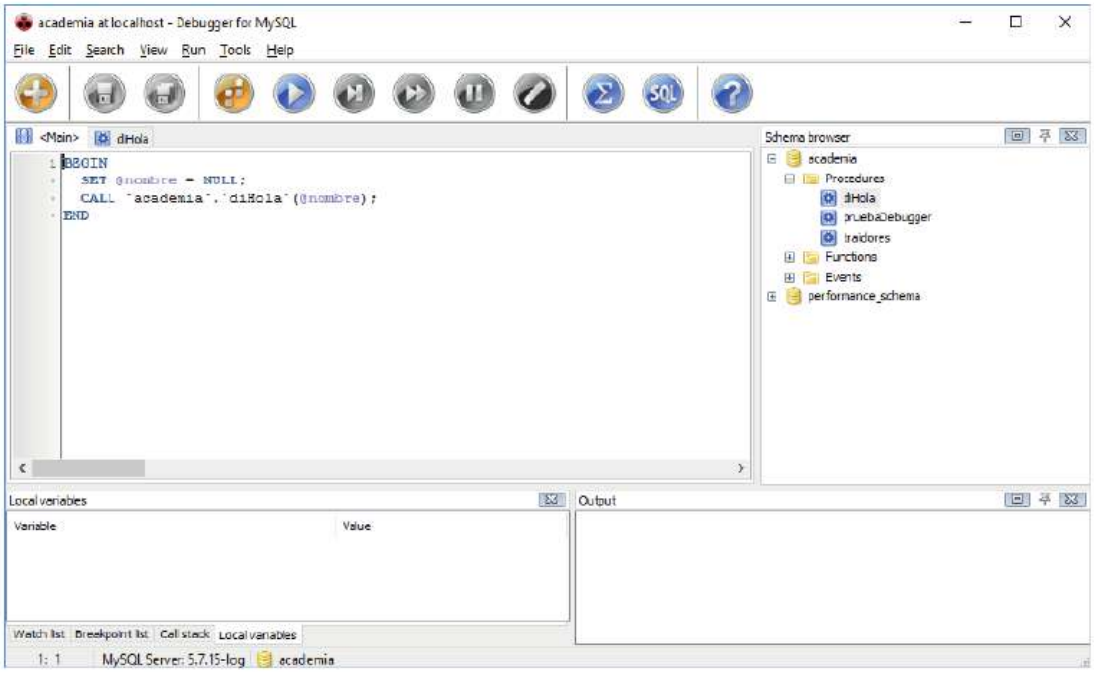


Figura 8.51. *Código generado con la llamada al procedimiento diHola*

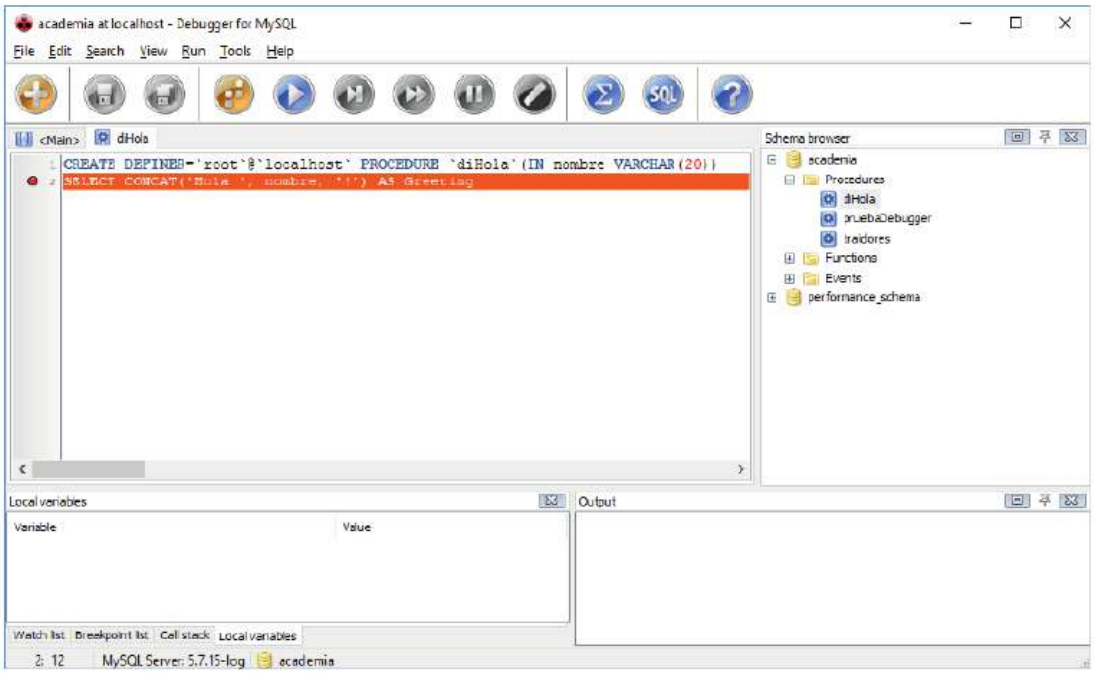


Figura 8.52. *Código fuente diHola con la selección del punto de interrupción que la herramienta ha seleccionado como más adecuado*

No obstante, se pueden añadir nuevos *breakpoints* haciendo clic en el editor, en la zona izquierda de los números de líneas. De hecho, en la Figura 8.53 podemos ver un nuevo punto de interrupción en la primera línea del procedimiento.

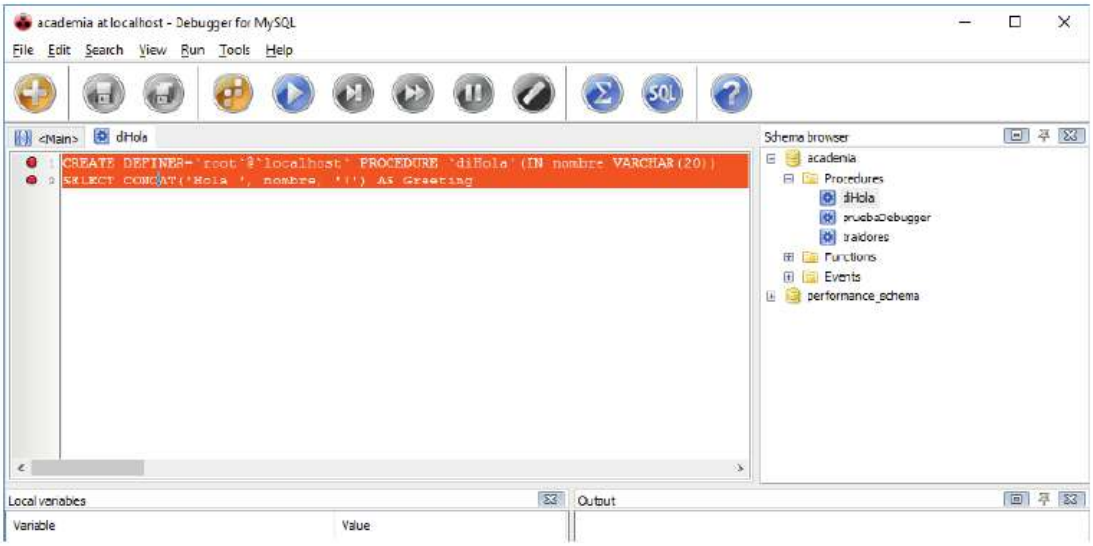


Figura 8.53. *Código fuente diHola con un segundo punto de interrupción*

Y, finalmente, para lanzar el depurador tendríamos que utilizar los botones de progreso o pulsar sucesivamente la tecla *F9*.

De esta forma, como se puede ver en las figuras inferiores, se observará la evolución de los contenidos de las variables locales, la lista de *breakspoints*, etc., junto a las salidas del proceso de depuración.

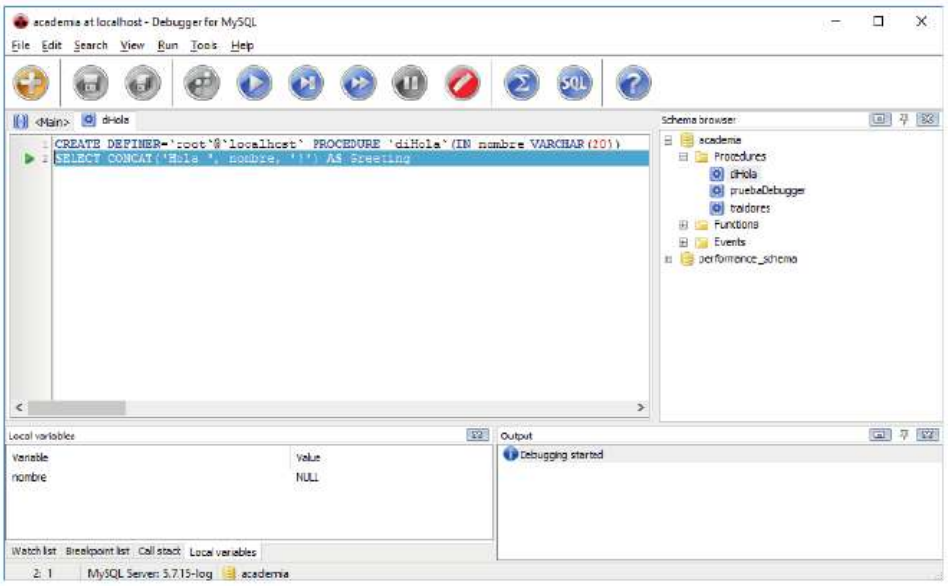


Figura 8.54. *Proceso de depuración iniciado*

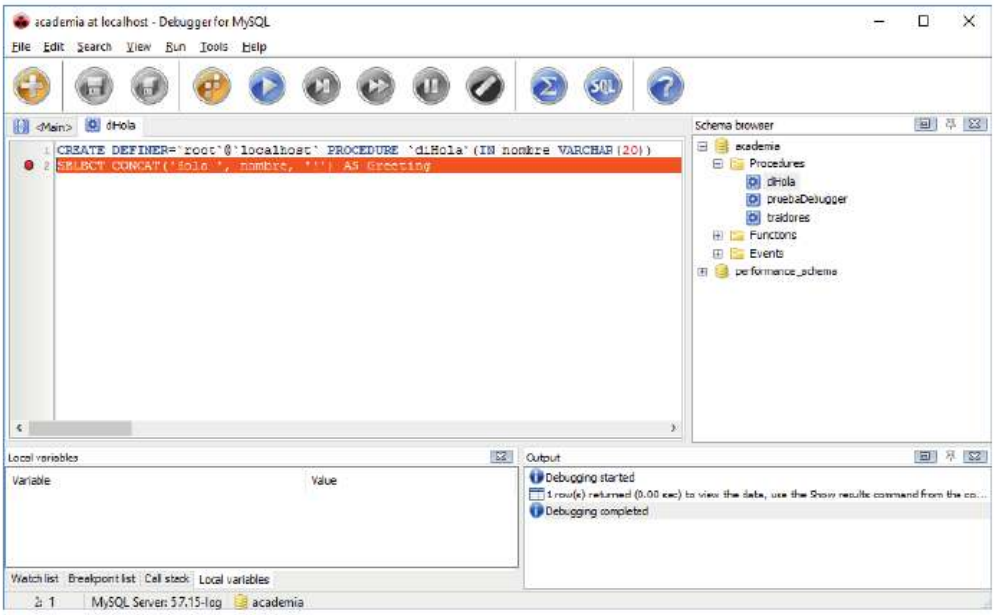
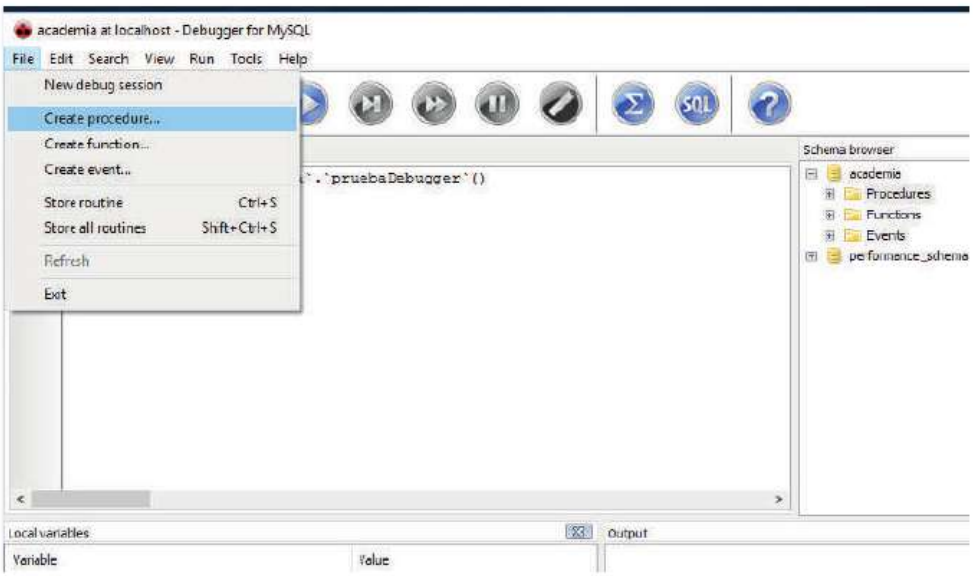


Figura 8.55. *Proceso de depuración finalizado*

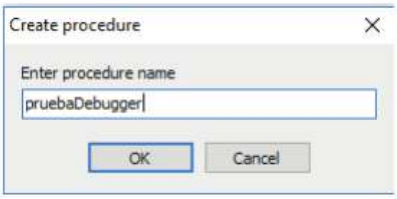
NOTA

Aunque la forma de trabajar que estamos empleando en este capítulo es la de programar los sucesivos procedimientos y funciones utilizando el IDE MySQL Workbench y solo usar esta herramienta (Debugger for MySQL) para la depuración de dichos subprogramas, también es posible crear y editar funciones y procedimientos utilizando la presente aplicación.

