



UF1845: Acceso a datos en aplicaciones web del entorno servidor

Certificado de Profesionalidad

IFCD0210 - Desarrollo de aplicaciones con tecnologías web



IFCD0210 > MF0492_3 > UF1845

**Acceso a datos en aplicaciones web del entorno
servidor.
IFCD0210**

Alejandro Bernabé Durán

ic editorial

Acceso a datos en aplicaciones web del entorno servidor. IFCD0210

Autor: Alejandro Bernabé Durán

1^a Edición

© IC Editorial, 2014

Editado por: IC Editorial

C.I.F.: B-92.041.839

c/ Cueva de Viera, 2, Local 3 Centro Negocios CADI

29200 ANTEQUERA, Málaga

Teléfono: 952 70 60 04

Fax: 952 84 55 03

Correo electrónico: iceditorial@iceditorial.com

Internet: www.iceditorial.com

IC Editorial ha puesto el máximo esfuerzo en ofrecer una información completa y precisa. Sin embargo, no asume ninguna responsabilidad derivada de su uso, ni tampoco la violación de patentes ni otros derechos de terceras partes que pudieran ocurrir. Mediante esta publicación se pretende proporcionar unos conocimientos precisos y acreditados sobre el tema tratado. Su venta no supone para **IC Editorial** ninguna forma de asistencia legal, administrativa ni de ningún otro tipo.

Reservados todos los derechos de publicación en cualquier idioma.

Según el Código Penal vigente ninguna parte de este o cualquier otro libro puede ser reproducida, grabada en alguno de los sistemas de almacenamiento existentes o transmitida por cualquier procedimiento, ya sea electrónico, mecánico, reprográfico, magnético o cualquier otro, sin autorización previa y por escrito de IC EDITORIAL; su contenido está protegido por la Ley vigente que establece penas de prisión y/o multas a quienes intencionadamente reprodujeren o plagiaren, en todo o en parte, una obra literaria, artística o científica.

ISBN: 978-84-16433-07-0

Nota de la editorial: IC Editorial pertenece a Innovación y Cualificación S. L.

Presentación del manual

El **Certificado de Profesionalidad** es el instrumento de acreditación, en el ámbito de la Administración laboral, de las cualificaciones profesionales del Catálogo Nacional de Cualificaciones Profesionales adquiridas a través de procesos formativos o del proceso de reconocimiento de la experiencia laboral y de vías no formales de formación.

El elemento mínimo acreditable es la **Unidad de Competencia**. La suma de las acreditaciones de las unidades de competencia conforma la acreditación de la competencia general.

Una **Unidad de Competencia** se define como una agrupación de tareas productivas específica que realiza el profesional. Las diferentes unidades de competencia de un certificado de profesionalidad conforman la **Competencia General**, definiendo el conjunto de conocimientos y capacidades que permiten el ejercicio de una actividad profesional determinada.

Cada **Unidad de Competencia** lleva asociado un **Módulo Formativo**, donde se describe la formación necesaria para adquirir esa **Unidad de Competencia**, pudiendo dividirse en **Unidades Formativas**.

El presente manual desarrolla la Unidad Formativa **UF1845: Acceso a datos en aplicaciones web del entorno servidor**,

perteneciente al Módulo Formativo **MF0492_3: Programación web en el entorno servidor**,

asociado a la unidad de competencia **UC0492_3: Desarrollar elementos de software en el entorno servidor**,

del Certificado de Profesionalidad **Desarrollo de aplicaciones con tecnologías web**.

Índice

Portada

Título

Copyright

Presentación del manual

Índice

Capítulo 1 Modelos de datos

1. Introducción
 2. Concepto de dato. Ciclo de vida de los datos
 3. Definición de un modelo conceptual
 4. El modelo relacional
 5. Construcción del modelo lógico de datos
 6. El modelo físico de datos. Ficheros de datos
 7. Transformación de un modelo lógico en un modelo físico de datos
 8. Herramientas para la realización de modelos de datos
 9. Resumen
- Ejercicios de repaso y autoevaluación

Capítulo 2 Sistemas de Gestión de Bases de Datos (SGBD)

1. Introducción
 2. Definición de SGBD
 3. Componentes de un SGBD. Estructura
 4. Terminología de SGDB
 5. Administración de un SGDB
 6. Gestión de transacciones en un SGBD
 7. Soluciones de SGBD
 8. Criterios para la selección de SGBD comerciales
 9. Instalación del SGBD *MySQL*
 10. Conexión a *MySQL* desde un lenguaje de servidor
 11. Resumen
- Ejercicios de repaso y autoevaluación

Capítulo 3 Lenguajes de gestión de bases de datos. El estándar SQL

1. Introducción
 2. Descripción del estándar SQL
 3. Creación de bases de datos
 4. Gestión de registros en tablas
 5. Consultas
 6. Conversión, generación y manipulación de datos
 7. Consultas múltiples. Uniones (joins)
 8. Agrupaciones
 9. Vistas
 10. Funciones avanzadas
 11. Resumen
- Ejercicios de repaso y autoevaluación

Capítulo 4 Lenguajes de marcas de uso común en el lado servidor

1. Introducción
 2. Origen e historia de los lenguajes de marcas. El estándar XML
 3. Características de XML
 4. Estructura de XML
 5. Estándares basados en XML
 6. Análisis XML
 7. Uso de XML en el intercambio de información
 8. Resumen
- Ejercicios de repaso y autoevaluación

Bibliografía

Capítulo 1

Modelos de datos

1. Introducción

El objetivo de un modelo de datos es representar un problema de la vida real de forma que pueda ser resuelto por una aplicación informática.

Para ello, debe comprenderse qué es un dato y cómo realizar modelos de datos.

Un dato es un valor que no contiene información por sí sola. Es el entorno el que le da valor. Para eso se introducen los modelos de datos, para relacionar e identificar los datos entre sí.

El objetivo de este capítulo consiste en pasar de una serie de requisitos de una aplicación a un modelo lógico que pueda ser implementado en cualquier sistema gestor de bases de datos o cualquier otro soporte físico. Normalmente, este trabajo será realizado por un analista informático, el cual pasará su trabajo a los administradores de bases de datos y programadores informáticos para que implementen el problema.

El primer paso del analista consistirá en dibujar un diagrama entidad/relación para representar de forma visual el problema a resolver. A continuación, convertirá ese diagrama en un esquema relacional, el cual deberá estar normalizado para asegurarse de que no dará problemas en el futuro.

Este esquema relacional normalizado recibe el nombre de “modelo lógico de datos”. Es esto lo que se debe enviar a los administradores de bases de datos y programadores para que implementen de forma real en su aplicación.

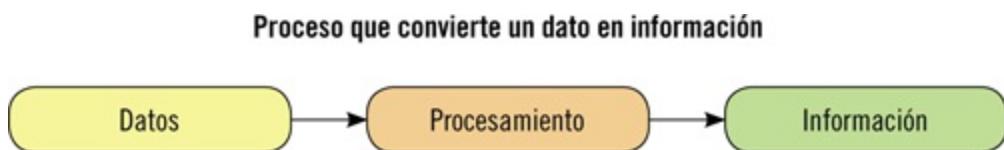
Todo esto constituye un proceso complejo porque requiere de inteligencia y experiencia. Hacer un modelo lógico de datos sencillo y que sirva para un volumen de datos pequeño es casi trivial. Lo complicado es hacer modelos lógicos complejos y que nunca den problemas en el futuro, que acepten ampliaciones y no provoquen fallos de rendimiento cuando el número de usuarios o datos aumente.

2. Concepto de dato. Ciclo de vida de los datos

Un dato es un valor. Un dato es la unidad mínima con la que trabaja un sistema de información. No confundir “sistema informático” con “sistema de información”. En un sistema informático la unidad mínima es un bit (un uno o un cero); el sistema de información se coloca en una capa superior.

Piense en un dato como en un valor: el número 23 o el texto “hola” serían perfectos ejemplos de qué es un dato.

Un dato no contiene información. Es simplemente un valor. Para que contenga información, para que sea de utilidad, **un dato debe ser procesado**.



Ejemplo

Si ve una hoja en la que aparece el número 23, no tendrá utilidad para usted. En cambio, si en esa hoja aparece escrito “Carlos tiene 23 años”, sí puede ser de utilidad para usted.

En este caso se ha procesado el dato “23” para especificar en qué consiste y dar información al usuario.

El ciclo de vida de un dato es el siguiente:

- **Creación del dato.** Estos datos pueden ser creados manualmente por un usuario o automáticamente de forma informática en base a un evento o suceso. El modelo de datos se encargará de definir la manera en la que esos datos serán almacenados y si son aceptados o no.
- **Manipulación.** Si los datos sufren modificaciones, el modelo de datos definirá la manera en la que se podrán mantener esos datos actualizados. El concepto “manipulación” incluye el consultar los datos para su uso.
- **Destrucción.** Debe ser posible destruir esos datos cuando deja de ser necesario su almacenamiento.

2.1. Tipos de datos

Como ya se ha visto anteriormente, un dato se coloca una capa por encima del sistema informático, por lo tanto, no se trata simplemente de “unos y ceros”. El número de tipos de datos es infinito, puesto que es un concepto abierto. Es el programador o el sistema el que limita qué tipos de datos utilizará.

Un dato puede ser desde el número 99 hasta un componente informático que defina el comportamiento de un programa, pasando por una cadena de caracteres que contenga la palabra “hola”; es decir, puede almacenarse cualquier dato, solo el sistema que se utilice pondrá límites.

Una vez explicado que un dato **puede ser cualquier cosa**, en los siguientes apartados se verá cómo se pueden agrupar esos datos para ser utilizados en un modelo de datos.

2.2. Básicos

Los tipos básicos de datos son los tipos elementales que un modelo de datos manejará. No pueden ser descompuestos en datos de un nivel inferior. No son datos que contienen más datos en su interior.

Algunos tipos de dato básicos son estos:

- **Números enteros:** simplemente un número sin decimales, por ejemplo, 100, 99, -30.
- **Caracteres:** por ejemplo, la palabra “hola”.
- **Números con decimales:** por ejemplo, 29'30, 3'1416.
- **Datos booleanos:** *true/false*, verdadero/falso.

Son datos aislados, no guardan relación con otros datos por sí mismos. Recuerde que un dato puede ser cualquier cosa, pero un dato básico se caracteriza porque no puede ser descompuesto en otros datos y porque no tiene relación con otros datos, como sí pueden tenerla los registros y los datos dinámicos.

2.3. Registros

Los registros son un conjunto de datos formando una estructura. Dentro de un registro puede haber datos que, a su vez, sean otras estructuras.

Un registro podría contener la información siguiente:

- Nombre: Carlos
- Edad: 42 años
- Nacionalidad: Española

El nombre de cada elemento tiene el nombre de **campo**.