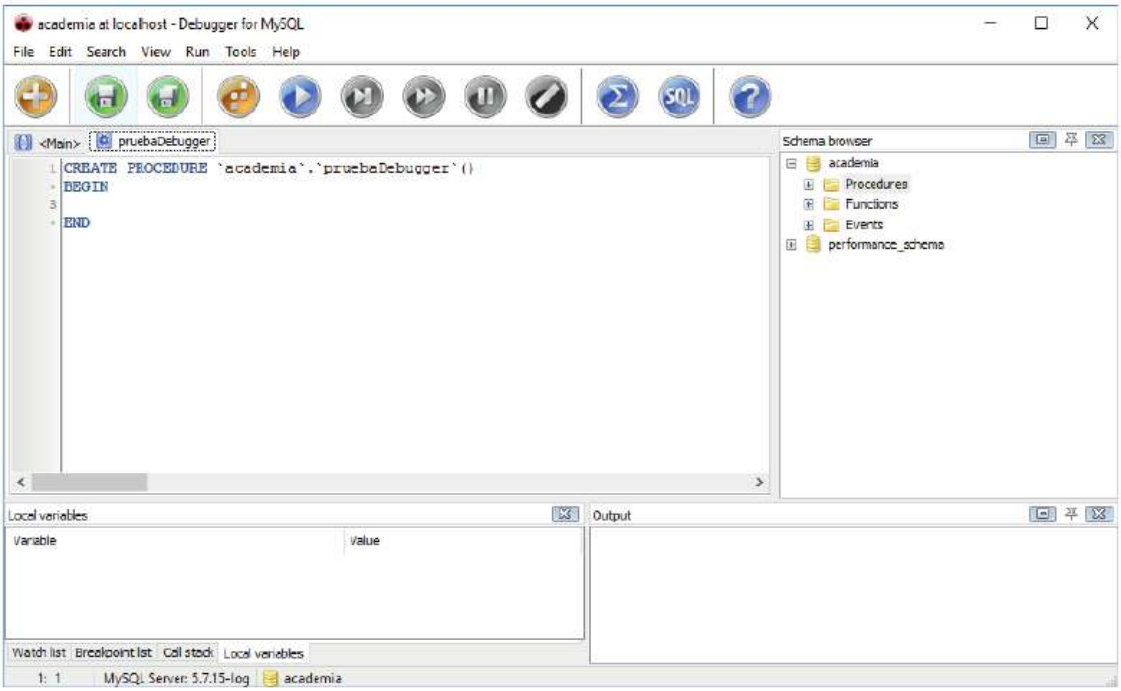
Para ello tendríamos que pulsar la opción *File → Create procedure…* o bien *File → Create function…*



Ante esta selección, se abrirá un cuadro de diálogo en el cual nos solicitará por el nombre del nuevo subprograma.



Y, tras escribir el nombre de dicho subprograma, se creará el mismo (pudiéndose ver en el *Schema browser*). Además, se abrirá en el editor de la herramienta con los elementos básicos del código fuente del mismo, como vemos en la figura.



NOTA

Como comentábamos anteriormente, también se pueden establecer **puntos de interrupción condicionales**, es decir, puntos de interrupción en los que solo se detiene el depurador bajo determinadas circunstancias. Para hacer eso, debemos llevar a cabo los siguientes pasos:

- Hacemos clic en el punto de interrupción del cuadro de diálogo *Breakpoint list* y seleccionamos la opción *Properties…* del menú emergente.
- En el cuadro de diálogo para configurar dichas propiedades, introduciremos una expresión condicional en el campo *Condition textbox*.

8.6 HERRAMIENTAS GRÁFICAS DE DESARROLLO INTEGRADAS EN LA BASE DE DATOS

El objetivo de este capítulo es poder utilizar un entorno gráfico amigable para llevar a cabo operaciones básica en el entorno del DBMS MySQL, el cual no requiera conocimientos de programación.

Dicho de otra forma, lo que se pretende es lograr que un usuario no especializado pueda realizar las operaciones básicas sobre los datos de la base de datos (dar de alta, modificar, eliminar registros) sin necesidad de tener conocimientos de programación, sin la utilización del IDE con el que hemos estado trabajando, que recordamos es MySQL Workbench, y sin trabajar directamente con MySQL *Server*.

Para ello, proponemos como herramienta la extendida suite ofimática Microsoft Office, ya que se trata del *software* más utilizado en casi todos los ámbitos profesionales de todo el mundo. Concretamente la aplicación a utilizar será Microsoft Access.

Microsoft Access es por sí mismo un sistema de gestión de bases de datos, que ofrece seguridad y gestión a sus propias tablas, relaciones, restricciones, datos, etc., pero en este caso, nosotros no utilizaremos el motor de gestión de bases de datos, sino que lo vincularemos a nuestro DBMS MySQL. De esta forma, tan solo utilizaremos el *software* de Microsoft para vincularlo a la base de datos que ya tenemos implementada y, sobre ella, crear nuestros informes y formularios, es decir, una interfaz gráfica entre la base de datos y el usuario que trabajará con ella.

Instalación de MySQL Connector/ODBC

Para hacer esto posible, tenemos que instalar un MySQL Connector, es decir, un *software* específico que nos permitirá conectar la base de datos MySQL con el *software* que deseemos, en este caso Access.

En la página de MySQL dedicada a los conectores: *http://dev.mysql.com/downloads/connector/*

Podemos observar el amplio catálogo de sistemas *software* y lenguajes de programación con los que puede vincularse el DBMS MySQL, de los cuales, nosotros tendremos que instalar el conector *ODBC* (válido para *software* de los sistemas operativos Windows, Linux, Mac y Unix).

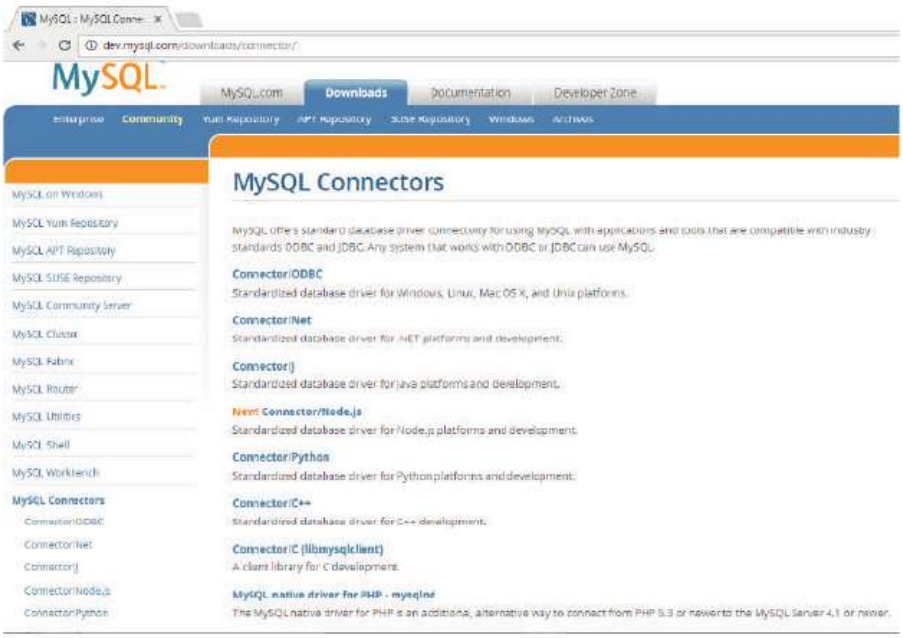


Figura 8.56. *Ventana de descarga de todos los conectores de MySQL con otro software o lenguajes de programación*

No obstante, nosotros no descargaremos el conector ODBC a través de esta página sino a través de la herramienta MySQL Installer con la que gestionamos todo el *software* instalado de MySQL.

Así pues, en primer lugar, entramos en la aplicación, que nos mostrará los dos programas que tenemos instalados, y tendremos que pulsar en el botón *Add…* para añadir el nuevo *software*.

Figura 8.57. *Ventana principal de MySQL Installer*

Tras esto, nos aparecerá un listado con todo el *software* susceptible de ser instalado en nuestro ordenador. En este caso, desplegaremos la opción de los conectores, es decir, MySQL Connectors, como vemos en la Figura 8.58, escogeremos, seguidamente, el conector ODBC (Connector/ODBC), en este caso la última versión disponible y, en último lugar, tendremos que marcar la versión de 32 o de 64 bits, como vemos en la Figura 8.59.

Una vez visto el *software* adecuado, pulsaremos el botón  para pasarlo de la ventana de productos disponibles (*Avaliable Products*) a la de productos que van a ser instalados a través de este proceso (*Products / Features To Be Installed*).

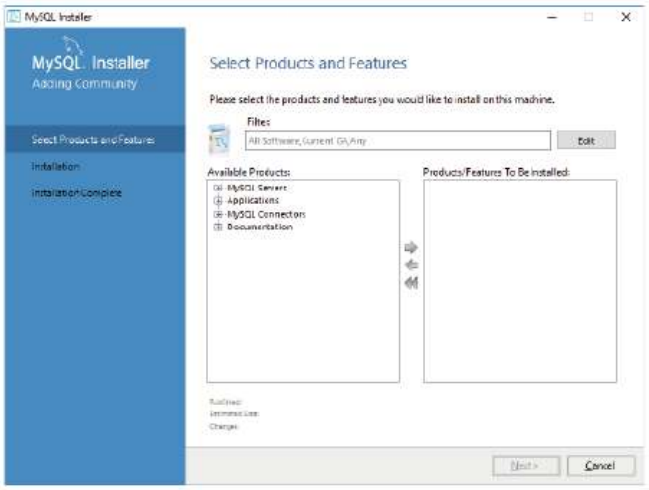


Figura 8.58. *Ventana de selección de productos MySQL a instalar (I)*

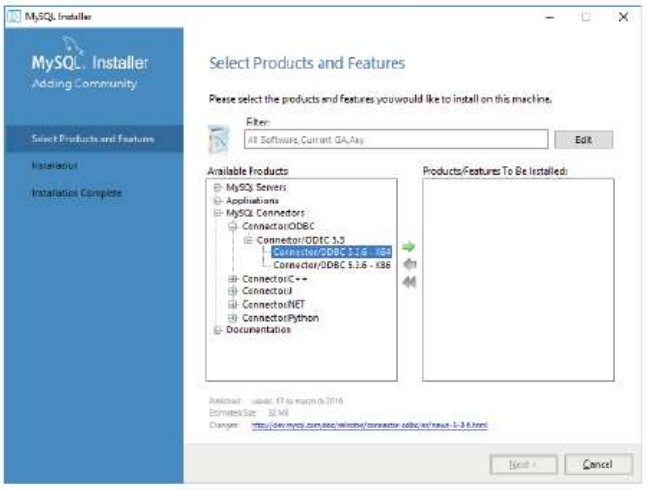


Figura 8.59. *Ventana de selección de productos MySQL a instalar (II)*

Una vez hayamos seleccionado el *software* que deseemos, en este caso, el conector ODBC, pulsaremos el botón *Next* > para iniciar la instalación.

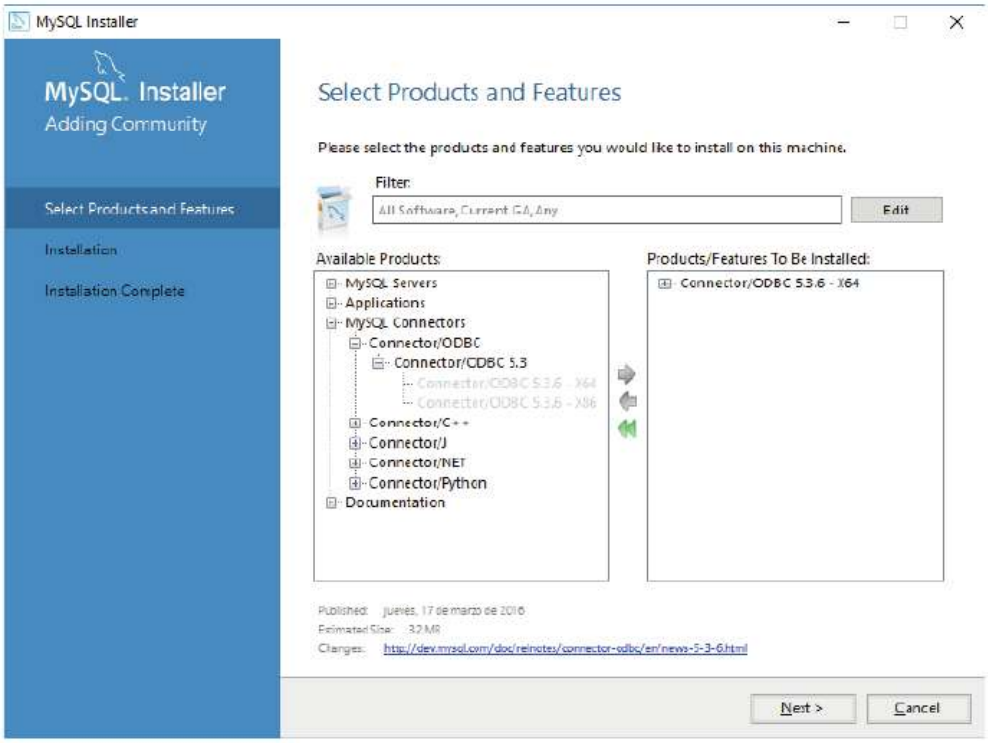
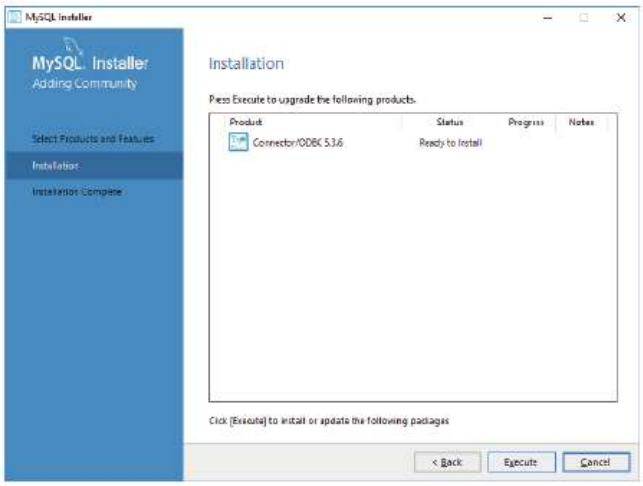


Figura 8.60. *Ventana de selección de productos MySQL a instalar (III)*

Por último, se instalará el *software* para lo cual simplemente iremos avanzando en dicho proceso en el que nos irá indicando el desarrollo del mismo, hasta que veamos, nuevamente en la ventana principal de MySQL Installer, el nombre del nuevo producto recién adquirido, como vemos en la Figura 8.64.