Dentro de un registro puede haber un campo que sea otra estructura. El registro del ejemplo podría tener otro campo llamado “Hijos” que contuviera, a su vez, otras personas dentro.

Aplicado al campo de las bases de datos, un registro es una fila de una tabla de datos en la que todos sus elementos siguen la misma estructura.



Nota

En una base de datos, un registro no puede tener registros dentro, aunque de forma conceptual sí.

Lo importante de un registro es que es un dato que contiene campos y que siempre sigue la misma estructura.

2.4. Dinámicos

La diferencia entre los tipos de datos dinámicos y los registros es que los registros tienen una serie de campos que son fijos. Los datos dinámicos pueden variar los campos que contienen para adaptarse a las necesidades del momento.



Actividades

1. Identifique el tipo de dato:

- 30
 - {nombre: "Oliver", apellido: "Twist"}
 - 01010100111011
-

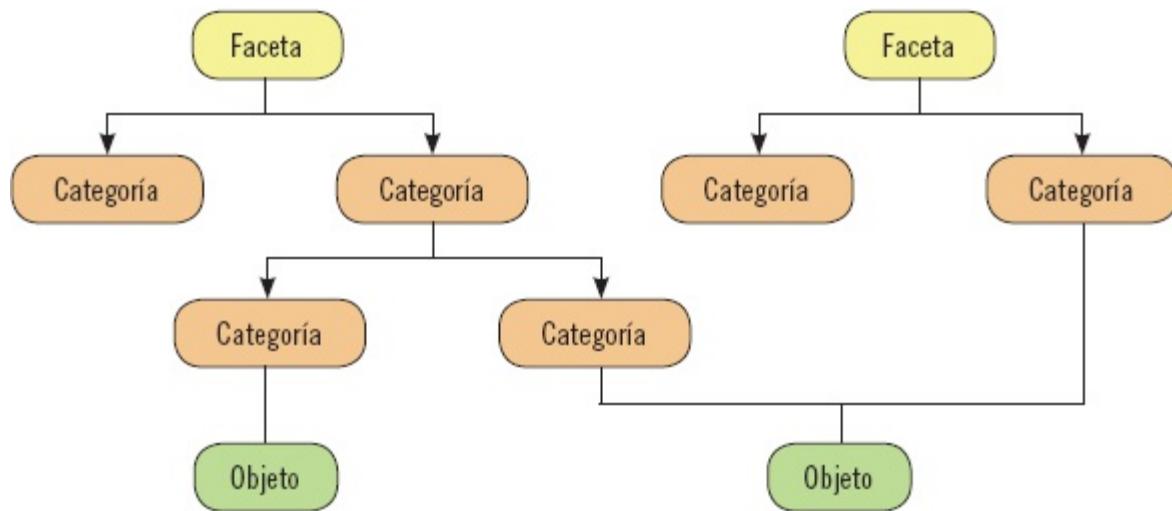
3. Definición de un modelo conceptual

Un modelo conceptual es un modelo utilizado para representar cosas. Un modelo de datos no es más que **un modelo conceptual** utilizado para **representar los datos** utilizados por una aplicación informática.

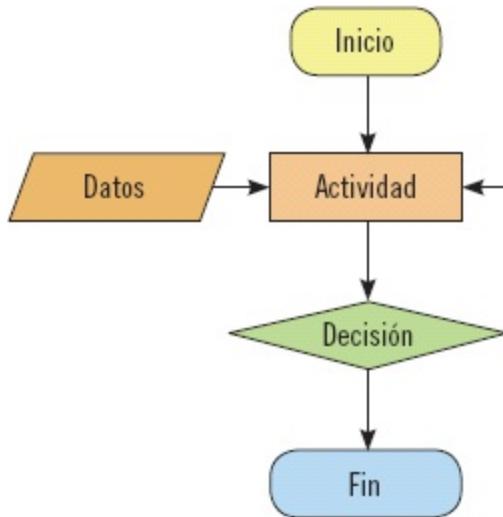
El concepto de “modelo conceptual” es muy amplio y puede utilizarse para cualquier cosa imaginable. Es simplemente una representación gráfica de algún problema o situación. Puede representar flujos de información, jerarquías, relaciones, etc. Este capítulo se concentrará en los modelos de datos, puesto que son los más importantes de cara al uso de bases de datos.

A continuación, se muestran ejemplos de modelos conceptuales.

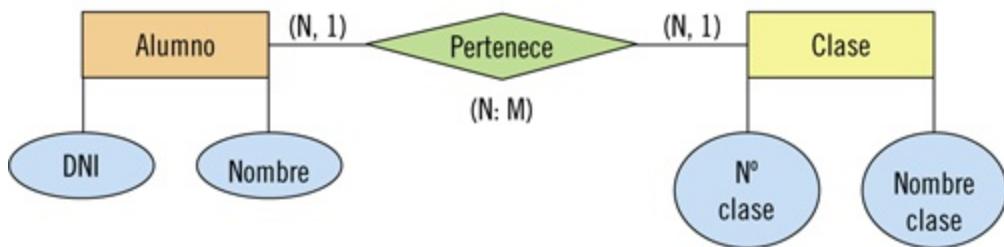
Ejemplo de modelo jerárquico



Ejemplo de diagrama de flujo



Ejemplo de modelo entidad/relación



Un **modelo de datos** es una capa que se coloca encima del sistema de información y del sistema informático para gestionar los datos utilizados en un problema de forma aislada.

Las capas inferiores se encargan de que los datos se guarden en el lugar físico correcto, de que no se pierda información, de que su gestión sea eficiente, etc. Todo este comportamiento será transparente al modelo de datos, el cual solamente se encargará de definir las necesidades de un problema concreto.

Una vez explicada su utilidad, hay que definir exactamente en qué consiste. Un modelo de datos es una **representación de objetos junto a sus propiedades y relaciones entre ellos**. Se utiliza típicamente para describir problemas del mundo real, para definir qué datos debe almacenar una aplicación y cómo se relacionan entre sí.



Nota

Un modelo de datos es útil para una aplicación que gestione los empleados de una empresa. Puede especificar la relación entre empleados, departamentos, empleados que son supervisores de otros empleados, etc. El modelo definirá qué información necesita la aplicación para funcionar y proporcionará las herramientas al usuario para almacenarla.

Además de definir qué se almacenará, también definirá restricciones para decidir qué datos serán aceptables o no, condiciones de integridad, cómo se relaciona cada objeto con los demás y qué operaciones se pueden efectuar con ellos.

Un modelo de datos se divide en dos partes, dos sublenguajes:

- **Lenguaje de definición de datos (DDL *Data Definition Language*):** lenguaje que se utiliza para escribir específicamente cómo serán los datos a almacenar en la base de datos y las condiciones que deben cumplir para ser aceptados. Además de definir relaciones entre objetos.
- **Lenguaje de Manipulación de Datos (DML *Data Manipulation Language*):** lenguaje utilizado para manipular esos datos (leerlos, añadir nuevos datos, modificarlos, borrarlos).

El modelo de datos que se utilizará mayoritariamente en este libro será el **modelo relacional**, que es el más utilizado en sistemas gestores de bases de datos. En este capítulo se tratará la parte de la definición de datos.

3.1. Patrones

Los patrones son soluciones a situaciones comunes que se producen a la hora de modelar datos.

Desde que se utilizan los modelos de datos se han creado miles de ellos por empresas y personas de todo el mundo. Por ello se han analizado miles de escenarios diferentes y aplicado muchas soluciones diferentes a los mismos problemas. Gracias a la experiencia durante años utilizando modelos de datos, se han podido determinar patrones para resolver las situaciones más comunes. Los patrones determinan cómo actuar ante diferentes situaciones, proporcionando la solución más eficiente y más extendida.

En los siguientes apartados titulados “Modelo Relacional” y “Construcción del

“modelo lógico de datos” se aplicarán patrones a la hora de crear modelos de Entidad/Relación.

3.2. Modelos genéricos

Al igual que los patrones, los modelos genéricos ofrecen soluciones a problemas comunes. Mientras que los patrones simplemente ayudan a resolver situaciones concretas, los modelos genéricos proporcionan un modelo de datos completo que cubre la mayoría de los requisitos de un problema.

En lugar de diseñar un modelo desde cero, se utiliza un modelo de datos genérico y se modifica para que se adapte a nuestro problema.

Por ejemplo, en un blog de Internet, siempre se contará con entradas, comentarios, escritores y categorías. En vez de estudiar en profundidad cada elemento y sus relaciones, puede usarse un modelo genérico que incluya todos los elementos necesarios y adaptarlo si es que hace falta.

Utilizar patrones y modelos genéricos siempre es positivo porque son técnicas desarrolladas ampliamente gracias a la experiencia durante años de ingenieros de todo el mundo, por lo que son ideas probadas y eficientes. No obstante, los requisitos de la aplicación siempre deben ir por delante de la utilización de componentes genéricos.

4. El modelo relacional

El modelo relacional es un modelo de datos utilizado para representar datos interrelacionados.

Es el modelo más utilizado en las aplicaciones de procesamiento de datos porque es muy potente al representar los datos. Fue desarrollado por Edgar Frank Codd en 1970, miembro de IBM. Está basado en el concepto de que la estructura básica es la “relación”. Los datos se almacenan en relaciones (tablas y estas, a su vez, se relacionan con otras tablas).

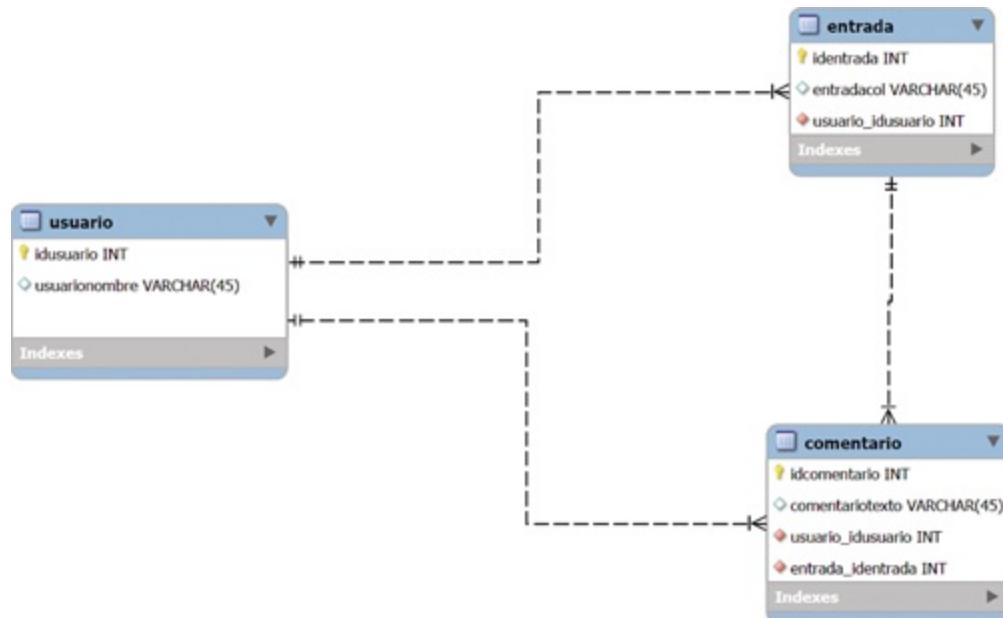
Piense en una “relación” como en un tipo de objeto. Estos datos se almacenarán en tablas y se relacionarán con otras tablas.