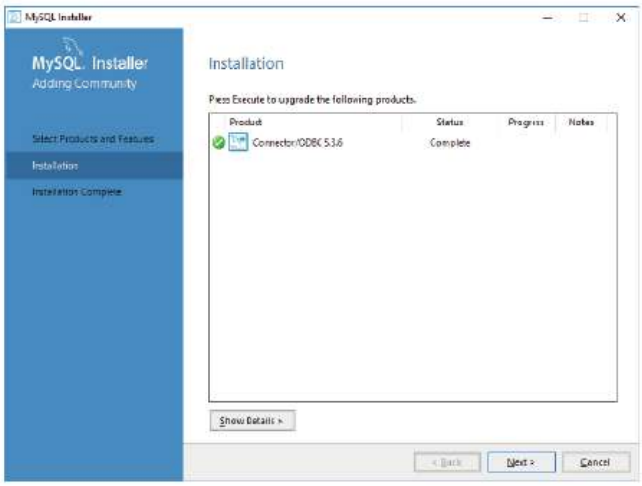


Figura 8.61. *Proceso de instalación de MySQL Connector/ODBC (I)* ***Figura 8.62.*** *Proceso de instalación de MySQL Connector/ODBC (II)*

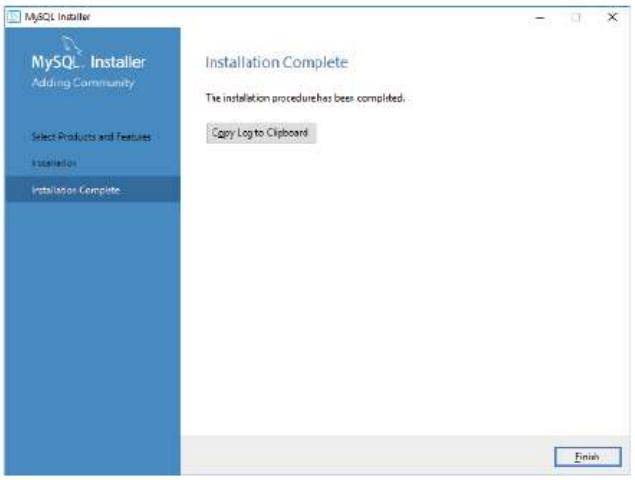


Figura 8.63. *Proceso de instalación de MySQL Connector/ODBC (III)*

Figura 8.64. *Ventana principal de MySQL Installer con el nuevo software instalado*

Configuración del sistema operativo para la detección de la base de datos MySQL

Una vez instalado el ODBC, tenemos que conectar nuestro sistema operativo, en este caso Windows 10, con nuestra base de datos. Para ello, tenemos que entrar en *Herramientas administrativas* del sistema operativo, que se ubica en el *Panel de control*. Desde allí haremos doble clic a la herramienta denominada *Orígenes de datos ODBC (64 bits)* o su versión de 32 bits.

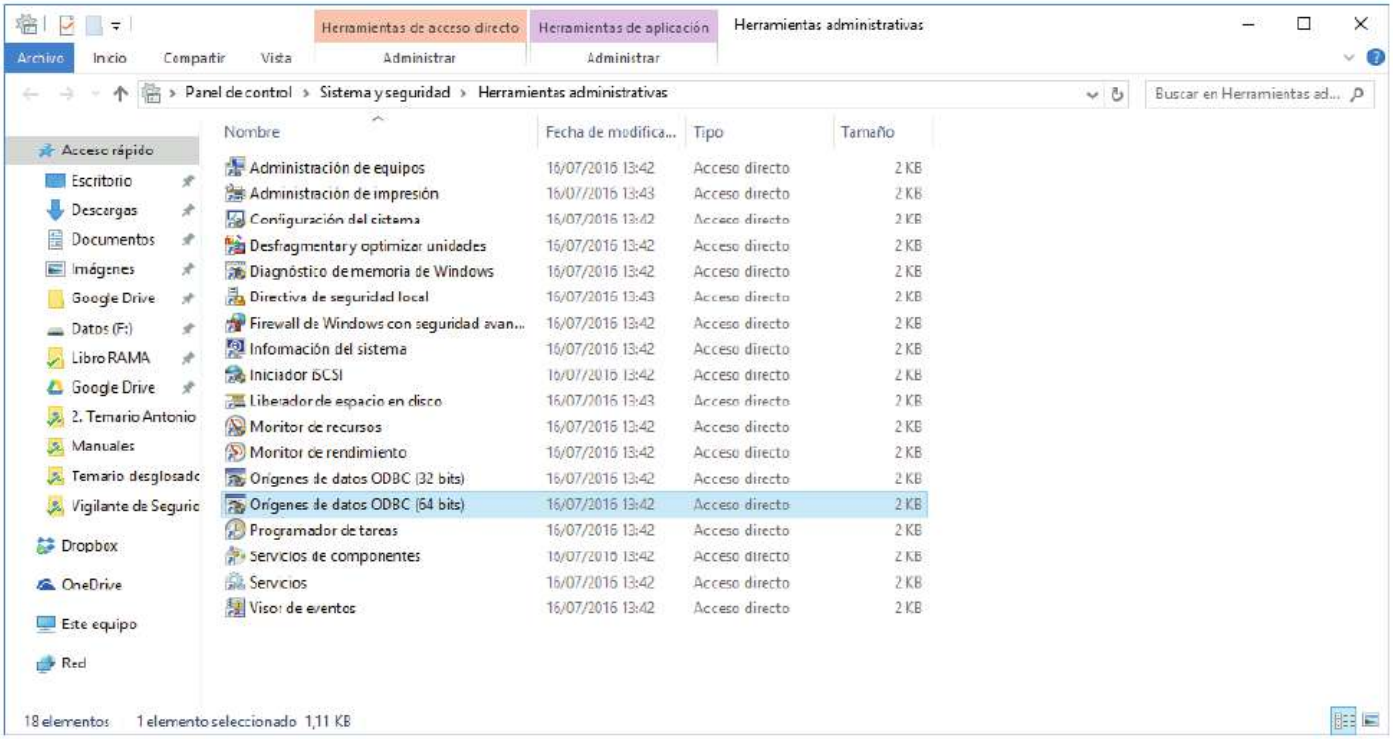


Figura 8.65. *Vista de la ventana donde se organizan las Herramientas administrativas*

Ya en la herramienta, tenemos que *Agregar…* el nuevo origen de datos ODBC y seleccionar el que nos ocupa, es decir, MySQL ODBC 5.3 ANSI Driver.

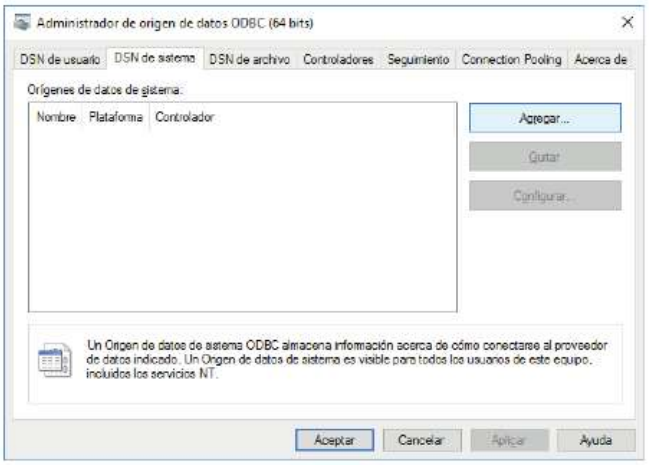


Figura 8.66. *Herramienta de "Administrador de origen de datos ODBC (64 bits)"*

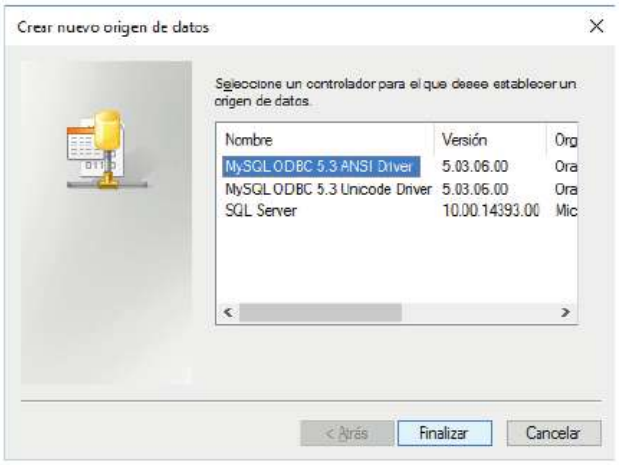


Figura 8.67. *Proceso de incorporación del driver MySQL ODBC 5.3*

Para concluir con el proceso de vinculación, tenemos que introducir los parámetros para que el sistema operativo pueda conectar con la base de datos, los cuales son:

- **Data Source Name** y **Description:** nombre y descripción con el que queremos identificar la nueva conexión a la base de datos.
- **TCP/IP Server:** se establece la dirección IP del servidor donde esté instalado el DBMS MySQL con nuestra base de datos. En nuestro caso, como instalamos MySQL en nuestro *host*, escribiremos ***localhost***.
- **Port:** puerto a través del cual se conectará con el DBMS. Como podemos recordar en la Figura 8.12, el más usual (y el que hemos utilizado) es el puerto número ***3306***.
- **User:** por defecto, se establece ***'root'***.
- **Password:** contraseña de acceso al sistema de gestión de base de datos. Como vemos en la Figura 8.12, nosotros establecimos como contraseña la cadena ***'root'***.
- **Database:** nombre del esquema de base de datos donde tenemos implantada nuestra base de datos. En el ejemplo de este libro, el esquema se denomina ***'academia'***.

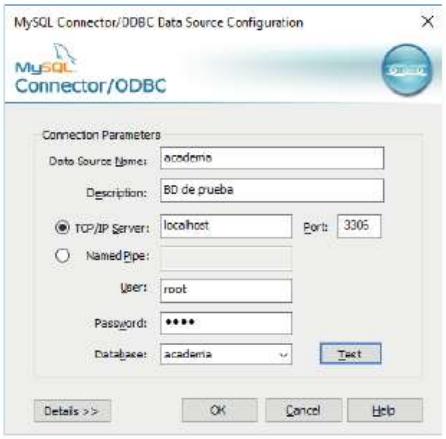


Figura 8.68. *Configuración de la conexión del driver ODBC*

Es recomendable, antes de aceptar la configuración, hacer una prueba de conectividad (*Test*), pulsando el botón para ello, como se muestra en la figura anterior.

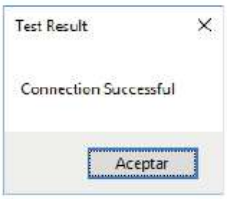


Figura 8.69. *Cuadro de diálogo con el resultado del test de conectividad* ***Importar a Microsoft Access la base de datos MySQL***

El proceso final de vinculación de nuestra base de datos MySQL a Microsoft Access, aplicación con la que, en apartados sucesivos, vamos a crear la interfaz (formularios e informes) con la que un usuario podrá interactuar con nuestra base de datos, consistirá en crear una nueva base de datos en blanco y, posteriormente, en lugar de crear las nuevas tablas, relaciones, etc., importaremos la base de datos del *driver* ODBC que acabamos de configurar.

Para ello, seleccionaremos, dentro de la ficha *Datos externos*, en el grupo *Importar y vincular*, el botón denominado *Base de datos ODBC*, como vemos en la siguiente figura.

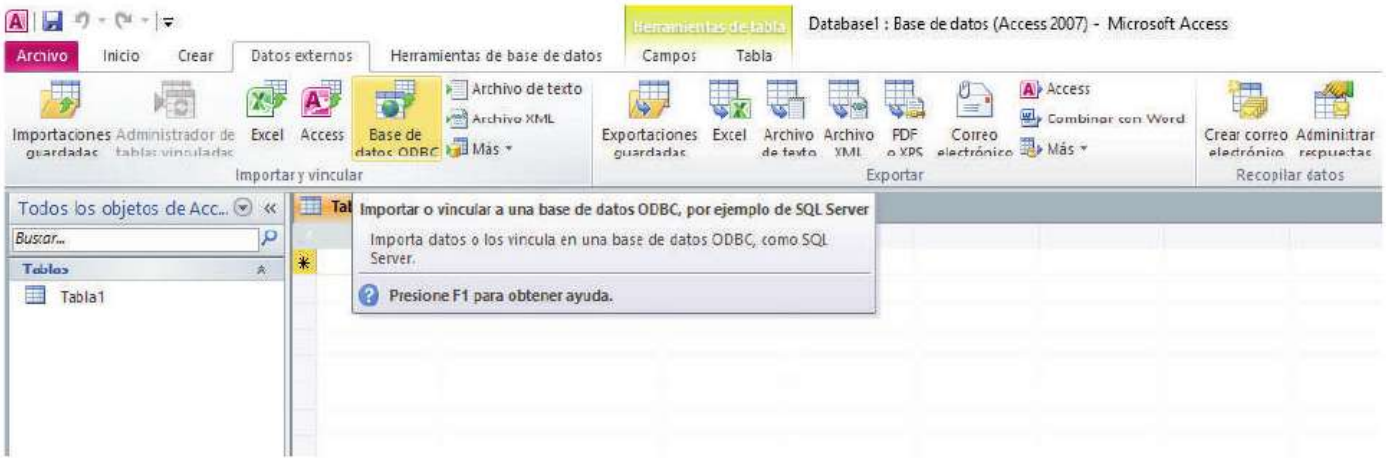


Figura 8.70. *Proceso de importación de una base de datos externa (I)*

A continuación, se iniciará un asistente de importación en el que, en primer lugar, preguntará en qué términos queremos realizar la importación. Se plantean dos opciones:

- **Importar el origen de datos en una nueva tabla de la base de datos actual:** en cuyo caso se importa una copia de la base de datos original y los cambios a realizar en Access no afectarán al origen.
- **Vincular al origen de datos creando una tabla vinculada:** que, como su descripción indica, actualizará los cambios realizados en una u otra herramienta.

En este caso, la opción elegida es la segunda, ya que el objetivo es crear una herramienta gráfica para trabajar con los datos de la base de datos que está almacenada en MySQL, lo cual incluye introducir nuevos datos, modificarlos, eliminarlos, etc.