Para el modelo relacional suele utilizarse una representación gráfica de cada objeto y sus relaciones. De esta manera el administrador de base de datos podrá crear fácilmente la base de datos siguiendo las indicaciones creadas por el analista informático.

La representación gráfica utilizada para representar modelos relacionales son los **diagramas de entidad/relación**. Estos diagramas nos permiten leer un modelo de datos de forma visual.

Los diagramas entidad/relación no cumplen el modelo relacional a rajatabla. Habrá que realizar una serie de transformaciones para que así sea. Estos diagramas se utilizan únicamente con fines informativos para poder entender cómo implementar ese modelo de datos. El modelo relacional sí se utiliza para el almacenamiento físico de los datos.

A lo largo de este apartado se explicará el modelo relacional junto a cómo se representa cada elemento mediante diagramas de entidad/relación.



Ejemplo de un modelo entidad/relación realizado con MySQL Workbench

4.1. Descripción

El modelo relacional está basado en el modelo relacional matemático. Su objetivo es separar el almacenamiento físico y otras características propias del sistema informático de la propia lógica de los datos.



Nota

De esta manera, el usuario del modelo de datos simplemente debe preocuparse de los datos que va a utilizar su aplicación, no de cómo se organizarán internamente.

La lista de objetivos es la siguiente:

- **Independencia física de los datos:** la manera en que los datos estén organizados en el soporte físico utilizado (disco duro, disco extraíble, memoria RAM) no debe afectar en absoluto al modelo lógico utilizado. El modelo relacional añade una capa de abstracción que se encargará de manejar los datos internamente sin que el usuario tenga que definir cómo se guardarán.
- **Independencia lógica de los datos:** si se cambia la lógica del modelo datos, esto no debe afectar al sistema informático que guarde los datos.
- **Encapsulamiento del comportamiento interno:** el sistema que implemente el modelo relacional se encargará de manejar los datos, las relaciones y las restricciones internamente para hacer sencilla al usuario su utilización. Para el usuario será sencillo manipular los datos, puesto que estarán presentados mediante tablas y relaciones.

El creador del modelo relacional fue Edgar F. Codd. Tras unos años viendo cómo se implementaba su sistema en el mercado, vio que muchos usuarios no entendieron correctamente el **concepto de relación**. La mayoría simplemente almacenaba la información en tablas, pero sin unas reglas adecuadas de **normalización** y sin especificar las relaciones entre tablas correctamente. Por eso, en 1985 estableció las famosas **12 reglas de Codd** (que en realidad son 13). Estas son las reglas que cualquier sistema que implemente el modelo relacional debe seguir:

- **Regla 0:** el sistema debe ser relacional, tanto como base de datos como sistema de gestión. Un sistema de gestión de bases de datos relacional (RDBMS por sus siglas en inglés) debe utilizar sus opciones relacionales únicamente para gestionar la base de datos.
Esta regla es la regla básica y afirma que el sistema de base de datos debe contar con un sistema que gestione las relaciones de forma eficaz y este sistema relacional solamente debe ser utilizado para gestionar la información que se encuentra en la base de datos.
- **Regla 1:** regla de información. La información de la base de datos relacional estará alojada en tablas.

- **Regla 2:** regla de acceso garantizado. Todos los datos almacenados en la base de datos deben ser accesibles. Cada registro de una tabla tendrá una **clave primaria** que será un dato único que representará a ese registro en la tabla. Cada dato será accesible, especificando el nombre de la tabla en que se encuentra y el valor de su clave primaria.
- **Regla 3:** manejo sistemático de valores nulos. El sistema debe permitir a un campo tener un valor nulo (vacío). Esto será utilizado cuando un registro no disponga de un valor conocido para un campo determinado. El sistema gestor de base de datos deberá tratar estos valores de forma especial y diferente a los demás valores. Los valores nulos son independientes del tipo de dato del campo (texto, numérico, etc.) y son tratados siempre de la misma manera.
Nota: no es lo mismo que un campo tenga valor vacío a que sea nulo. Un campo podría contener el número 0 o una cadena de texto vacía y no sería nulo. Un campo nulo significa que se desconoce su contenido. En los anteriores casos sí se sabía su contenido: ningún contenido.
- **Regla 4:** catálogo disponible basado en el modelo relacional. El sistema debe proporcionar una forma de acceder al catálogo del modelo relacional (lista de tablas, campos, relaciones, etc.) utilizando el mismo lenguaje de consulta empleado para acceder a los datos. A este catálogo se le llama **diccionario de datos** y se consulta mediante SQL.
Ejemplo: será tan posible ver la lista de registros de una tabla como ver la lista de tablas que están presentes en la base de datos de una forma unificada.
- **Regla 5:** el sistema puede tener, a su vez, varios sistemas para comunicarse con la base de datos, pero debe haber un lenguaje unificado que:
 - Tenga una sintaxis lineal (que se lea de izquierda a derecha sin necesidad de saltos de línea).
 - Pueda ser utilizado de forma interactiva para comunicarse con la base de datos manualmente y también desde otras aplicaciones.
 - Soporte operaciones para definir el modelo de datos, operaciones de manipulación de datos, seguridad y restricciones de integridad además de gestión de transacciones.

Nota: el lenguaje unificado más utilizado es SQL (*Structured Query Language*), el cual se tratará en profundidad más adelante.

- **Regla 6:** la regla de las vistas actualizables. Todas las vistas que sean teóricamente modificables deben ser modificables por el sistema.
- **Regla 7:** alto nivel en alta, modificación y borrado de datos. El sistema debe ser capaz de proporcionar datos de tablas que estén recibiendo altas, bajas o modificaciones en sus registros en ese momento. Todos los registros de la tabla

deben ser accesibles en cualquier momento, no importa que se estén modificando en ese preciso instante.

- **Regla 8:** independencia física de los datos. Cambios internos en cómo están organizados físicamente los datos no deben afectar en absoluto al modelo relacional y no deben requerir cambios en la aplicación.
Ejemplo: si se actualiza el software del Sistema Gestor de Bases de Datos, el modelo relacional debe permanecer intacto y la aplicación que utilice la base de datos no debe necesitar cambios para seguir funcionando. Actualizar desde *MySQL* 5.1 a *MySQL* 5.6 no debe causar problemas (y así es).
- **Regla 9:** independencia lógica de los datos. Cambios en el modelo de datos no deben afectar a las aplicaciones que utilice la base de datos. Esto es más difícil de conseguir que la independencia física de los datos, puesto que la lógica de los datos es definida por el usuario de la base de datos y normalmente, son requeridos cambios en el modelo de datos porque también se harán cambios en las aplicaciones que utilice la base de datos. Sin embargo, siempre se tratará de mantener la estructura de tal forma que las aplicaciones no necesiten cambios para seguir funcionando.
- **Regla 10:** independencia de integridad. Las condiciones de integridad deben ser almacenadas en el catálogo de la base de datos (diccionario de datos) y ser independientes de la aplicación que la utilice. Debe ser posible cambiar las restricciones de integridad sin que eso requiera cambios en la aplicación.
- **Regla 11:** independencia de distribución. La distribución de distintas partes de la base de datos no debe afectar al usuario de la misma. Las aplicaciones que utilicen la base de datos no deben notar cambios cuando un sistema de distribución de datos sea implementado por primera vez y tampoco cuando un sistema de distribución existente añada nuevos elementos a su red.
Ejemplo: en bases de datos de uso masivo es muy habitual el concepto de replicación. Esto consiste en tener la misma base de datos en múltiples servidores para dividir el trabajo, aumentando el rendimiento. Si se añaden nuevos servidores replicados, esto no debe afectar en absoluto al funcionamiento de la base de datos más que en un aumento del rendimiento.
- **Regla 12:** regla de no subversión. Si el sistema ofrece un sistema de bajo nivel para modificar datos de la base de datos, este nunca debe poder permitir que se violen las restricciones de integridad de los datos.

A continuación, se tratará cómo representar el enunciado de un problema real en diagramas de entidad/relación. Por el momento, se realizará en papel, más adelante se enseñará cómo dibujar diagramas de entidad relación con el ordenador utilizando *MySQL Workbench*.



Actividades

2. Determine si las siguientes afirmaciones entran dentro de las reglas de Codd o no:

- Se debe contar con un sistema de copias de seguridad que se encargue de asegurar que no se pierde información en caso de fallo informático.
 - No debe ser posible violar las cláusulas de integridad bajo ningún concepto.
 - Todos los campos deben admitir contener valores nulos.
 - Debe ser posible replicar los datos en varios sistemas informáticos para aumentar el rendimiento sin que esto afecte al modelo de datos.
-

4.2. Entidades y tipos de entidades

Una entidad es **una cosa**. En el lenguaje natural una entidad es un sustantivo, puesto que representa una cosa.

En el modelo relacional, una entidad tendrá una serie de **atributos, una o varias claves** y se **relacionará** con otras entidades.

En un diagrama de entidad/relación, una entidad se representa mediante un rectángulo en cuyo interior estará escrito su nombre.

Al modelar un diagrama entidad/relación, lo primero que debe identificarse son las entidades del mismo. Para ello debe analizarse qué información es necesaria en la aplicación y qué se necesita para guardarla en la base de datos.

En este capítulo se mostrarán los ejemplos de diagrama entidad/relación mediante el programa informático *MySQL Workbench*. La notación es ligeramente diferente a la utilizada cuando se dibuja a mano un diagrama entidad/ relación, aunque solamente cambia la forma en que se representan los atributos de cada tabla, que en vez de aparecer como círculos conectados a la entidad principal, se encuentran dentro de la propia entidad.

Las entidades pueden ser **fuertes o débiles**, dependiendo de si dependen de otras entidades para existir o no:

- **Entidad fuerte:** una entidad que tiene significado por sí mismo. No necesita de otras entidades para tener sentido.
- **Entidad débil:** una entidad que depende de otras para tener sentido. Las entidades débiles se representan por un rectángulo pero con doble línea.

Dentro de las entidades débiles es posible distinguir entre **entidades débiles por dependencia** y **por identificación**. Una entidad débil por dependencia simplemente es débil porque no tiene sentido sin otra entidad, pero tiene su identificación propia. Las entidades por identificación, además, necesitan identificarse utilizando el identificador de la entidad fuerte.