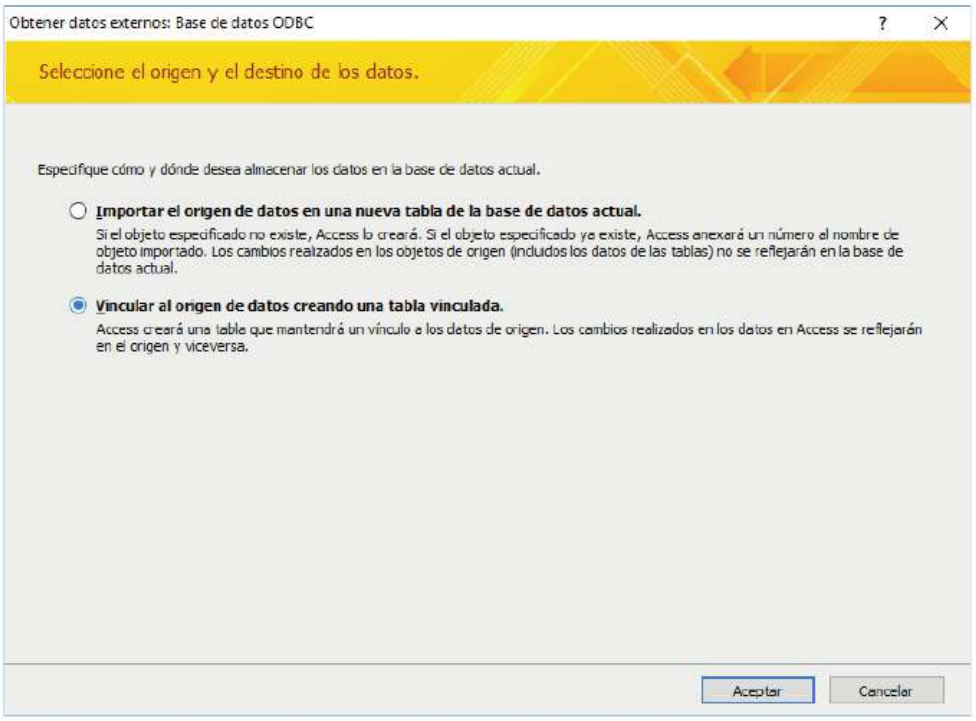


Figura 8.71. *Proceso de importación de una base de datos externa (II)*

Seguidamente, se mostrarán los posibles orígenes de datos en el equipo, seleccionando el que acabamos de configurar y aceptaremos el proceso.

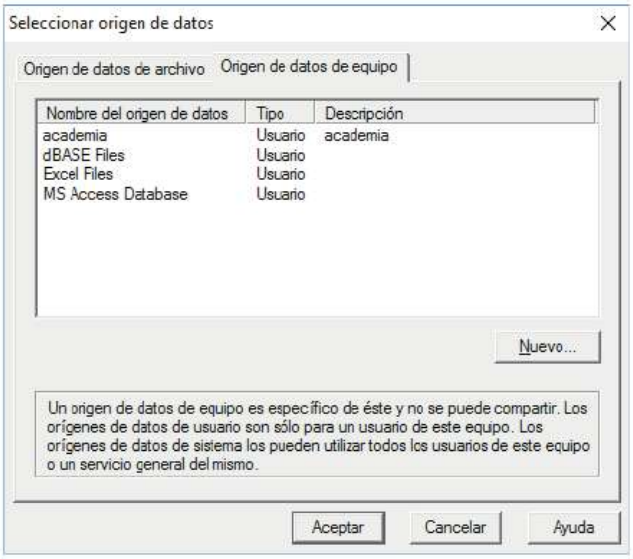


Figura 8.72. *Proceso de importación de una base de datos externa (III)*

El proceso de importación finaliza con la selección de la/s tabla/s concreta/s que queramos vincular, como vemos en la Figura 8.73.

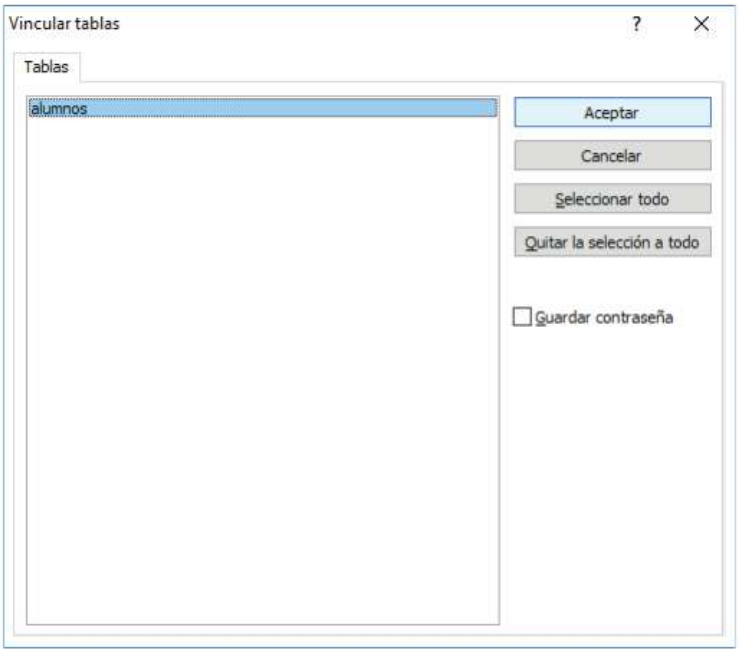


Figura 8.73. *Proceso de importación de una base de datos externa (IV)*

Una vez finalizada la importación, podemos observar que en el explorador de objetos de la base de datos (situado a la izquierda de la ventana) aparecerán las nuevas tablas vinculadas, pero con un icono identificador distinto, que nos indica que se trata de una tabla vinculada y no creada en Access.

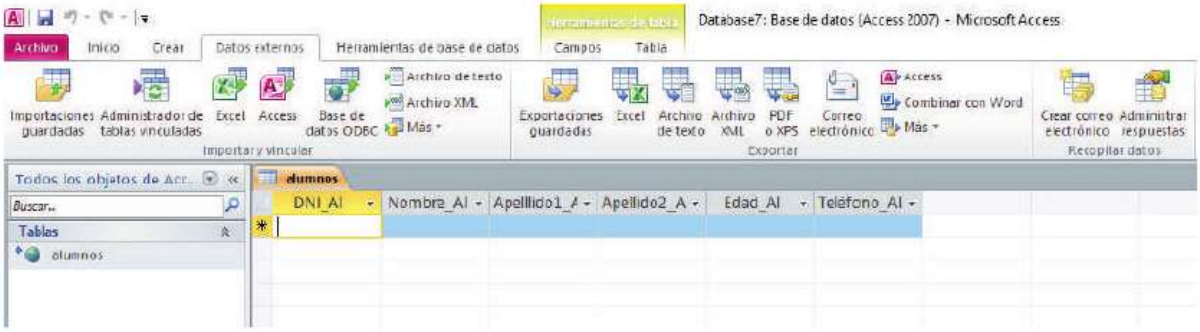


Figura 8.74. *Vista de la ventana principal de Microsoft Access con la tabla vinculada*

Del mismo modo, como podemos demostrar en las figuras siguientes, al realizar una modificación en la tabla de Access (en este caso la incorporación de cuatro alumnos), estos cambios se ven reflejados en MySQL Server.

Esto se puede comprobar realizando una sencilla consulta sobre la base de datos, como vemos en la Figura 8.76, que consiste en visualizar el *Nombre_Al* y la *Edad_Al* de los alumnos de la tabla *alumnos:*

```
USE academia;
SELECT Nombre_Al, Edad_Al FROM alumnos;
```

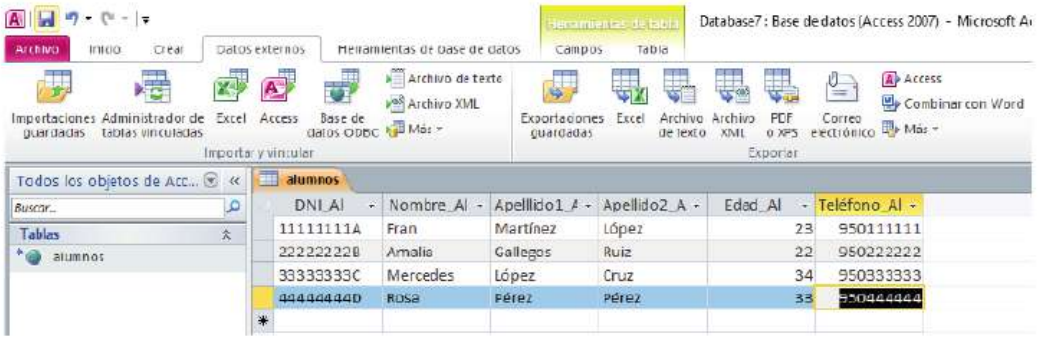


Figura 8.75. *Proceso de incorporación de cuatro alumnos a la base de datos a través de Access*

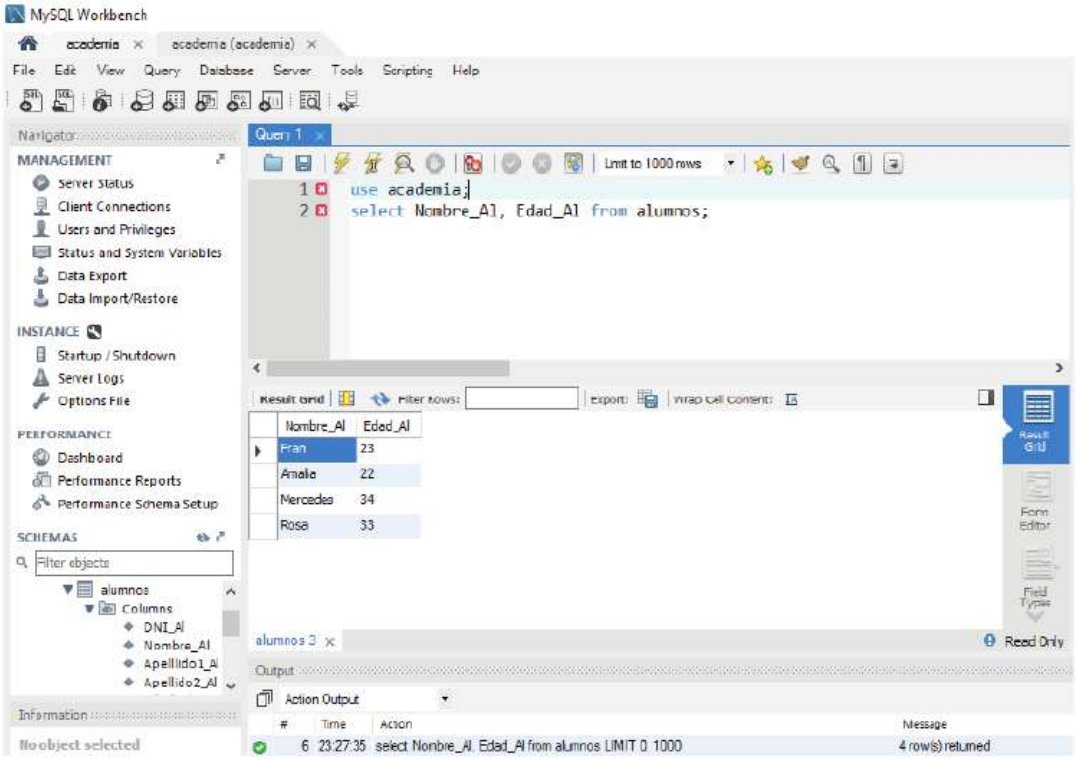


Figura 8.76. *Consulta de los alumnos de la base de datos mediante una consulta SQL ejecutada en MySQL Workbench*

Tanto para la creación de formularios como, posteriormente, para los informes, debemos aclarar que el proceso de elaboración, edición y uso es exactamente igual, siendo totalmente independiente de que se trabaje sobre tablas de Access o sobre tablas importadas o vinculadas desde fuentes externas.

Así pues, no nos vamos a detener en su elaboración, puesto que escapa de los objetivos de este libro y podemos documentarnos fácilmente a través de los muchos manuales *online* y *offline* que existen de Microsoft Access.

De esta forma, y con una intención eminentemente práctica, vamos a llevar a cabo un ejercicio guiado para cada uno de los siguientes apartados, en los que se creen formularios e informes. En ambos ejercicios guiados, vamos a partir del asistente y, si fuese preciso, se completará su edición (para añadirle nueva funcionalidad o cambios de diseño) de forma manual, a través de las opciones *Diseño del formulario* y *Diseño de informe* respectivamente (siempre que sea necesario).

8.6.1 CREACIÓN DE FORMULARIOS

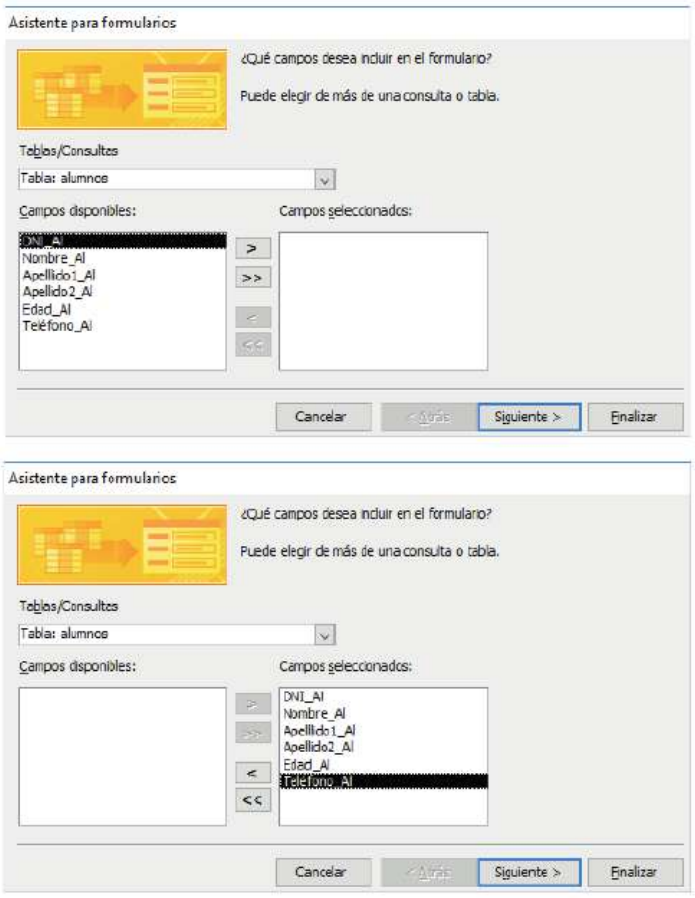
EJERCICIO GUIADO 8.3

Crearemos un formulario para gestionar los datos de los alumnos que contenga la siguiente funcionalidad: alta, baja y modificación de alumnos, así como cierre del formulario.