Ejemplo

Una entidad débil por dependencia es una tabla de pedidos que dependa de productos pero que tenga un ID de pedido y una débil por identificación sería, por ejemplo, una tabla de libros y otra de ediciones en la que cada edición necesite el ISBN del libro y un número correlativo de edición.



Actividades

3. Identifique las entidades en el siguiente enunciado:

Se desea realizar una aplicación que gestione un campeonato de automovilismo. El campeonato estará compuesto de pilotos que pertenecerán a equipos. Se desea guardar la puntuación de cada piloto en cada circuito a lo largo del año.

4.3. Elementos de datos. Atributos

Cada entidad tiene una serie de atributos. Dependiendo de la notación empleada, se utiliza una representación gráfica u otra. En la notación Chen se usan círculos conectados a la entidad principal en cuyo interior está el nombre del atributo. En la notación de patas de gallo se listan directamente dentro del cuadro de cada entidad, debajo del nombre de la misma.

Normalmente, al realizar un modelo entidad/relación hay dos posibilidades: que solamente se quieran dejar claras las relaciones entre entidades o que se quiera tener un documento detallado que además contenga toda la información sobre cada entidad.

En el primer caso, solamente se listarán los atributos más representativos, los que se salen de lo normal o tienen más importancia. Normalmente, la lista completa de campos de cada tabla iría en el modelo lógico de datos o en un documento de texto aparte.

En el segundo, se listarán todos los campos de forma detallada. De tal forma que simplemente con un diagrama entidad/relación se pueda continuar con el proceso sin necesidad de más documentación.

4.4. Relaciones: tipos, subtipos, cardinalidad

Cada entidad se relacionará con otras entidades. Para ser exactos, una entidad puede relacionarse con cero o más entidades. Incluso se puede relacionar consigo misma.

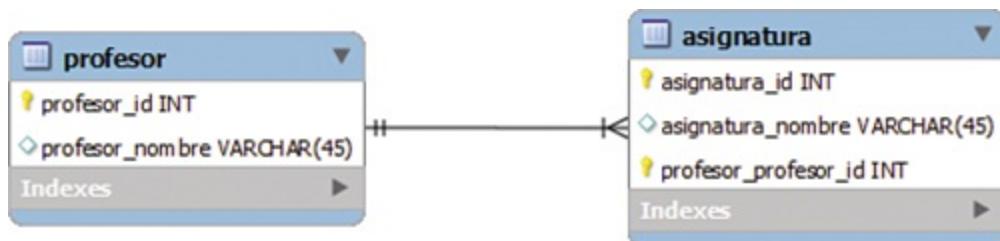
Gran parte de la complejidad a la hora de dibujar un diagrama de entidad/ relación recae en determinar las relaciones entre sus entidades.



Recuerde

Al modelar un diagrama entidad/relación, lo primero que debe identificarse son las entidades del mismo. Para ello debe analizarse qué información es necesaria en la aplicación y qué se necesita para guardarla en la base de datos.

Para representar las relaciones, se utilizarán líneas que unirán las distintas entidades entre sí. Hay diferentes notaciones para establecer cada tipo de relación. En este libro se explicará la notación mediante **patas de gallo**.

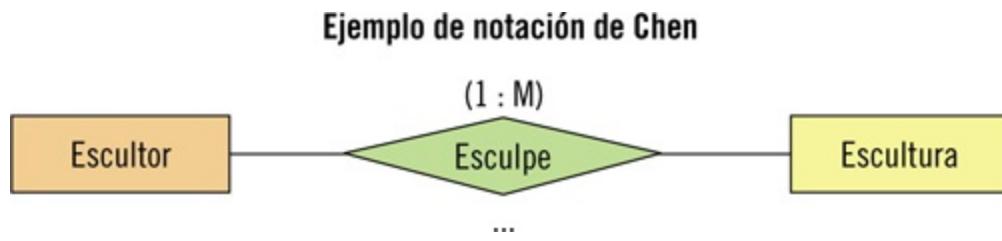


Ejemplo de notación mediante patas de gallo

En esta notación cada entidad se encuentra en un cuadrado cuyo título es el nombre de la misma. Debajo del nombre se listan los atributos. Las entidades se relacionan utilizando líneas que las conectan. Dependiendo de la notación en cada extremo de la relación, se indicará un tipo de relación u otra.

Hay otra notación bastante extendida llamada **notación de Chen** en la cual se utilizan rombos para escribir dentro un verbo que explique la relación y números en cada extremo para representar la cardinalidad. Sin embargo, el sistema de patas de gallo es más sencillo de dibujar, sobre todo cuando el número de entidades o relaciones crece. Para hacer un esbozo rápido de un par de entidades, la notación de Chen es más fácil de dibujar a mano, pero cuando hay muchas entidades y muchas relaciones, es más ordenada la notación de patas de gallo, porque se ahorra los rombos.

Una ventaja que aporta es que puede explicarse textualmente en qué consiste la relación, algo que las patas de gallo no proporcionan por sí solas.



Tipos de relaciones

Lo importante de una relación es **cómo contribuye** cada entidad implicada en la relación. Hay varios tipos de relaciones y para identificar qué tipo corresponde a cada relación, hay que pensar en los datos que albergarán esas entidades y pensar cuántos habrá de cada entidad en la relación.

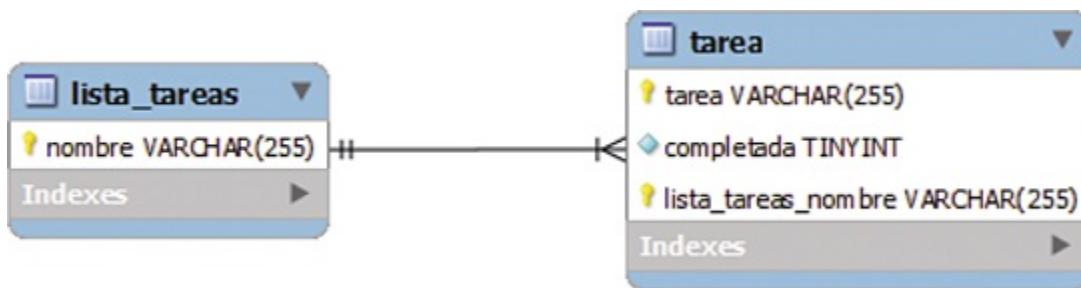
Una entidad puede relacionarse consigo misma, no es necesario que sean dos o más entidades diferentes. Por ejemplo, una tabla de empleados puede tener una relación consigo misma para determinar qué empleado supervisa a ese empleado. No hay ningún tipo de restricción en este tipo de relaciones, solamente presentará peculiaridades a la hora de crear las tablas de la base de datos.

Los tipos de relación son los siguientes:

- **Relación de uno a muchos (1: N):** un registro de la entidad 1 se relaciona con múltiples registros de la entidad 2. Ejemplo: una entidad “equipo” que se

relaciona con la entidad “jugador”. De un equipo formarán parte muchos jugadores.

Esto se representa gráficamente mediante una línea acabada en una pata de gallo. El extremo con la pata de gallo estará unido a la entidad que contendrá múltiples registros. El extremo recto irá a la tabla que contendrá solo uno.



Ejemplo de relación de uno a muchos representada gráficamente

Dentro de este apartado entrarían también las relaciones **cero a muchos**, en las que existe una relación sin que sea obligatorio que haya al menos un registro. Por ejemplo, dos entidades de “noticia” y “comentario” pueden relacionarse de cero a muchos, puesto que una noticia puede no tener comentarios.

- **Relación uno a uno (1:1)**: en esta, un registro de la entidad 1 se relaciona solo con un registro de la entidad 2. Dentro de este apartado entrarían las relaciones de **cero a uno**, en las que un registro puede relacionarse con otro de la entidad 2 o no. Se representan mediante una línea con ambos extremos rectos, como en la relación de uno a muchos.

Se utilizan para **separar dos conceptos estrechamente ligados** en dos entidades diferentes. Un ejemplo puede ser una entidad de “usuario” y otra de “perfil de Twitter”. En este caso la aplicación no permite que un usuario pueda tener más de uno.

También son útiles las relaciones cero a uno si quiere gestionarse, por ejemplo, el director de un departamento y la restricción es que solo un empleado puede ser el director y que un empleado solo pueda ser director de un departamento.

- **Relación de muchos a muchos (N: M)**: en esta relación múltiples registros de la entidad 1 se relacionan con múltiples registros de la entidad 2. Un ejemplo de relación de muchos a muchos pueden ser las entidades “libro” y “autor”. Un libro puede tener varios autores y un autor puede haber escrito muchos libros. Se representan mediante las patas de gallo en cada extremo.

Al igual que los dos tipos anteriores, es posible que uno o los dos extremos permitan no relacionarse con la otra entidad. Sería el caso de un autor que aún no haya escrito ningún libro, por ejemplo.



Sabía que...

La relación de uno a muchos es la más común y la que se encontrará más a menudo en el futuro.

La diferencia entre las relaciones 1: N y las relaciones N: M es que en las relaciones 1: N un registro de la entidad 1 se relaciona con muchos registros de la entidad 2, pero esto no sucede en el sentido contrario. Cuando varios registros de varias entidades se relacionen habrá que usar una relación de tipo N: M.

Ejemplos:

- Una relación entre productos y marcas: un producto pertenece a una sola marca y una marca tiene muchos productos. Relación 1: N.
- Una relación entre productos y categorías: un producto pertenece a varias categorías y en cada categoría hay muchos productos. Relación N: M.
- Una relación entre productos e imágenes: un producto puede tener varias imágenes pero una imagen solo pertenece a un producto. Relación 1: N.
- Relación entre productos y pedidos: un producto puede aparecer en muchos pedidos y en un pedido pueden aparecer muchos productos. Relación N: M.



Actividades

4. Identifique el tipo de relación que hay en los siguientes enunciados:

- Aplicación que gestione usuarios y permisos. Se necesita controlar que los usuarios tengan acceso a diferentes departamentos.
- En una aplicación de comercio electrónico hay que controlar los clientes y la lista de tarjetas de crédito que han añadido al sistema.
- En una web de búsqueda de empleo: se desea gestionar, por un lado, a personas y, por otro lado, el currículum vitae de cada persona.

Cardinalidad

Ya se ha avanzado bastante en cuanto a la cardinalidad explicando los tipos, pero aquí se profundizará sobre ello. La cardinalidad es el **número de registros** de cada extremo de la relación **que se pueden relacionar** con los de la otra entidad. Cada extremo de la relación tiene una cardinalidad.

Según el número de registros, puede ser: cardinalidad (0,1), cardinalidad (0, N), cardinalidad (1, 1), cardinalidad (1, N) y cardinalidad (N, M).