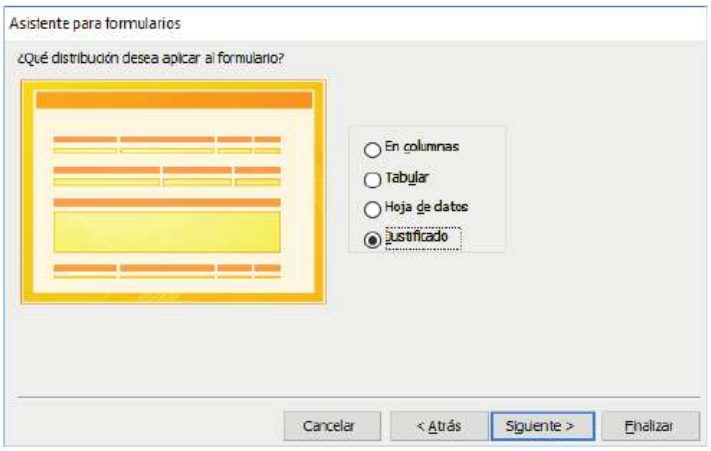
El primer paso será la creación de un formulario que muestre los datos de los alumnos, a través del *Asistente para formularios:*

Entraremos en un asistente que nos irá planteado una serie de interrogantes para configurar el formulario. De este modo, se intercambiarán los siguientes cuadros de diálogo:

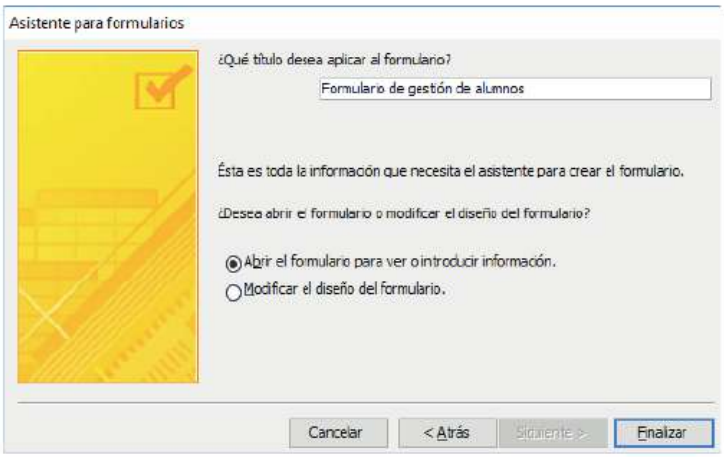
1. Selección de la/s tabla/s a utilizar y de los campos.



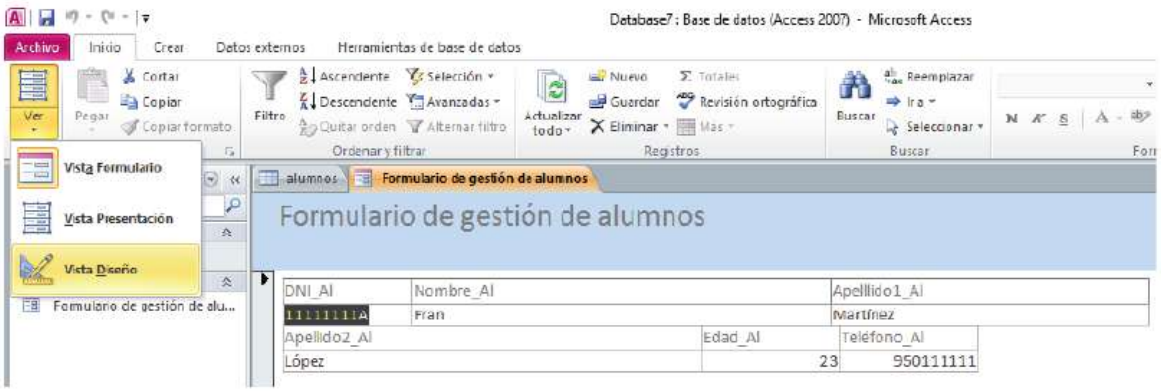
2. Elección de la distribución de los datos en el formulario.



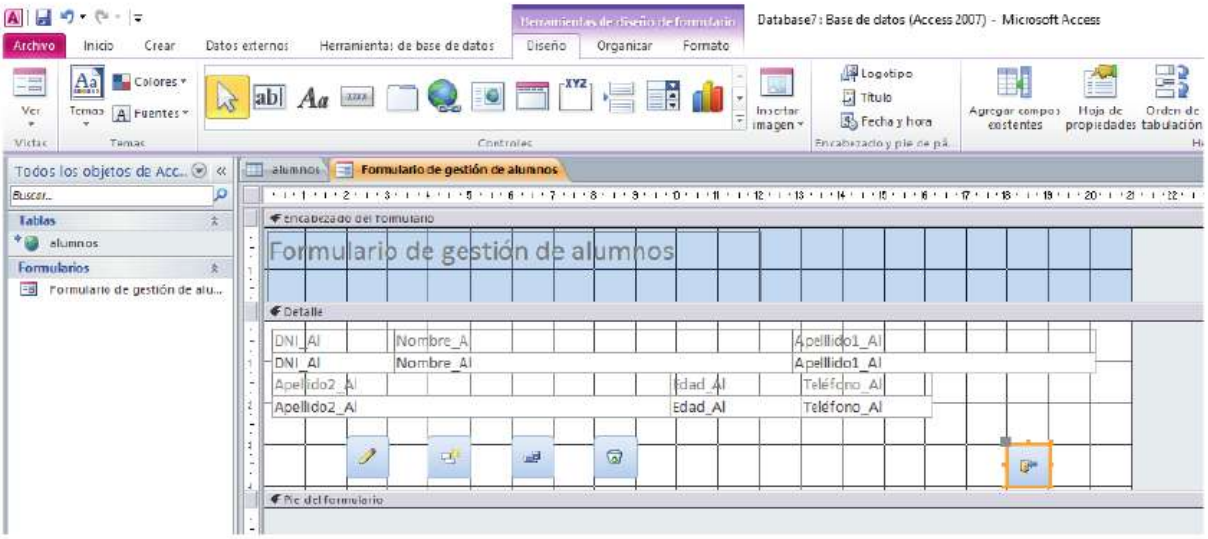
3. Definición del título del formulario y selección de si se quiere visualizar o se quiere editar el formulario manualmente.



Una vez llevados a cabo estos sencillos pasos, ya tendremos a nuestra disposición una primera versión del formulario.



Para añadir la operativa indicada, entraremos en *Vista de Diseño* y añadiremos sendos botones con dicha funcionalidad.



Finalmente, este será el resultado de la creación del nuevo formulario:

8.6.2 CREACIÓN DE INFORMES

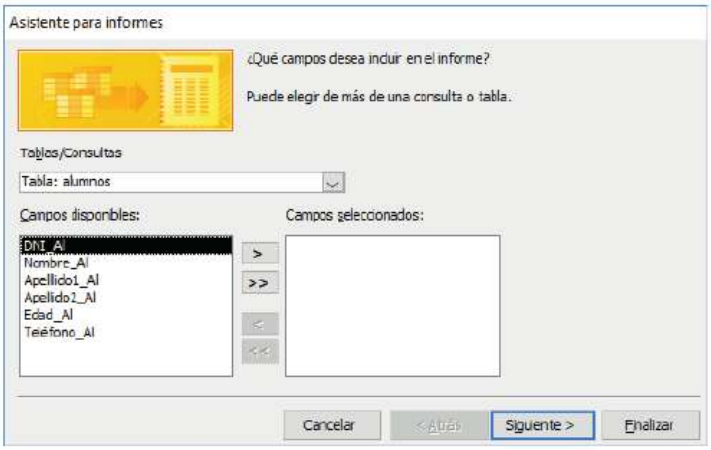
EJERCICIO GUIADO 8.4

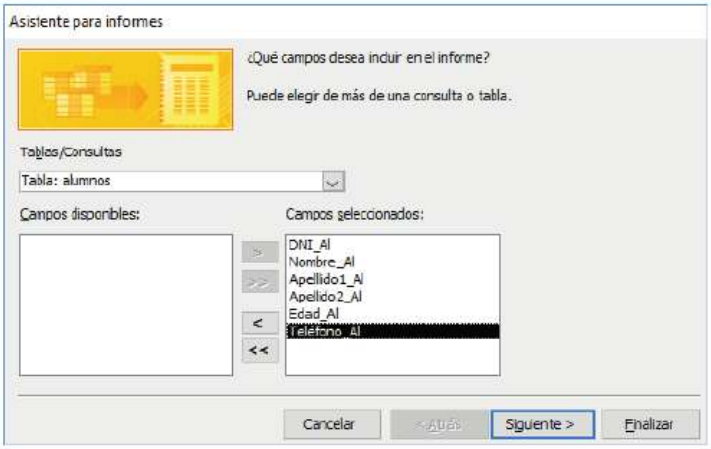
Crearemos un informe para listar todos los *Alumnos* ordenados alfabéticamente.

El primer paso será la creación de un informe que muestre los datos de los alumnos, a través del *Asistente para informes:*

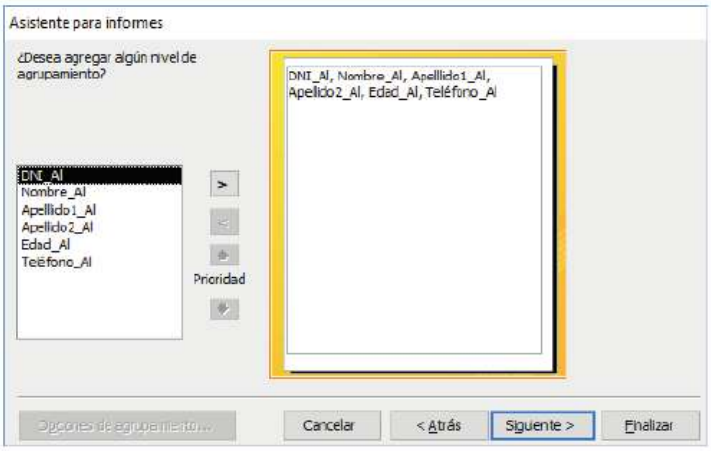
Entraremos en un asistente que nos irá planteado una serie de interrogantes para configurar el informe. De este modo, se intercambiarán los siguientes cuadros de diálogo:

1. Selección de la/s tabla/s a utilizar y de los campos.

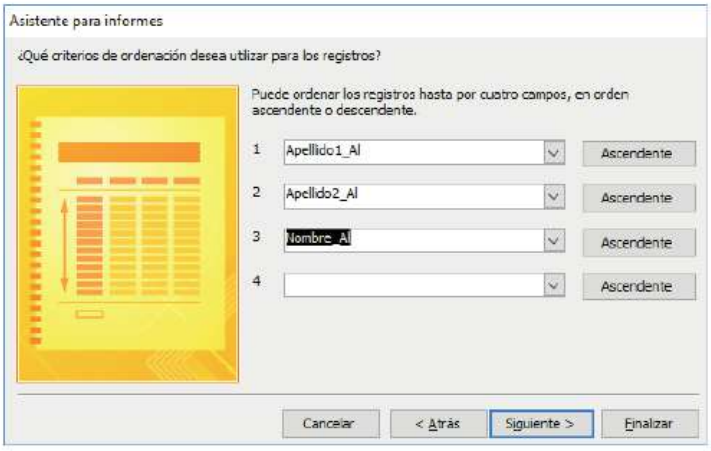




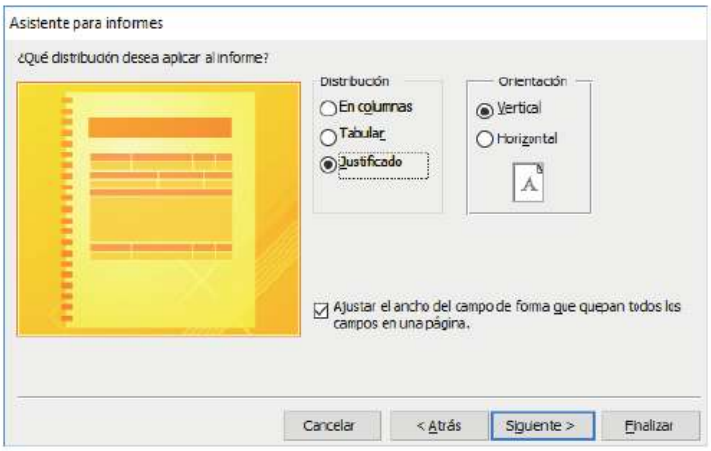
2. Definición de uno o varios niveles de agrupamientos (en nuestro caso no agruparemos).



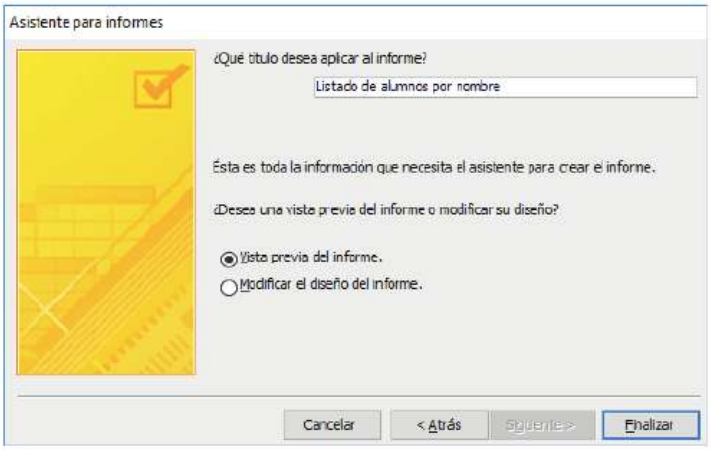
3. Definición del orden en el que aparecerán los registros en el informe (en nuestro caso el alfabético en relación al nombre y apellidos del alumno).



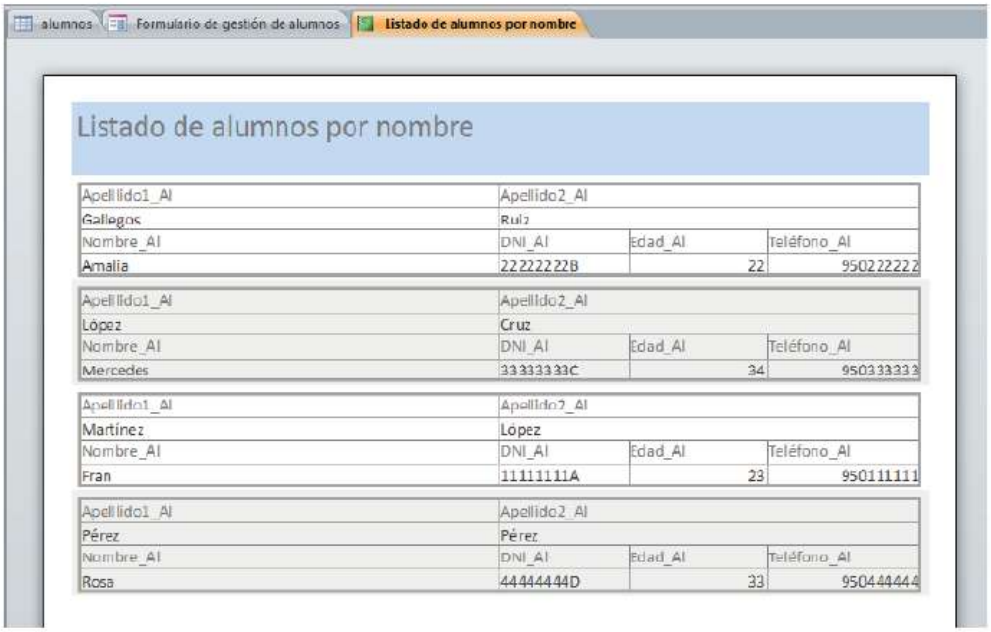
4. Elección de la distribución de los datos en el informe y la orientación del mismo (vertical u horizontal).