Cardinalidad (0, 1)

Un registro de una tabla se relaciona con otro de forma opcional. Se representa mediante un círculo seguido de una línea vertical en el extremo de la relación que contenga el elemento opcional. La notación representa el 0 y el 1 de forma sencilla. En el siguiente ejemplo, un perfil de una red social puede tener una foto principal o no (si el usuario aún no la ha puesto). Solo puede tener una foto principal.



Ejemplo de cardinalidad de cero a uno

Tanto este tipo de entidades como las (1, 1) suelen combinarse en una sola entidad que englobe ambas, por lo que es normal que los ejemplos sea vean un poco forzados.

Cardinalidad (0, N)

Un registro de una tabla se relaciona con muchos de la otra o con ninguna. La notación es un círculo seguido de una pata de gallo. El círculo representa un cero y las patas representan que puede haber muchos registros en esa entidad. En el ejemplo siguiente, un producto de una tienda online puede que tenga muchas imágenes, una o ninguna.



Ejemplo de relación de cero a muchos

Cardinalidad (1, 1)

Un registro de la entidad 1 se relaciona con un registro de la entidad 2. Este tipo de entidades se suele combinar en una sola entidad que tenga los atributos de ambas, pero a veces puede ser conveniente separarlas para evitar complejidades en el modelo de datos. Se dibuja mediante dos líneas verticales, imitando dos números uno.

El ejemplo siguiente podría tratarse de una entidad que guarde datos de un cliente (nombre, dirección, teléfono) y otra que guarde los datos específicos de su tarjeta sanitaria (número, fecha de caducidad, etc.). Un cliente solo tendrá una tarjeta y será obligatoria en este ejemplo.



Ejemplo de cardinalidad de uno a uno

Cardinalidad (1, N)

Un registro de la entidad 1 se relaciona con varios registros de la entidad 2. Siempre se relacionará con, al menos, uno. Se dibuja mediante una línea vertical seguida de las patas de gallo. Esto denota el uno y el hecho de que pueden ser muchos los registros involucrados.



Ejemplo de cardinalidad de uno a muchos



Ejemplo

Un ejemplo de cardinalidad 1, N sería un país, que puede tener muchas ciudades, pero al menos debe tener una.

Cardinalidad (N, M)

Este es un tipo especial de cardinalidad, puesto que a la hora de modelar el diseño de tablas y crear una base de datos, se creará una **tabla de relación** que

guardará las claves primarias de cada entidad para poder establecer la relación entre ambas.

En cada extremo se pondrá la cardinalidad y cada una será 0: N o 1: N, según sea aplicable.

Puede usarse el mismo ejemplo utilizado para definir este tipo de relaciones: el de libros escritos por múltiples autores, los cuales pueden haber escrito varios libros.



Ejemplo de cardinalidad de muchos a muchos



Actividades

5. Identifique la cardinalidad en los siguientes enunciados:

- Relación entre marcas de coche y modelos.
- Relación entre noticias y categorías.
- Relación entre usuarios y sus perfiles de Facebook (un usuario puede no tener perfil en Facebook).
- Relación entre productos y pedidos.
- Relación entre usuarios e intentos fallidos de acceder al sistema.

6. Realice un diagrama entidad/relación que sirva para representar el siguiente problema:

- Se quieren gestionar los datos de una escuela, la cual está compuesta por profesores y alumnos. De ellos quiere saberse su nombre y apellidos. De los profesores, su titulación.
- Los alumnos van a varias clases. Cada clase tiene un nombre y la imparte un único profesor.

4.5. Claves. Tipos de claves

Cada registro debe tener una clave primaria de forma obligatoria para cumplir con el modelo relacional. Y ya se ha visto que una clave primaria puede estar compuesta por uno o varios atributos, dependiendo de las necesidades de identificación que se tengan.

Una clave primaria es un valor único que no se repite y que identifica a un registro en particular. Cada atributo o conjunto de atributos que pueda servir como identificador único para los registros se llama **clave candidata**. Para que una clave candidata pueda ser utilizada como clave primaria se deben cumplir las siguientes condiciones:

- Debe identificar de forma única al registro. No debe ser posible que haya otro registro con el mismo valor.

Ejemplos de clave primaria:

- El DNI de una persona.
- Asignar un ID auto numérico correlativo a cada registro.
- El *e-mail* de un usuario registrado.
- Código de barras de un producto.

- No se admiten valores nulos en ninguno de los atributos que la componen.

Una vez conocidas las restricciones, pueden identificarse entre los siguientes tipos de claves:

- **Superclave:** un conjunto de atributos que identifica únicamente a una entidad. Si se añaden más atributos a esta superclave, esta seguirá siendo una superclave. Ejemplo: utilizar nombre, apellido1, apellido2 y fecha_nacimiento como clave primaria. Si se añade sexo a la clave, esta sigue siendo una superclave. Se dice conjunto de atributos pero puede ser solamente un atributo. Por ejemplo, pueden tenerse DNI, nombre, apellidos y utilizar DNI únicamente como clave primaria.
- **Clave candidata:** se parte de una superclave y se van eliminando atributos hasta que quede una clave con el menor número de atributos posibles y siga identificando únicamente al registro. Vea las siguientes claves posibles:

- (Nombre, Apellido1, Apellido2, Fecha_nacimiento, Sexo, Ciudad).
- (Nombre, Apellido1, Apellido2, Fecha_nacimiento, Sexo).
- (Nombre, Apellido1, Apellido2, Fecha_nacimiento). Esta sería la clave candidata, puesto que identifica de forma única a la entidad y tiene el menor número de campos posibles.

- **Clave primaria:** es la clave candidata elegida por el analista informático para

ser utilizada de entre la lista de claves candidatas. Normalmente, se elegirá tratando de escoger la que menos campos contenga, su tipo de dato sea más fácil de manejar por el Sistema Gestor de Bases de Datos o sea más fácil de usar por la aplicación que implemente la base de datos. Siguiendo el ejemplo anterior, puede encontrarse con las claves candidatas de (DNI) y de (Nombre, Apellido1, Apellido2, Fecha_nacimiento). El analista elegiría típicamente el DNI, puesto que contiene menos campos y es más simple, pero tiene libertad de usar cualquiera de las dos, puesto que ambas son válidas.

- **Atributo único:** es un valor que no puede repetirse en un atributo de una entidad. Típicamente puede ser considerada como clave candidata a menos que admita valores nulos. Por ejemplo, puede tener una tabla de usuarios en la que se guarde el ID de perfil de Facebook y tenga que ser único, pero que no sea obligatorio que todos los usuarios lo hayan llenado con un valor.

En un diagrama entidad/relación, el o los atributos que sirvan como clave primaria aparecerán subrayados.



Actividades

7. Determine las claves candidatas y elija la clave primaria óptima en los siguientes casos:

- Tabla de productos. Atributos: código producto, nombre, precio, fecha creación.
 - Tabla de usuarios. Atributos: e-mail, nombre de usuario (único), nombre completo, fecha de nacimiento, fecha de registro.
 - Tabla de clientes. Atributos: nombre, apellidos, fecha de nacimiento, DNI.
-



Aplicación práctica

Se acaba de reunir con el comercial de la empresa, el cual ha estado hablando con un nuevo cliente que quiere que se desarrolle una aplicación informática que le permita gestionar las facturas de su negocio.

Lo que el comercial comenta es lo siguiente:

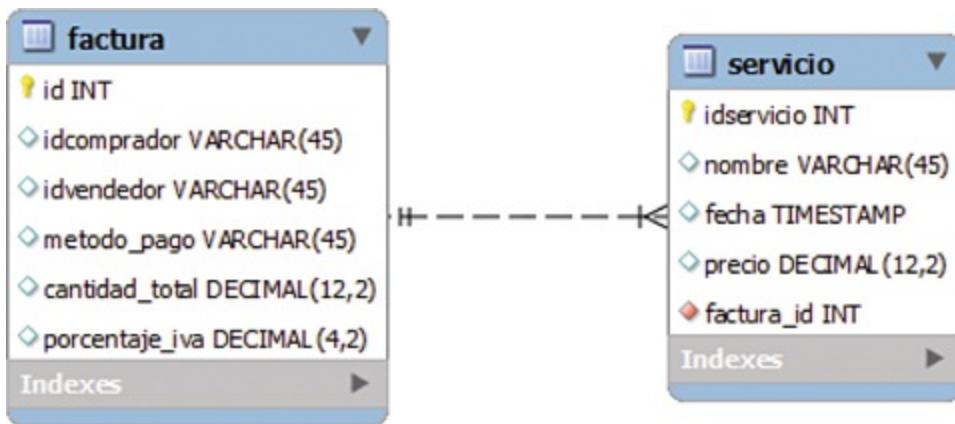
Cada factura tiene un número que la identifica. El número empieza en 1 y va aumentando de uno en uno para cada factura.

Se debe guardar la fecha, número de identificación del vendedor y del comprador (DNI o CIF dependiendo de si es un particular o una empresa). Además, se debe guardar el método de pago, cantidad total y porcentaje de IVA aplicado.

Cada factura está compuesta por una serie de servicios prestados. Cada servicio tiene un nombre, una fecha y un precio. Aquí el precio se almacenará sin IVA, porque esto se calculará a la hora de generar la factura completa.

Se necesita realizar un diagrama de entidad/relación de tal manera que sea posible implementar una base de datos que permita guardar estos datos siguiendo el modelo relacional.

SOLUCIÓN



Ejemplo de cardinalidad de muchos a muchos

4.6. Normalización. Formas normales

El uso de la normalización en bases de datos se utiliza para reducir **redundancia** y para establecer la dependencia de los atributos correctamente. El objetivo del modelo relacional es ser lo más descriptivo posible y evitar que la aplicación que lo implemente sea la que se encargue de administrar las restricciones y la integridad de los datos.

Para empezar a ilustrar qué es la normalización, se va a mostrar un ejemplo de una tabla no normalizada con una estructura realmente mala:

DNI	Nombre	Dirección	Departamento	Planta del departamento
26124A	José María Pérez	C/ Badajoz, 2	Recepción	2
26124A	José María Pérez	C/ Almería	Secretaría	3
82993W	María Vélez	C/ Cervantes, S/N	Secretaría	3

Suponiendo que se están guardando empleados y a qué departamento pertenecen y que un mismo empleado puede pertenecer a más de un departamento aparecen las siguientes anomalías:

- Hay registros duplicados. Un mismo empleado está repetido para poder reflejar que está en más de un departamento.
- La planta del departamento está repetida tantas veces como aparezcan empleados en ese departamento. La planta del departamento **depende** del departamento, no del empleado.
- Como se ve, la dirección de “José María Pérez” no es consistente. Esto se debe a que la que se encarga de que los datos sean consistentes es la aplicación que utiliza la base de datos. En este caso se ha actualizado un registro y se ha quedado el otro con el valor anterior.
- Si se modifica un registro hay que asegurarse de modificar todos los registros que se puedan ver afectados.
- Si se borra un registro tiene que saber si lo que se borra es el empleado o si lo que pasa es que ese empleado ya no pertenece a ese departamento.