5. Definición del título del informe y selección de si se quiere visualizar o se quiere editar el informe manualmente.



Una vez llevados a cabo estos sencillos pasos, ya tendremos a nuestra disposición una primera versión del informe con la que, en este caso, nos quedaremos.



8.7 TÉCNICAS PARA EL CONTROL DE LA EJECUCIÓN DE TRANSACCIONES

Dado que ya hablamos de las técnicas de control de transacciones en el capítulo anterior, vamos a enfocar este apartado centrándonos en el DBMS que estamos desarrollando en este octavo capítulo, es decir, MySQL.

Así, veremos las palabras reservadas y comandos propios de este sistema de gestión de bases de datos, junto a su sintaxis y una explicación de sus características.

INFORMACIÓN

Al igual que en diversos apartados de este capítulo, debemos decir que la información para poder desarrollar esta sección la hemos obtenido del manual de referencia que nos ofrece MySQL, en este caso, el de su versión 5.6. Por tanto, proponemos su consulta para completar y ampliar estos contenidos: *https://dev.mysql.com/doc/refman/5.6/en/*.

8.7.1 COMANDOS START TRANSACTION, COMMIT Y ROLLBACK

Antes de comenzar a desarrollar estos tres comandos, debemos aclarar que, por defecto, MySQL se ejecuta en modo *AUTOCOMMIT*, es decir, cuando ejecuta comandos que modifican una tabla, dichos cambios son almacenados en disco.

No obstante, cuando se usan tablas transaccionales, los desarrolladores en MySQL pueden optar por desactivar el modo *AUTOMMIT* mediante la siguiente instrucción:

```
SET AUTOMMIT = 0;
```

Será en este momento cuando se tengan en cuentas las palabras reservadas que encabezan la sección, siguiendo la siguiente *sintaxis*:

```
START TRANSACTION;

Instrucciones;
COMMIT | ROLLBACK;
```

La funcionalidad de estas tres palabras reservadas es la siguiente:

- **START TRANSACTION:** mantiene desactivada la opción de *AUTOCOMMIT* hasta el final de la transacción.
- **COMMIT:** permite almacenar los cambios en disco correspondientes a la transacción que se inició con *START TRANSACTION*.
- **ROLLBACK:** hace que se ignoren los cambios realizados desde el comienzo de la transacción.

NOTA

Los comandos **BEGIN** y **BEGIN WORK** son sinónimos que el anteriormente comentado ***START TRANSACTION.***

NOTA

Como excepción a lo comentado en este apartado, tenemos que añadir que los siguientes comandos (o sus sinónimos) hacen concluir una transacción almacenando los cambios producidos en disco (es decir, como si se utilizara un *COMMIT*):

- ALTER TABLE
- *CREATE TABLE*
- *DROP TABLE*
- *RENAME TABLE*
- *TRUNCATE TABLE*
- *CREATE DATABASE*
- *DROP DATABASE*
- *CREATE INDEX*
- *DROP INDEX*
- *BEGIN*
- *LOAD MASTER DATA*
- *SET AUTOCOMMIT = 1*
- *LOCK TABLES*
- *UNLOCK TABLES* (si existe previamente alguna tabla bloqueada)
- *START TRANSACTION*

8.7.2 COMANDOS SAVEPOINT Y ROLLBACK TO SAVEPOINT

Estos comandos son utilizados únicamente para las tablas transaccionales *InnoDB* en MySQL, bajo la siguiente *sintaxis*:

SAVEPOINT *nombre_punto*;

Instrucciones;

ROLLBACK TO SAVEPOINT *nombre_punto*;

Su funcionalidad es la siguiente:

- **SAVEPOINT:** crea un punto, identificado por un *nombre_punto*, dentro de una transacción.
- **ROLLBACK TO SAVEPOINT:** deshace una transacción hasta el punto indicado por el *nombre_punto*, deshaciendo las modificaciones entre las dos palabras reservadas.

8.7.3 COMANDOS LOCK TABLES Y UNLOCK TABLES

Mientras que la instrucción **LOCK TABLES** bloquea tablas durante el transcurso del flujo en el que se ejecuta dicha instrucción, **UNLOCK TABLES** libera cualquier bloqueo realizado en dicho flujo actual.

También es necesario tener en cuenta las siguientes afirmaciones:

- Todas las tablas bloqueadas por el flujo actual se liberan implícitamente cuando se realiza otro *LOCK TABLES*, o cuando la conexión con el servidor se cierra.
- Un bloqueo de tabla protege solo contra lecturas inapropiadas o escrituras de otros clientes.

8.8 OPTIMIZACIÓN DE CONSULTAS

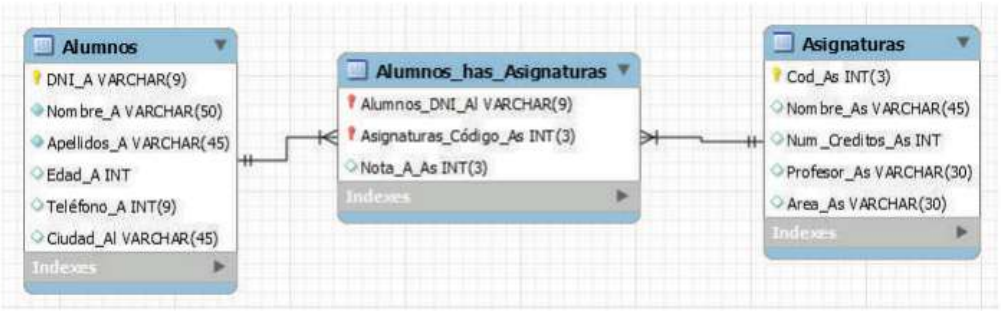
RECUERDA

Este apartado ya lo hemos estudiado en el apartado 7.13. "Procesamiento y optimización de consultas".

8.9 EJERCICIOS RESUELTOS

1. Utilizando la herramienta MySQL Workbench (concretamente su editor gráfico para diseño de bases de datos) vamos a diseñar las tablas especificadas en el Ejercicio resuelto 1 del capítulo 7). Posteriormente, exportaremos el *script* SQL correspondiente.

El diseño en MySQL Workbench quedaría de la siguiente forma:



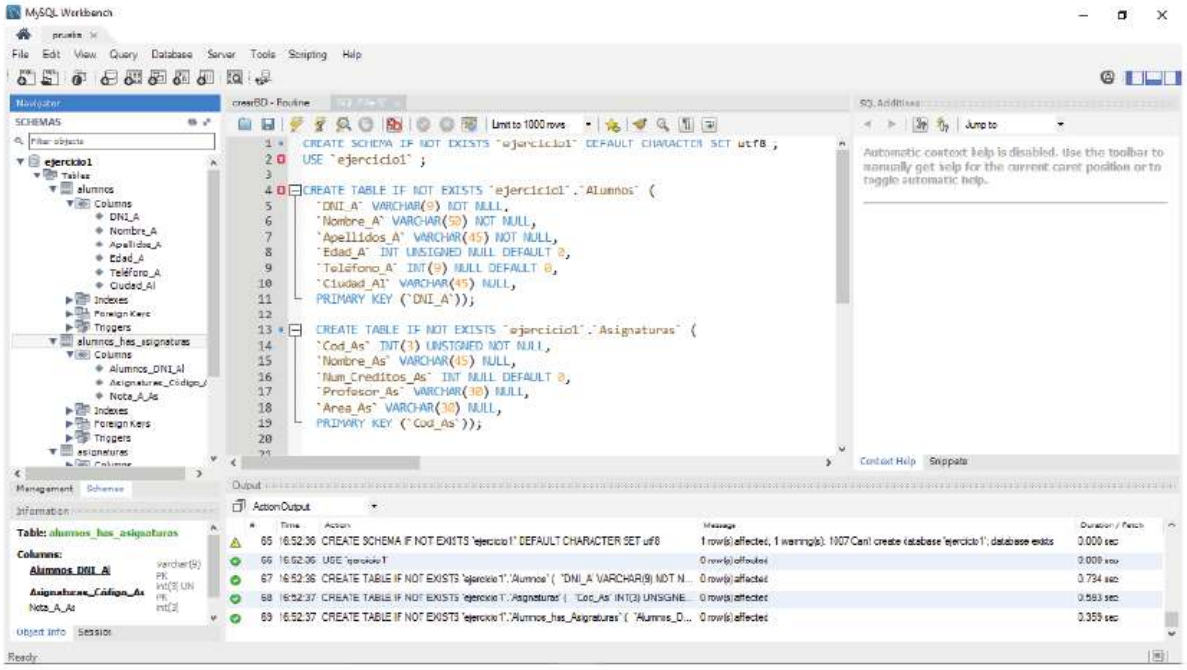
Mientras que el *script* SQL resultado de exportar el diseño sería:

```
-- MySQL Script generated by MySQL Workbench
-- 10/09/16 16:08:20
-- Model: New Model    Version: 1.0
-- MySQL Workbench Forward Engineering

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema academia
-- -----------------------------------------------------

-- -----------------------------------------------------
-- Schema academia
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `academia` DEFAULT CHARACTER SET utf8 ;
USE `academia` ;

-- -----------------------------------------------------
-- Table `academia`.`Alumnos`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `academia`.`Alumnos` (
  `DNI_A` VARCHAR(9) NOT NULL,
  `Nombre_A` VARCHAR(50) NOT NULL,
  `Apellidos_A` VARCHAR(45) NOT NULL,
  `Edad_A` INT UNSIGNED NULL DEFAULT 0,
  `Teléfono_A` INT(9) NULL DEFAULT 0,
  `Ciudad_Al` VARCHAR(45) NULL,
  PRIMARY KEY (`DNI_A`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `academia`.`Asignaturas`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `academia`.`Asignaturas` (
  `Cod_As` INT(3) UNSIGNED NOT NULL,
  `Nombre_As` VARCHAR(45) NULL,
  `Num_Creditos_As` INT NULL DEFAULT 0,
  `Profesor_As` VARCHAR(30) NULL,
  `Area_As` VARCHAR(30) NULL,
  PRIMARY KEY (`Cod_As`))
ENGINE = InnoDB;
```

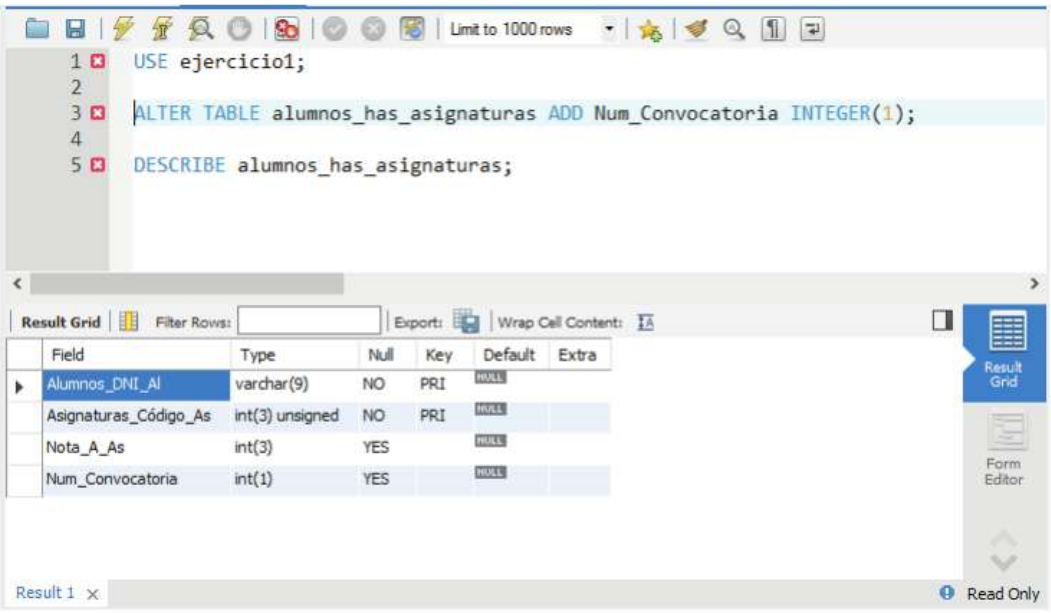
```
47  -- -----------------------------------------------------
48  -- Table `academia`.`Alumnos_has_Asignaturas`
49  -- -----------------------------------------------------
50  CREATE TABLE IF NOT EXISTS `academia`.`Alumnos_has_Asignaturas` (
51    `Alumnos_DNI_Al` VARCHAR(9) NOT NULL,
52    `Asignaturas_Código_As` INT(3) UNSIGNED NOT NULL,
53    `Nota_A_As` INT(3) NULL,
54    PRIMARY KEY (`Alumnos_DNI_Al`, `Asignaturas_Código_As`),
55    INDEX `fk_Alumnos_has_Asignaturas_Asignaturas1_idx` (`Asignaturas_Código_As` ASC),
56    INDEX `fk_Alumnos_has_Asignaturas_Alumnos1_idx` (`Alumnos_DNI_Al` ASC),
57    CONSTRAINT `fk_Alumnos_has_Asignaturas_Alumnos1`
58      FOREIGN KEY (`Alumnos_DNI_Al`)
59      REFERENCES `academia`.`Alumnos` (`DNI_A`)
60      ON DELETE NO ACTION
61      ON UPDATE NO ACTION,
62    CONSTRAINT `fk_Alumnos_has_Asignaturas_Asignaturas1`
63      FOREIGN KEY (`Asignaturas_Código_As`)
64      REFERENCES `academia`.`Asignaturas` (`Cod_As`)
65      ON DELETE NO ACTION
66      ON UPDATE NO ACTION)
67  ENGINE = InnoDB;
68
69  USE `academia` ;
70
71  -- -----------------------------------------------------
72  -- routine1
73  -- -----------------------------------------------------
74
75  DELIMITER $$
76  USE `academia`$$
77  $$
78
79  DELIMITER ;
80
81  SET SQL_MODE=@OLD_SQL_MODE;
82  SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
83  SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
84
```

2. Utilizando la herramienta MySQL vamos a probar las consultas que implementamos en el Ejercicio resuelto 1 del capítulo 7, mostrando sus resultados.