Por lo tanto, los objetivos de la normalización de tablas son los siguientes:

- **Eliminar la posibilidad de anomalías al modificar los datos** (se busca que modificar un registro no afecte necesariamente a otros registros de una tabla). Tiene que asegurarse que al modificar un registro no pueda haber implicaciones en otros registros. Pueden surgir los siguientes problemas:
 - Anomalías al insertar datos: en el ejemplo anterior, si un empleado acaba de llegar y aún no ha sido asignado a un departamento, tendría que mantener esos campos como nulos; tantos campos nulos como campos dependan del departamento. En este caso son dos, pero podría haber 20 campos con

información sobre el mismo.

- Anomalías al modificar datos: puede que modificar un registro implique cambios también en otros registros, por lo que si no se modifican todos, hay inconsistencia en los datos. Habrá casos en que haya que modificar todos los registros asociados o no, dependiendo de la situación.
- Anomalías al borrar: depende de si se quiere borrar una fila o sus datos asociados, habrá que borrar todas las filas o no. En el ejemplo anterior, dependiendo de si el objetivo es borrar a un empleado o si es desasignar un departamento habrá que borrar un número de filas u otro.

- **Minimizar el impacto en cambios en la estructura de la base de datos** de tal manera que las aplicaciones que la usen apenas se vean afectadas.
- **Hacer que el modelo de datos aporte más información** al usuario. La base de datos explica más información al que la usa. Las relaciones y las restricciones están claras. En la tabla del ejemplo se sabe que un empleado puede pertenecer a más de un departamento porque se ve un duplicado, pero de lo contrario sería imposible saberlo.
- **Facilitar las consultas a la base de datos.** Con diseños no normalizados es posible que se pierdan muchas posibilidades de consultar la información. Por lo general, una tabla normalizada permitirá utilizar todas las características del lenguaje SQL sin problemas.

Cuando una persona experimentada realiza un modelo entidad/relación de no mucha complejidad, por lo general, se encontrará ya normalizado, porque habrá tenido en cuenta los posibles problemas conforme los iba encontrando.

A grandes rasgos, la normalización consiste en asegurarse de que no hay duplicidad en los datos y de que cada registro tenga su significado completo; que cada entidad esté bien definida.



Sabía que...

A veces, en aplicaciones con grandes exigencias de rendimiento, puede ser necesario *denormalizar* una tabla para poder realizar accesos más rápidos a la misma. Normalmente, se busca un compromiso entre rendimiento y normalización.

En este libro no se tratarán estas técnicas, puesto que lo ideal es seguir estrictamente el modelo relacional y utilizar tablas normalizadas. Solo en contadas ocasiones es necesario denormalizar una tabla.

A continuación, se ahondará en los aspectos clave para normalizar una base de datos.



Actividades

8. ¿Por qué es negativa la duplicidad de datos?

9. ¿Cómo puede contribuir a un buen rendimiento tener una tabla normalizada?

Concepto de dependencia

En normalización todo se basa en la dependencia de los atributos de las entidades. **Todos los atributos deben depender únicamente de la clave primaria**, de lo contrario la tabla no está normalizada. La dependencia se representa mediante una flecha: $A \rightarrow B$ representa que el atributo B depende del atributo A.

A continuación, se explicará cada tipo de dependencia y se utilizará esa notación para ello.

Dependencia funcional

En una entidad, el atributo B tiene una dependencia funcional del conjunto de atributos A (puede ser un atributo o varios) (se escribe $A \rightarrow B$) si y solo si para cada valor de A solo hay un posible valor de B.

Este es el nivel más básico de dependencia. Un ejemplo podría ser una tabla de usuarios en la que $\{\text{usuario_id}\} \rightarrow \{\text{usuario_fecha_creacion}\}$. Para un `usuario_id` hay una y solo una fecha de creación.

Dependencia funcional completa

Un atributo B es dependiente funcional completo del conjunto de atributos A si y solo si:

- Es dependiente funcional de A.
- No es dependiente funcional de ningún subconjunto de A. `{usuario_email}` es dependiente funcional de `{usuario_id, usuario_nombre}` pero no es completamente funcional, porque también depende de `{usuario_id}` por

separado. Si se elimina el campo `{usuario_nombre}` seguiría habiendo dependencia $A \rightarrow B$.

Es sencillamente una simplificación. Es una dependencia en la que un atributo depende del mínimo número posible de atributos.

Dependencia transitiva

Suponiendo que $A \rightarrow B$ y $B \rightarrow C$, se deduce que $A \rightarrow C$ aunque esto no sea evidente. Según la siguiente tabla:

Producto	Precio	Proveedor	Ciudad Proveedor
Pack de bolígrafos	5,99 €	José Manuel	Málaga
Paquete de folios	2,99 €	Manuela	Ciudad Real

Se deduce:

- $\{\text{Producto}\} \rightarrow \{\text{Proveedor}\}$
- $\{\text{Proveedor}\} \rightarrow \{\text{Ciudad Proveedor}\}$
- $\{\text{Ciudad Proveedor}\}$ NO depende de $\{\text{Producto}\}$

Sin embargo, existe una dependencia transitiva, puesto que conociendo el producto, puede obtenerse la ciudad del proveedor. Esto es una dependencia transitiva. Normalmente esto se quiere evitar.

Dependencia multivaluada

A multidetermina a B si por cada valor de A existen varios valores posibles de B. No importan las demás relaciones de los demás atributos.

Las dependencias multivaluadas son en la mayoría de casos las principales responsables de datos duplicados, puesto que habrá múltiples registros para especificar cada valor de B. Se puede ver en el siguiente ejemplo:

Noticia_título	Noticia_fecha	Autor_nombre

Caso de corrupción	29/12/2013	Alberto Higueros
Caso de corrupción	29/12/2013	Rafael Humberto
Boda real	15/01/2014	Maria Vargas

Se aprecia que {Noticia_titulo} tiene dependencia multivaluada con {Autor_nombre} porque por cada valor de {Noticia_titulo} hay varios posibles valores de {Autor_nombre}.



Actividades

10. ¿Cuál es la prioridad máxima a la hora de normalizar tablas?
 11. Cuenta con una entidad de nombre “pedido”. En esa tabla aparecen un ID de pedido, fecha y hora del mismo, nombre del cliente, dirección del cliente, marca y modelo del producto junto a su número de identificación.
Determine las dependencias entre cada atributo.
-

Claves

Además de las dependencias, de cara a la normalización existen otras características de los atributos que conviene destacar:

- **Superclave:** un conjunto de atributos que identifica de forma única a un registro. Puede haber varias superclaves.
- **Clave candidata:** es el conjunto de atributos de menor tamaño que identifica de forma única a un registro. Es posible que haya varias, puesto que varias claves pueden tener el mismo número de atributos.
- **Atributo no primario:** un atributo que no se incluye en ninguna clave candidata.
- **Atributo primario:** un atributo que aparece dentro de alguna de las claves candidatas.
- **Clave primaria:** solo puede haber una. No es más que la clave candidata elegida por el diseñador de la base de datos.

Formas normales

Las formas normales son convenciones para asegurar que los niveles de redundancia

e inconsistencia están bajo control en una base de datos. Cuanto mayor sea su nivel de normalización, menos posible será que una tabla presente problemas de ese tipo.

Las formas normales se aplican a tablas en particular. Si se dice que una base de datos está normalizada se hace referencia a que todas sus tablas están normalizadas. Aunque haya más, por ahora se verán las siguientes formas normales:

- 1FN: Primera forma normal
- 2FN: Segunda forma normal
- 3FN: Tercera forma normal
- FNBC: Forma normal de Boyce-Codd
- 4FN: Cuarta forma normal
- 5FN: Quinta forma normal



Sabía que...

Las tres primeras formas normales fueron creadas por Edgar F. Codd tras ver el uso que daban los analistas informáticos del modelo relacional. Las demás fueron añadidas años después por otras personas tras afrontar problemas aún más complejos aunque menos comunes.

Cada forma normal asume que su nivel inferior se cumple. Es decir, una tabla en 3FN quiere decir que está en 3FN, 2FN y 1FN.

Las tres primeras formas normales fueron las originales, creadas por Edgar F. Codd y son las que garantizan que las tablas tengan un nivel aceptable de normalización.

Normalmente, un diseñador de bases de datos experimentado realizará modelos de datos que automáticamente se encuentren en 3FN, puesto que conocerá los problemas más comunes y los subsanará conforme los vaya reconociendo en el enunciado del problema que quiera resolver. No obstante, cuando el modelo entidad relación tenga un tamaño considerable, siempre habrá que realizar comprobaciones, porque es muy probable que se haya colado algún caso que no esté tratado correctamente.

Los siguientes niveles de normalización se fueron añadiendo mediante la experiencia en el uso de bases de datos de todo tipo y son más específicos; resuelven problemas concretos.

A continuación, se explicará cada forma normal y después se estudiarán aplicaciones prácticas de cada una y la manera de subsanarlas.

A lo largo de las siguientes secciones por cada forma normal se utilizarán ejemplos sencillos, por lo que se pueden deducir las dependencias entre los campos fácilmente. No suelen tener dependencias no explícitas pero si las tuvieran, estarían explicadas.

1FN: Primera forma normal

Una tabla se considera en forma normal si y solo si cada atributo de la misma contiene un único valor.

Esta es una propiedad básica del modelo relacional y todas las tablas tendrán atributos de valor único.

Ejemplo:

Nombre	Email
José	jose@ejemplo.com jose@gmail.com
Manu	manu@hotmail.com

No es posible tener más de un valor en el mismo campo.

2FN: Segunda forma normal

Una tabla se encuentra en 2FN si y solo si se encuentra en 1FN y todos los atributos no primarios dependen de forma completa de una clave candidata. Los atributos no primarios eran los que no pertenecían a ninguna clave candidata.

Si una tabla no tiene claves candidatas conformadas de más de un atributo, se encuentra en 2FN automáticamente, puesto que no sería posible tener este problema.

Se hace referencia a “clave candidata” puesto que, en caso de haber más de una clave candidata, esta afirmación debe cumplirse para ***todas las claves candidatas***. Si un atributo solo depende de forma completa de algunas claves candidatas, es porque la estructura es incorrecta.

Normalmente esto no sucede, por lo que lo habitual es fijarse en la clave primaria simplemente.

A partir de la 2FN es necesario haber detectado todas las dependencias funcionales entre atributos, para ver posibles anomalías.

Ejemplo:

Autor	Libro	Autor ciudad
Manuel	Informática para amas de casa	Sevilla
Manuel	Programación web para novatos	Sevilla
José	Montaje de ordenadores	Barcelona

{Autor} no puede ser clave candidata porque no identifica de forma única a cada registro. Tendría que utilizarse {Autor, Libro} como clave candidata.

{Autor Ciudad} es un atributo no primario (no pertenece a la clave candidata) y solo depende de {Autor}.



Sabía que...

Normalmente, si una tabla no está en 2FN es porque no se han identificado las entidades del problema correctamente. No se ha diferenciado adecuadamente entre varias entidades que realmente son distintas.



Actividades

12. ¿Se encuentra la siguiente tabla en 2FN?

Marca	Modelo	Precio	Marca_país
Bosch	29AMF	50	Alemania
Bosch	123ACC	200	Alemania

Toshiba	M49BCC	100	Japón
---------	--------	-----	-------

3FN: Tercera forma normal

Una tabla se encuentra en 3FN si y solo si está en 2FN y ninguno de los atributos no primarios depende transitivamente de cada superclave de la tabla.

Se hace referencia a la superclave porque todo atributo debe depender de forma funcional completa de cualquier clave candidata o superclave de la tabla. Esto impide que haya dependencias transitivas entre ningún campo de una tabla.

Un atributo no primario es uno que no pertenece a la clave candidata. Una dependencia transitiva se produce al deducir que $A \rightarrow C$ a través de $A \rightarrow B$ y $B \rightarrow C$ cuando la relación no es obvia por sí sola.

Lo que viene a decir es que todos los atributos de la tabla no dependen unos de otros, sino que **todos** dependen de forma completa de la clave primaria (y de cualquier clave candidata).

El problema radica en las relaciones transitivas que no dependen de la clave primaria.



Ejemplo

Campeonato	Ganador	Nacionalidad
2004 Formula One World Drivers' Championship	Michael Schumacher	Alemana
2005 Formula One World Drivers' Championship	Fernando Alonso	Española
2006 Formula One World Drivers' Championship	Fernando Alonso	Española

La clave primaria es el campeonato. El ganador depende de forma funcional del campeonato. La nacionalidad depende de ganador, por lo que hay una dependencia transitiva entre la nacionalidad y el campeonato.

Este tipo de cosas se deben evitar. En el siguiente apartado se estudiará cómo se podría arreglar esto.



Actividades

13. ¿Se encuentra la siguiente tabla en 3FN?

Certamen	Ganador	Certamen_lugar_celebración
Miss España 2004	María Jesús Ruiz Garzón	Castellón
Miss España 2009	Estíbaliz Pereira Rabade	Cancún
Miss España 2011	Andrea Huisgen	Sevilla

FNBC: Forma normal de Boyce-Codd

Es una versión mejorada de la 3FN. Fue diseñada por Raymond F. Boyce junto a Edgar F. Codd para solventar algunos problemas no cubiertos por la 3FN.

Una tabla se encuentra en FNBC si y solo si se encuentra en 3FN y si cada dependencia funcional no trivial tiene una clave candidata como determinante.



Sabía que...

Esta forma normal es muy especial y solo se usa para tratar unos casos específicos que no cubre la 3FN. Muy rara vez aparecerán problemas de este tipo. Casi todos los ejemplos están forzados.

Una dependencia funcional no trivial es aquella que no se deduce automáticamente (por ejemplo, que A depende de A). Un determinante es un atributo del cual depende funcionalmente de manera completa otro atributo de la relación.

Si una tabla no tiene más de una clave candidata compuesta con atributos en común, puede deducirse automáticamente que está en FNBC, puesto que no es posible que se dé la condición.

Si existen varias claves candidatas compuestas y hay un elemento común entre ellos, la tabla no se encuentra en FNBC. Una tabla cuyas claves candidatas sean (A, B) y (B, C) no está en FNBC.

Muy raramente se producirá esto, pero hay que saber cómo identificarlo y solucionarlo. La dificultad recae en identificar las claves candidatas.



Ejemplo

Tabla para saber qué profesor imparte qué asignatura a cada alumno y que se encuentra en 3FN.

Profesor	Asignatura	Alumno
Paco	Lengua	José
Luisa	Filosofía	Marta
Rafael	Matemáticas	Luis

La dificultad recae en saber cuáles pueden ser las claves candidatas. En este caso:

- (Profesor, Asignatura).
- (Asignatura, Alumno).

No se encuentra en FNBC, puesto que no todas las dependencias no triviales dependen de forma completa de la clave candidata.

4FN: Cuarta forma normal

Una tabla está en 4FN si y solo si se encuentra en FNBC y no existen dependencias multivaluadas.



Sabía que...

Normalmente, una tabla no se encuentra en 4FN porque las relaciones entre varias entidades no han sido identificadas de forma correcta.

Las dependencias multivaluadas se daban cuando para un valor de A podía haber uno o más valores de B.



Ejemplo

Producto	Color	Talla
Calcetines tobilleros	Negro	38-40
Calcetines tobilleros	Negro	40-42
Calcetas	Blanco	40-42

Por cada valor de {Producto, Color} hay varios valores de {Talla}.

Esto es incorrecto puesto que hay duplicidad.

5FN: Quinta forma normal

Una tabla se encuentra en 5FN si y solo si se encuentra en 4FN y toda dependencia de reunión es consecuencia de las claves candidatas.



Sabía que...

Que una tabla se encuentre o no en 5FN depende de los requisitos del sistema. Una tabla por sí sola no aporta suficiente información como para saber si lo está o no.

Una dependencia de reunión se produce cuando solo unos registros de la tabla A se pueden relacionar con otros registros de la tabla B. Es decir, que no cualquier registro de A se puede relacionar con cualquier registro de B, sino que hay restricciones de por medio que no lo permiten. Por ejemplo, que no todos los empleados puedan estar asociados a coches de empresa, que sólo los directores puedan.

Esto se aplica a restricciones impuestas por los problemas del mundo real que

quiera resolver la base de datos. Es decir, son restricciones en los datos que son conocidas, pero no quedan reflejadas en el modelo de datos.

Dependen del enunciado del problema planteado, por lo que en unos casos la misma tabla estará normalizada y en otros no.



Actividades

14. ¿Es posible determinar si una tabla se encuentra en 5FN simplemente fijándose en el diagrama entidad/relación?
-



Ejemplo

Médico	Tratamiento	Paciente	Fecha
Dr. Torreiglesias	Anestesia	Manuel	20/09/2013
Dr. Torreiglesias	Vacuna	Marcos	21/09/2013
Dra. Gómez	Radiografía	Alberto	18/09/2013
Dra. Díaz	Anestesia	María	19/09/2013

Quiere saberse qué tratamiento le ha dado cada médico a cada paciente y cuándo.

La tabla está en 4FN y todo está normalizado, pero, ¿qué ocurriría si cada doctor solo pudiera dar una serie de tratamientos y que eso estuviera reflejado en la base de datos? Tal y como está planteada la tabla ahora mismo, sería perfectamente posible que la Dra. Gómez realizara una anestesia cuando solamente puede hacer radiografías y ecografías.

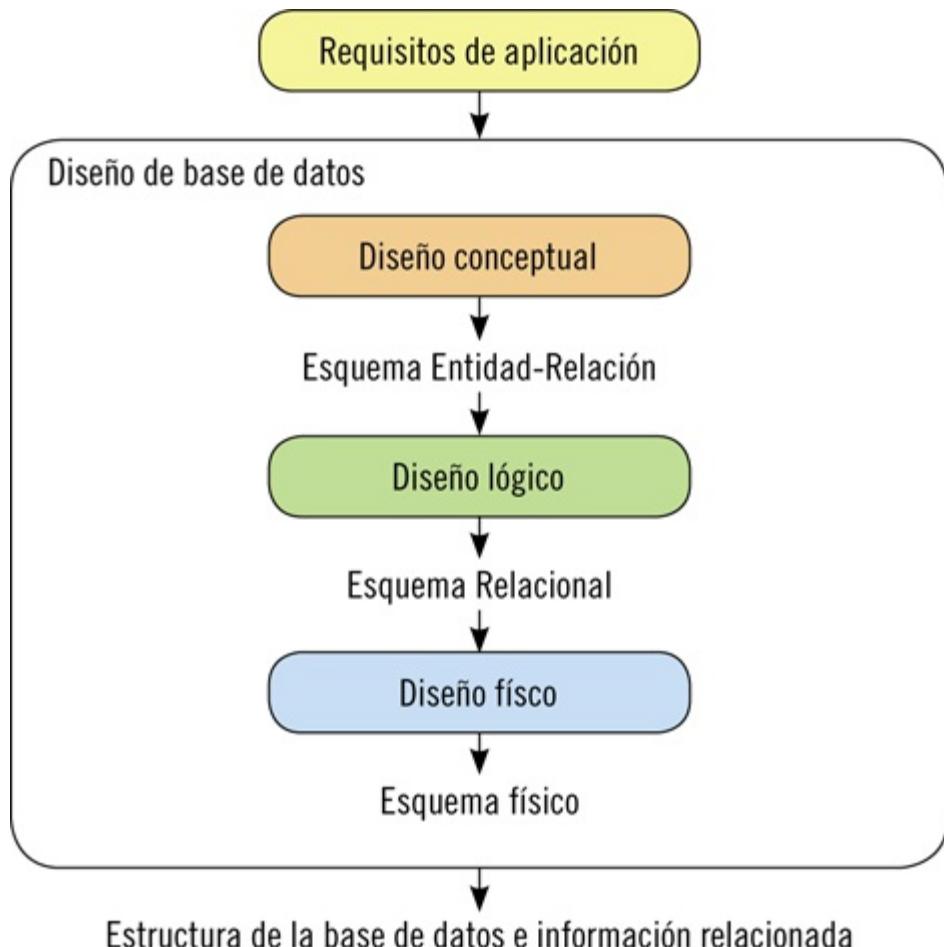
Este tipo de problema solo se da en casos muy concretos en los que el problema real no queda reflejado en el modelo lógico de datos y su gestión quedaría en manos de la aplicación.

5. Construcción del modelo lógico de datos

Una vez se ha dibujado el diagrama entidad relación, se puede decir que termina la fase de **diseño conceptual**. Este diseño simplemente sirve para plasmar los requisitos de una aplicación de forma visual.

Ahora, el objetivo será pasar a la fase de **diseño lógico**: convertir esa entidad/relación en una serie de tablas y relaciones que **cumplan con el modelo relacional**.

Solamente habrá que seguir unos sencillos pasos para conseguirlo. Por lo general, cada entidad se corresponderá con una tabla y cada atributo, con una columna. Las relaciones entre entidades se convertirán o bien en columnas o en nuevas tablas.



Al diseñar una base de datos se seguirá este proceso. Primero se crea un diseño conceptual, el cual se convertirá en un diseño lógico. Este diseño lógico permitirá implementar la base de datos como se deseé.

El objetivo de este paso es crear un **esquema relacional**, que no es más que una representación textual explicando cada tabla con su nombre y sus columnas. Este esquema relacional se le podrá entregar a cualquier administrador de bases de datos o programador informático para que lo implemente en su SGBD o sistema de archivos.