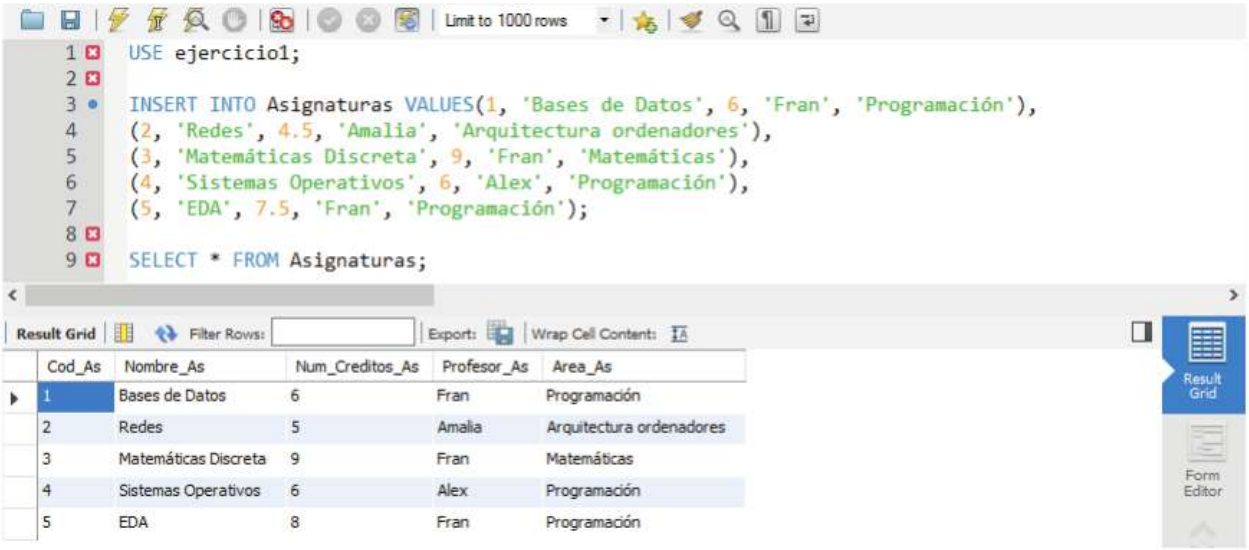
En primer lugar, crearemos una conexión y las tres tablas, escribiendo el *script* SQL que hemos exportado:



- **Añadir a la tabla *Estudian* la columna *Num_Convocatoria* de tipo entero.**



- **Insertar cinco alumnos (de los cuales algunos tengan la misma edad), cinco asignaturas (que compartan profesores y donde haya un área de programación) y diez relaciones en la tabla *Estudian*.**



```
USE ejercicio1;

INSERT INTO alumnos_has_asignaturas VALUES('11111111A', 1, 10, 1), ('11111111A', 3, 5.6, 3),
('22222222B', 2, 6, 3), ('22222222B', 1, 8.4, 2), ('33333333C', 5, 7, 1),
('44444444D', 4, 9, 2), ('44444444D', 5, 5.6, 5), ('44444444D', 1, 7, 2),
('55555555E', 3, 6, 2), ('55555555E', 4, 8.9, 1);

SELECT * FROM alumnos_has_asignaturas;
```

Result Grid

Alumnos_DNI_Al	Asignaturas_Código_As	Nota_A_As	Num_Convocatoria
11111111A	1	10	1
11111111A	3	6	3
22222222B	1	8	2
22222222B	2	6	3
33333333C	5	7	1
44444444D	1	7	2
44444444D	4	9	2
44444444D	5	6	5
55555555E	3	6	2
55555555E	4	9	1
NULL	NULL	NULL	NULL

- **Subir un punto a todos los *Alumnos* de 18 años en *Asignaturas* del *Área_As* de *"Programación"*.**

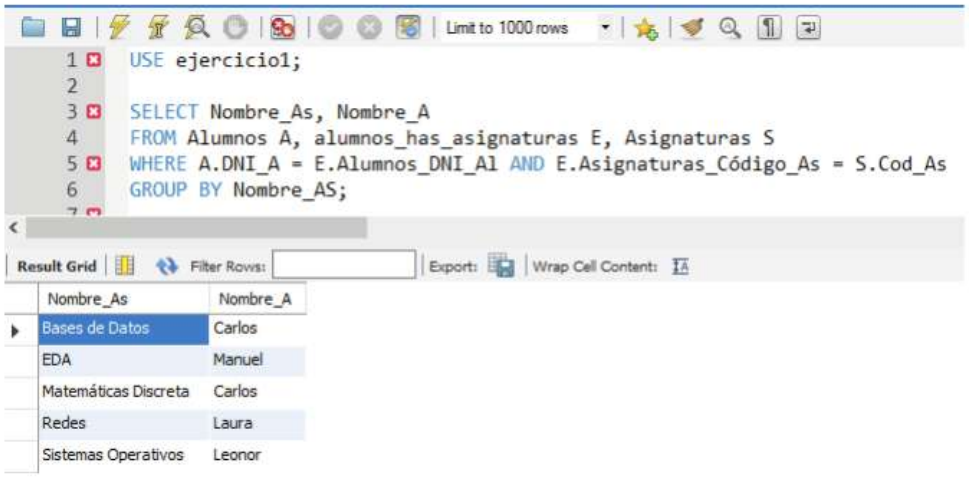
```
USE ejercicio1;

UPDATE alumnos_has_asignaturas SET Nota_A_As = Nota_A_As + 1
WHERE Alumnos_DNI_Al IN (SELECT DNI_A FROM Alumnos WHERE Edad_A = 18)
AND Asignaturas_Código_As IN (SELECT Cod_As FROM Asignaturas WHERE Area_As = 'Programación')
AND Nota_A_As < 10;

SELECT Nombre_A, Nota_A_As FROM Alumnos A, alumnos_has_asignaturas S
WHERE A.DNI_A = S.Alumnos_DNI_Al;
```

- **Eliminar los *Alumnos* cuyo *Nombre_A* contengan la "r".**

```
USE ejercicio1;
DELETE FROM Alumnos WHERE Nombre_A LIKE '%r%' OR Nombre_A LIKE '%R%';
SELECT Nombre_A FROM Alumnos;
```

A continuación, llevaremos a cabo las consultas sobre los datos originales que insertamos en las primeras instrucciones:

- **Mostrar los *Alumnos* matriculados en cada *Asignatura*.**



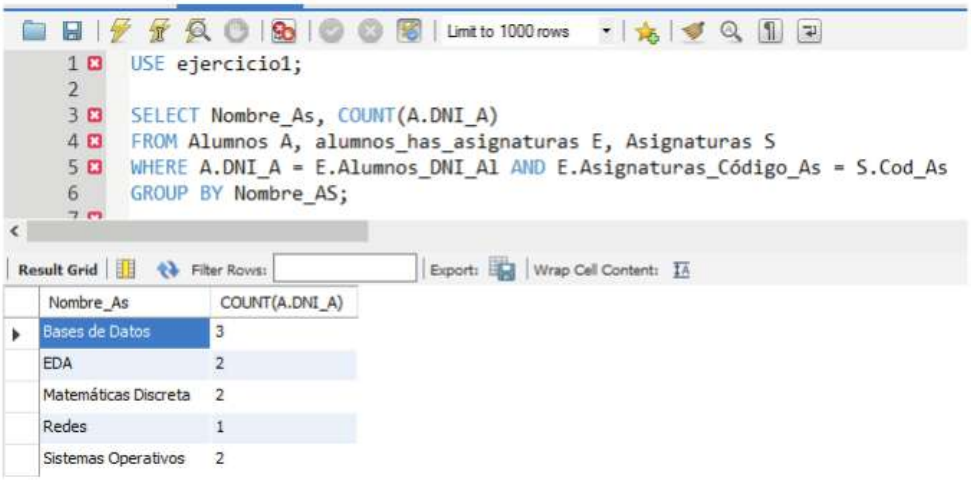
```
USE ejercicio1;

SELECT Nombre_As, Nombre_A
FROM Alumnos A, alumnos_has_asignaturas E, Asignaturas S
WHERE A.DNI_A = E.Alumnos_DNI_Al AND E.Asignaturas_Código_As = S.Cod_As
GROUP BY Nombre_AS;
```

Result Grid

Nombre_As	Nombre_A
Bases de Datos	Carlos
EDA	Manuel
Matemáticas Discreta	Carlos
Redes	Laura
Sistemas Operativos	Leonor

Podemos observar que MySQL Workbench no muestra adecuadamente los resultados de un *GROUP BY*, aunque si ejecuta la consulta correctamente.

Podemos probar esto mediante la siguiente consulta: "Mostrar el número de alumnos matriculados en cada asignatura"



```
USE ejercicio1;

SELECT Nombre_As, COUNT(A.DNI_A)
FROM Alumnos A, alumnos_has_asignaturas E, Asignaturas S
WHERE A.DNI_A = E.Alumnos_DNI_Al AND E.Asignaturas_Código_As = S.Cod_As
GROUP BY Nombre_AS;
```

Result Grid

Nombre_As	COUNT(A.DNI_A)
Bases de Datos	3
EDA	2
Matemáticas Discreta	2
Redes	1
Sistemas Operativos	2

- **Mostrar los *Alumnos* matriculados en cada *Asignatura* pero mostrando solo las *Asignaturas* que tengan más de un *Alumno* (volveremos a mostrar el número de alumnos).**

```
USE ejercicio1;

SELECT Nombre_As, COUNT(A.DNI_A)
FROM Alumnos A, alumnos_has_asignaturas E, Asignaturas S
WHERE A.DNI_A = E.Alumnos_DNI_Al AND E.Asignaturas_Código_As = S.Cod_As
GROUP BY Nombre_AS
HAVING COUNT(A.DNI_A)>1;
```

Result Grid

Nombre_As	COUNT(A.DNI_A)
Bases de Datos	3
EDA	2
Matemáticas Discreta	2
Sistemas Operativos	2

- **Mostrar el nombre de las *Asignaturas* con mayor número de créditos.**

```
USE ejercicio1;

SELECT Nombre_As
FROM Asignaturas
WHERE Num_Creditos_As = (SELECT MAX(Num_Creditos_As) FROM Asignaturas);
```

Result Grid

Nombre_As
Matemáticas Discreta

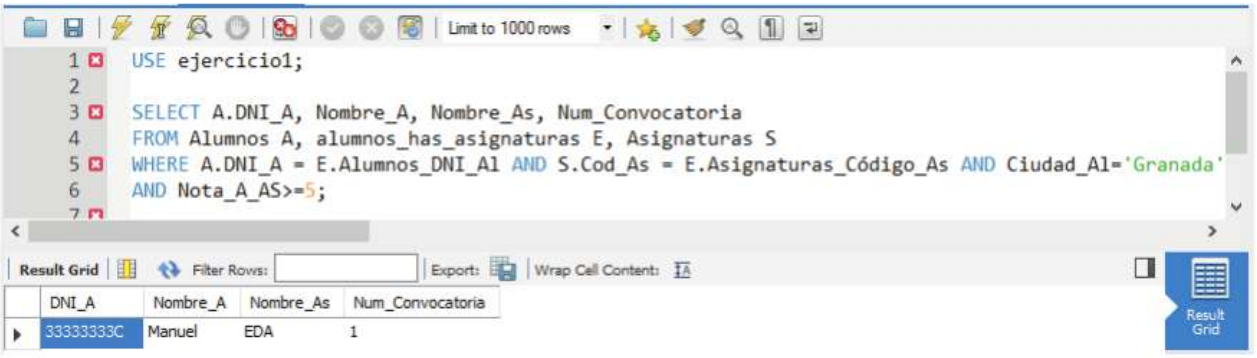
- **Mostrar la *Nota_A_As* media de cada *Alumno*.**

```
USE ejercicio1;

SELECT A.DNI_A, Nombre_A, AVG(Nota_A_As) AS 'Nota Media'
FROM Alumnos A, alumnos_has_asignaturas E
WHERE A.DNI_A = E.Alumnos_DNI_Al
GROUP BY DNI_A;
```

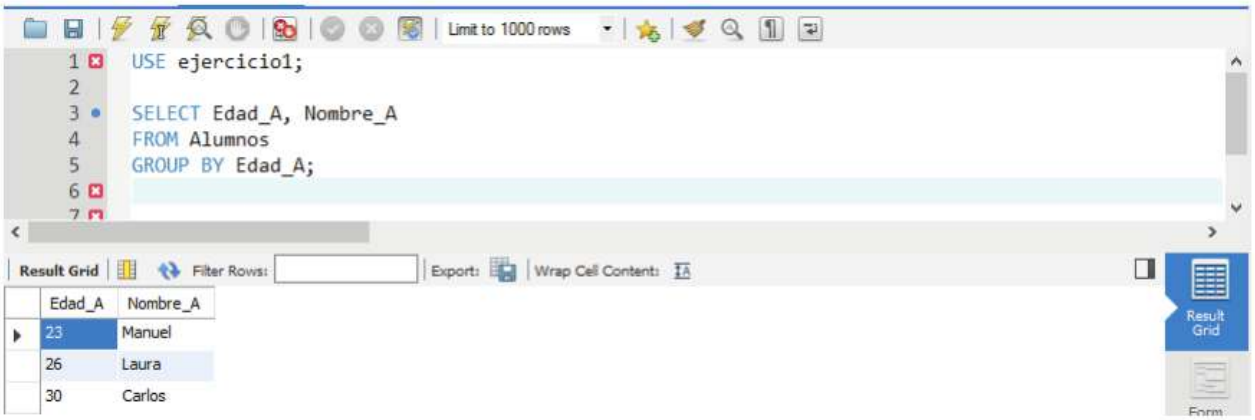
Result Grid

DNI_A	Nombre_A	Nota Media
11111111A	Carlos	8.0000
22222222B	Laura	7.0000
33333333C	Manuel	7.0000
44444444D	Leonor	7.3333
55555555E	Pepe	7.5000

- **Mostrar el número de convocatoria en la que aprobaron los alumnos que viven en Granada.**

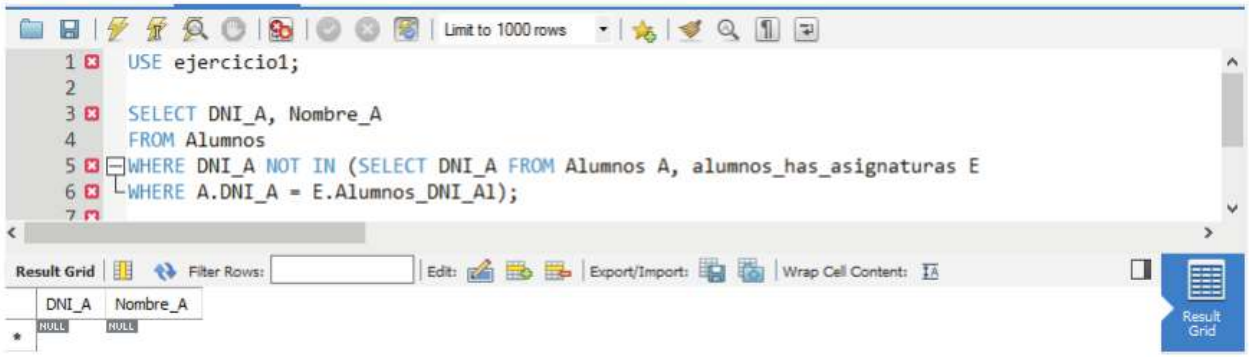


- **Mostrar el *Nombre_A* de los *Alumnos* por grupo de *Edad_A*.**



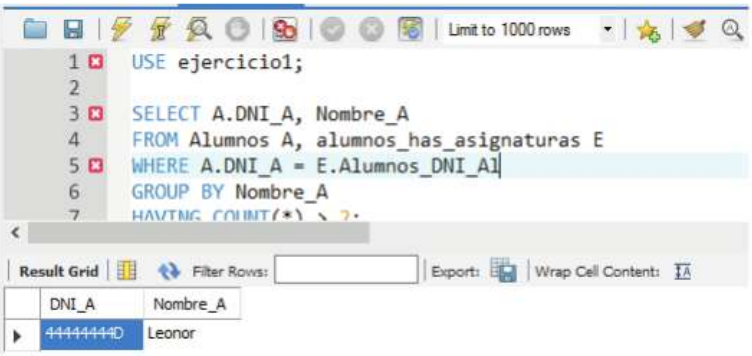
Como ya hemos comentado antes, el resultado del *GROUP BY* no se muestra adecuadamente.

- **Mostrar los alumnos que no están matriculados en ninguna asignatura.**



Lo cual significa que todos los alumnos están matriculados de alguna asignatura.

- **Mostrar los alumnos que están matriculados en más de dos asignaturas.**



8.10 EJERCICIOS PROPUESTOS

1. Utilizando la herramienta MySQL Workbench (concretamente su editor gráfico para diseño de bases de datos) se pide diseñar las tablas especificadas en el Ejercicio propuesto 1 del capítulo 7). Posteriormente, exportaremos el *script* SQL correspondiente.

2. Utilizando la herramienta MySQL se pide probar las consultas que implementamos en el Ejercicio propuesto 2 del capítulo 7, mostrando sus resultados.

3. Probar las consultas del Ejercicio resuelto 2 y del Ejercicio propuesto 2 con un depurador de código, por ejemplo Debugger for MySQL.

4. Implementa un procedimiento que seleccione los *Nombre_A* de Alumnos del Ejercicio propuesto 1 y muestre el resultado de su concatenación. Prueba la implementación tanto en MySQL Workbench como en Debugger for MySQL.

5. Vincular la base de datos del Ejercicio resuelto 1 con Microsoft Access y realizar los formularios e informes que creas convenientes.

6. Vincular la base de datos del Ejercicio propuesto 1 con OpenOffice Database y realizar los formularios e informes que creas convenientes.

8.11 TEST DE CONOCIMIENTOS

1 Selecciona cuál no es una característica generalizada de los IDE:
- **a)** Son multiplataforma.
- **b)** Poseen reconocimiento de sintaxis.
- **c)** Poseen herramientas de ayuda interactiva a través de Internet.
- **d)** Poseen depuradores.

2 Selecciona cuál no es una característica generalizada de los IDE:
- **a)** Tienen integrados los *framework* más populares.
- **b)** Tienen integrados editores para la implementación del entorno gráfico de las aplicaciones a desarrollar.
- **c)** Ofrecen *pluggins* con extensiones y nueva funcionalidad.
- **d)** Ofrecen múltiples idiomas para su entorno.

3 Los componentes más generalizados de un IDE son:
- **a)** Editor de texto, editor de texto web, compilador, intérprete, depurador, control de versiones, interfaz gráfica y refactorización
- **b)** Editor de texto, compilador, intérprete y depurador
- **c)** Editor de texto, editor de texto web, compilador, intérprete, depurador, control de versiones e interfaz gráfica
- **d)** Editor de texto, compilador, intérprete, depurador, control de versiones, interfaz gráfica y refactorización

4 El editor de texto jEdit, ¿bajo qué plataforma opera?:
- **a)** Windows
- **b)** Linux
- **c)** Unix
- **d)** Todas las respuestas son ciertas.

5 El editor de texto *Notepad++*, ¿bajo qué plataforma opera?:
- **a)** Windows
- **b)** Linux
- **c)** Unix
- **d)** Todas las respuestas son ciertas.

6 El editor de texto Vi, ¿bajo qué plataforma opera?:
- **a)** Windows
- **b)** Linux
- **c)** Unix
- **d)** Todas las respuestas son ciertas.

7 Las fases por las que evoluciona un compilador al evaluar la veracidad de un programa son:
- **a)** Análisis léxico, análisis sintáctico y análisis semántico
- **b)** Análisis léxico, análisis semántico y análisis sintáctico
- **c)** Análisis sintáctico, análisis léxico y análisis semántico
- **d)** Ninguna de las anteriores

8 Señala la afirmación incorrecta:
- **a)** Una GUI actúa de interfaz de usuario proporcionando un entorno visual sencillo con el que interactuar con el ordenador.
- **b)** La refactorización consiste en reestructurar el código que ha programado el desarrollador sin modificar la funcionalidad final de dicho código.
- **c)** Las versiones que suele contemplar un control de versiones son *alfa*, *beta* y *rtm*.
- **d)** Los IDE no dan soporte a todos los lenguajes de programación.

9 La principal diferencia de un IDE en el entorno de la base de datos con respecto a otro de propósito general es:
- **a)** Que posee una herramienta para diseñar la base de datos.
- **b)** Que su editor está optimizado para las consultas SQL.
- **c)** Que el compilador está optimizado para las consultas SQL.
- **d)** Que su control de versiones tiene en cuenta las versiones de la base de datos y no del código fuente.

10 Para instalar un producto de MySQL, usaremos la herramienta:
- **a)** MySQL Server
- **b)** MySQL Workbench
- **c)** MySQL Installer
- **d)** MySQL Connector

11 Un ejemplo de declaración de una variable de tipo entero es:
- **a)** SET numero = 10;
- **b)** DECLARE numero = 10;
- **c)** SELECT numero INT;
- **d)** DECLARE numero INT(5);

12 ¿Qué acción lleva a cabo este fragmento de código?:

```
DECLARE PROCEDURE newProcedure (INT num INT)
BEGIN
DECLARE i DEFAULT 2;
DECLARE res DEFAULT 0;
WHILE i < num DO
        IF (num%i) == 0 THEN
                SET res = 1;
        END IF
```

```
        SET i = i + 1;
END WHILE;
IF res == 0 THEN
        SELECT "Respuesta 1";
ELSE
        SELECT "Respuesta 2";
END IF;
END;
```

a) Calcula el factorial de un número.
b) Calcula si un número es primo.
c) Calcula el primer múltiplo de un número.
d) Calcula la sumatoria de un grupo de números.

13 En lugar de utilizar la estructura WHILE en el código de la cuestión 12, se podrá haber utilizado:
a) SWITCH
b) DO-WHILE
c) FOR
d) REPEAT

14 ¿Cómo se denomina la función que retorna el valor absoluto de un número?:
a) ACOS
b) ABSOLUT
c) TRUNCATE
d) ABS

15 Los parámetros de una función o procedimiento pueden tener los siguientes *modos*:
a) IN, OUT
b) IN, OUT, INOUT
c) IN, OUT, RETURN
d) IN, OUT, INOUT (solo para procedimientos), RETURN (solo para funciones)

Bibliografía

REFERENCIAS BIBLIOGRÁFICAS

Berzal Galiano, Fernando: *El ciclo de vida de un sistema de información.* Apuntes de la asignatura "Diseño de Bases de Datos".

Codd, E. F. (IBM Research Laboratory, San Jose, California): "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM*. Editorial P. Baxendale. Volumen 13, nº 6, pp. 377-387.

Elmasri, Ramez A. y Navathe, Shamkant B: *Fundamentos de Sistemas de Bases de Datos (3ª edición).* Editorial Addison-Wesley, 2002.

Ferrer Martínez, Juan: *Aplicaciones Web.* Editorial Ra-Ma. Madrid, junio de 2012.

Llanos Ferraris, Diego R: *Fundamentos de Informática y Programación en C.* Editorial Paraninfo. Madrid, 2010.López Quijado, José: *Domine PHP y MySQL (2ª edición).* Editorial Ra-Ma. Madrid, junio de 2010.

López Sanz, Marcos, Soltero Domingo, Francisco J., Sánchez Fúquene, Diana M., Moreno Pérez, Ángel, Bollati, Verónica A. y Vara Mesa, Juan Manuel: *Programación web en el entorno servidor (MF0492_3).* Editorial Ra-Ma. Madrid, 2016.

Martínez López, Francisco Javier y Ramallo Martínez, Alejandro: *Teoría, diseño e implementación de Compiladores de Lenguajes.* Editorial Ra-Ma. Madrid, 2014.

Melton, Jim y Eisenberg, Andrew: *SQL y Java. Guía para SQLJ, JDBC y tecnologías relacionadas.* Editorial Ra-Ma. Madrid, 2002.

Pressman, Roger S.: *Ingeniería del Software. Un enfoque práctico (5ª edición).* Editorial Mc Graw Hill. Madrid, 2002.

Rodríguez Yunta, Luis: *Evaluación e indicadores de calidad en bases de datos.* Revista española de documentación científica (*http:/ /redc.revistas.csic.es*) - 21, 1, 1998.

http:/ /digital.csic.es/ bitstream/ 10261/ 27342/ 1/ 594.pdf

Silberschatz, Abraham: *Fundamentos de bases de datos (5ª edición).* Editorial S.A. MCGRAW-HILL / INTERAMERICANA DE ESPAÑA, 2006.

Taylor, E. S.: *An Interim Report on Engineering Design.* Massachusetts Institute of Technology, 1959.

REFERENCIAS WEB

Enciclopedia Wikipedia

Se trata de una enciclopedia libre, políglota y editada colaborativamente. Concretamente, la hemos empleado para las siguientes referencias bibliográficas:

- Biografía de Herman Hollerith: *https://es.wikipedia.org/wiki/Herman_Hollerith*
- Las 12 reglas de Codd*: https://es.wikipedia.org/wiki/12_reglas_de_Codd*
- Definición de los tipos de restricciones de integridad:

  https://es.wikipedia.org/wiki/Integridad_de_datos
- Biografía de Ronald Fagin*: https://en.wikipedia.org/wiki/Ronald_Fagin*
- Biografía de Raymond F. Boyce*: https://en.wikipedia.org/wiki/Raymond_F._Boyce*

Portal de divulgación científica *desarrolloweb.com*

Es un portal que contiene una gran cantidad de información, manuales, monográficos, cursos formativos, etc. orientados a los programadores.

Concretamente, hemos utilizado dos artículos de su importante archivo de documentación:

- Un resumen con los tipos de datos de MySQL*: http://www.desarrolloweb.com/articulos/1054.php*
- Un artículo sobre la gestión de paquetes en DBMS Oracle:

  http://www.desarrolloweb.com/articulos/paquetes-oracle.html

Otros portales de divulgación informática

- Programacion.net*: http://programacion.net/articulo/eclipse_i_historia_y_toma_de_contacto_288*
- Hermosa programación: *http://www.hermosaprogramacion.com/2014/07/mysql-java-conectar-como/*
- Hypertextual:

  https://hipertextual.com/archivo/2013/10/mejores-editores-de-texto-para-desarrolladores/
- DataPrix*: http://www.dataprix.com/422-sentencia-explain*
- Blog de Historia de la Informática (del Museo de Informática de la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia): *http://histinf.blogs.upv.es/*
- Blog de Arsys: *http://blog.arsys.es/como-optimizar-bases-de-datos-mysql/*
- Guía de Oracle*: http://epnbdd-oracle.blogspot.com.es/2012/05/optimizacion-basica-de-sql.html*
- Riders Of The Bit: *http://ridersofthebit.net/tag/explain/*

Documentación oficial de las herramientas de MySQL

Concretamente, los siguientes manuales de referencia:

- Manual de referencia de MySQL Workbench:

  http://dev.mysql.com/doc/workbench/en/

- Manual de diferentes versiones de MySQL:
  - *https://dev.mysql.com/doc/refman/5.6/en/*
  - *http://dev.mysql.com/doc/refman/5.7/en/data-types.html*
  - *http://dev.mysql.com/doc/refman/5.7/en/server-logs.html*

Del mismo modo, también hemos hecho referencia a las zonas de descarga de *software* de la web oficial de MySQL:

http://dev.mysql.com/downloads/

Página web oficial de la herramienta Debugger for MySQL

http://mydebugger.com/

NOTA

Queremos, en este apartado, dar las gracias a Alfonso Bosh y Manuel Torres, profesores de la Universidad de Almería, por habernos transmitido el interés por las bases de datos.

Gracias por su entrega y buen hacer docente.
Además, han puesto un granito de arena importante, ya que hemos utilizado algunos de sus exámenes en diversos ejercicios, tanto resueltos como propuestos.

Solucionario de los test de conocimientos

Pregunta	Cap. 1	Cap. 2	Cap. 3	Cap. 4	Cap. 5	Cap. 6	Cap. 7	Cap. 8
1	B	B	C	D	A	A	C	C
2	D	D	D	A	B	B	B	B
3	C	A	A	D	B	C	A	D
4	A	C	B	D	C	C	C	D
5	C	C	D	A	C	D	A	A
6	C	B	A	C	D	A	A	C
7	D	B	D	A	A	B	A	A
8	B	D	C	B	C	C	A	C
9	D	D	D	C	D	A	C	A
10	A	D	C	D	C	D	B	C
11	D	D	A	C	D	C	B	D
12	D	A	B	A	B	D	D	B
13	C	D	C	D	A	A	A	D
14	C	B	A	A	D	A	B	D
15	A	B	A	C	B	A	A	B

CERTIFICADO DE PROFESIONALIDAD

PROGRAMACIÓN DE BASES DE DATOS RELACIONALES

MF0226_3

FRANCISCO JAVIER MARTÍNEZ LÓPEZ
AMALIA GALLEGOS RUIZ

Ra-Ma®
www.ra-ma.es/cp