Actividades

15. Se desea realizar una aplicación que sirva para gestionar el inventario de una tienda. Hace falta tener una tabla con todos los productos. De cada producto consta su código de barras, marca, modelo, precio, fecha de compra. Dibuje un diagrama entidad/relación que refleje esto.

5.1. Especificación de tablas

El paso fundamental en un esquema relacional es listar las tablas que una base de datos tendrá. Aquí la representación gráfica no es tan importante como en los diagramas entidad/relación. No obstante, lo normal es utilizar una sintaxis de la siguiente forma:

Nombre_de_tabla (clave_primaria, campo1, campo2)

El nombre de la tabla no debería contener espacios o caracteres especiales. Tras el nombre de la misma, entre paréntesis se listaría cada columna separada por comas. La clave primaria va subrayada.

Cada tabla con sus columnas iría en una línea de texto distinta. Al convertir un diagrama entidad/relación en esquema relacional, por lo general, **cada entidad se convertirá en una tabla** del mismo nombre.

En relaciones tipo **uno a uno** son posibles dos opciones: crear dos tablas y relacionarlas entre sí de manera estándar o crear una **superentidad** que englobe los campos de ambas y crear una sola tabla. Ambas soluciones son correctas y es cuestión del analista informático decidir qué es lo mejor.

Lo normal es unir ambas entidades en una tabla para evitar complejidad. Si las dos entidades son muy grandes por sí solas, sí puede convenir mantenerlas separadas. Un

ejemplo puede ser el de dos entidades de usuarios y currículums, donde ambas tablas tendrían muchos atributos y podría ser interesante dejarlas separadas.

En relaciones de **muchos a muchos**, se necesitará crear una tabla extra de relación entre las entidades involucradas. Siguiendo el ejercicio antes propuesto de “Determinar la cardinalidad en la relación entre noticias y categorías”, en este caso la respuesta era que la cardinalidad era de muchos a muchos, puesto que una noticia puede tener muchas categorías y una categoría puede tener muchas noticias.

Se crearían las tablas noticia (noticia_id clave primaria) y categoría (categoría_id como clave primaria).

Tendría que crearse la tabla noticia_categoria (noticia_id, categoría_id clave primaria). De esta manera, al utilizar la base de datos, para saber a qué categorías pertenece una noticia, basta mirar la tabla noticia_categoria.

5.2. Definición de columnas

Por lo general, cada atributo de cada entidad se convertirá en una columna de la tabla. Debe especificarse si la columna admitirá valores nulos o no. También debe especificarse qué columnas formarán la clave primaria subrayando su nombre. No se permiten nombres de columna repetidos.



Importante

Es conveniente utilizar nombres sin símbolos extraños o espacios para así garantizar la compatibilidad con los distintos Sistemas Gestores de Bases de Datos.



Nota

Lo ideal es normalizar la notación de columnas: que todas sigan la misma estructura.

Por ejemplo, utilizar como prefijo de cada columna el nombre de la tabla, utilizar todos los nombres en singular, que la clave primaria aparezca al principio, etc.

No hay estándares fijados para esto, pero es conveniente utilizar el mismo criterio siempre.

Para crear las relaciones entre tablas, habrá que crear columnas adicionales para poder identificarlas. Estas nuevas columnas recibirán el nombre de **claves foráneas** y contendrán el valor de la clave primaria de la tabla con la que se relacionan.

Esta es una regla de transformación al modelo relacional. Cuando la cardinalidad de una relación no sea N: M, habrá que crear columnas que contengan el valor de la clave primaria de la tabla referenciada. De esta manera pueden relacionarse los registros de una tabla con otra. Simplemente, habría que consultar los registros cuyo valor de la clave primaria de B sea igual que el valor de la clave foránea en A.

En una relación de uno a muchos entre productos e imágenes, tendrían que crearse las siguientes tablas:

- producto (producto_id, producto_nombre, producto_precio).
- imagen (imagen_id, imagen_url, producto_id [clave foránea de producto]).

De esta manera podrían relacionarse productos con imágenes. El valor de la clave primaria de la tabla producto pasaría a la tabla imagen. Si se quisiera obtener la lista de imágenes de un producto, tan solo habría que usar el valor de la clave primaria en la tabla de imágenes.

Se debe tener cuidado, porque hay que mirar a ambos lados de la cardinalidad. Por ejemplo, en una relación entre usuarios y comentarios donde se permitan comentarios anónimos no realizados por usuarios, aparecería una relación de 0: N en el lado de los comentarios y una de 0: 1 en el lado de los usuarios. En este caso, se llevaría la clave primaria de usuarios a la tabla de comentarios, pero debería permitir que admitiera valores nulos, puesto que no todos los comentarios estarán escritos por usuarios.

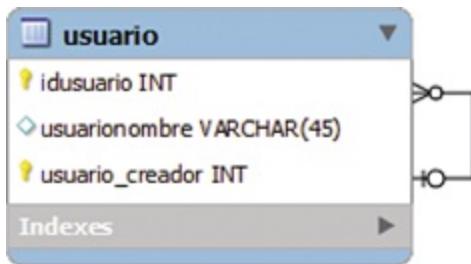


Relación entre usuarios y comentarios. La columna “usuario_idusuario” de “comentario” admitiría nulos, porque se permiten comentarios anónimos.

Recuérdese que además, las relaciones pueden ser reflexivas: una tabla puede relacionarse consigo misma. En este caso se procedería de la misma manera, añadiendo las columnas extra a la tabla. Por ejemplo:

Tabla categoría (categoria_id, categoria_nombre). Piense en un menú desplegable de navegación por categorías. Una categoría puede tener categorías hijas y categorías padre. Una categoría hija solo tiene un parente. Sería una relación de 0: N por parte de la categoría padre y de 0:1 por parte de la categoría hija.

En ese caso la tabla se quedaría como categoría (categoria_id, categoria_nombre, categoria_padre_id [permite nulos, clave foránea de categoría]). Se añade un campo con el valor de la clave primaria del otro registro que se corresponde a su categoría parente. Puesto que habrá categorías que no tengan parente, deben permitirse nulos. Dado que la clave primaria ya tiene un nombre y no pueden repetirse nombres de columna, habrá que utilizar otro que sea parecido e identifique un poco el tipo de relación que tiene.



En este caso un usuario puede haber creado otros usuarios. Un usuario puede haber sido creado por otro o no (en caso del administrador, por ejemplo). La columna usuario_creador admitiría nulos.

5.3. Especificación de claves

Ya se ha visto cómo crear las claves a partir del modelo entidad/relación. Debe especificarse la clave primaria de cada tabla, nombrar todas las claves foráneas que se encuentren y decir de dónde provienen.

No obstante, puede hacerse un pequeño resumen:

- Clave primaria: conjunto de columnas que identifican de forma única a un registro.
- Clave candidata: conjunto de columnas que podrían identificar de forma única a un registro. Puede haber más de una. No debe admitir valores nulos, de lo contrario no sería clave candidata.

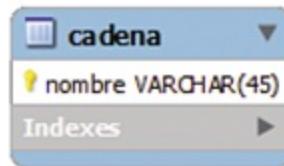
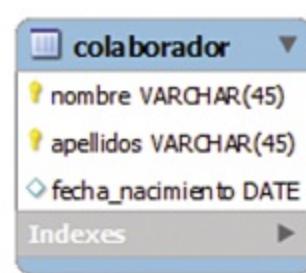
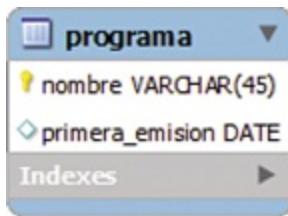
- Clave única: conjunto de columnas cuyo valor no puede repetirse en otros registros. Pueden admitir valores nulos, en cuyo caso sí se permitirían varios registros cuyo valor contuviera nulos.

Al crear un esquema relacional, las columnas que contengan la clave primaria deben ir subrayadas. Las demás claves, si no son obvias, pueden ser explicadas aparte, para dejar claro que esa columna no debe tener valores duplicados, etc.



Actividades

16. Cree un esquema relacional del siguiente entidad/relación:



Se ha dejado el esquema sin las relaciones para que tenga más dificultad. Un colaborador pertenece a una única cadena. Un programa pertenece a una única cadena.

Un colaborador puede colaborar en varios programas. En un programa colaboran varios colaboradores.

Cree el esquema relacional listando las tablas resultantes con sus columnas.

5.4. Conversión a formas normales. Dependencias

En el apartado anterior se explicó en qué consistían las formas normales. En este se aplicarán esos conocimientos a la hora de crear las tablas.

Primero resulta conveniente repasar las formas normales:

- 1FN: cada campo puede tener un solo valor.
- 2FN: todos los atributos no primarios dependen de forma completa de la clave candidata.
- 3FN: no hay dependencias transitivas.
- FNBC: no hay claves compuestas interpuestas.
- 4FN: no existen dependencias multivaluadas.
- 5FN: las dependencias de reunión se deducen de la clave candidata.

Ahora con una serie de ejemplos se verá cómo transformar tablas no formalizadas en tablas que cumplan las formas normales.

1FN: Primera forma normal

Esta es la forma normal más sencilla. Simplemente restringe que en un determinado campo no pueda haber más de un valor.

Por ejemplo, una tabla de clientes en las que un cliente puede tener varios números de teléfono.

Nombre	Apellidos	Teléfono
Paco	Jiménez	64213899 64984213
Luisa	Pérez	68383883 63883632

Esta tabla no puede tener varios valores para cada columna. Para arreglarlo tienen que crearse tablas que contengan la lista de valores. Quedaría así:

Tabla Clientes:

Nombre	Apellidos
Paco	Jiménez
Luisa	Pérez

Tabla Cliente_Teléfono:

Nombre	Apellidos	Teléfono
Paco	Jiménez	64213899
Paco	Jiménez	64984213
Luisa	Pérez	68383883
Luisa	Pérez	63883632

Se convierten atributos multivaluados en tablas que se relacionan con la tabla original.

A veces se siente la tentación de utilizar un solo campo para almacenar varios valores añadiendo separadores, por ejemplo, una lista de teléfonos separada por comas.



Nota

Usar un solo campo para almacenar varios valores añadiendo separadores haría que la tabla no estuviera formalizada aunque en algunos casos puede llegar a ser lo óptimo.

2FN: Segunda forma normal

En este paso la preocupación se centra en los atributos que no dependen de forma completa de la clave primaria.

A partir de este paso, deben tenerse bien identificadas las dependencias funcionales entre columnas para poder proceder.

Siguiendo el ejemplo que pusimos en el anterior apartado:

Autor	Libro	Autor ciudad
Manuel	Informática para amas de casa	Sevilla

Manuel	Programación web para novatos	Sevilla
Jose	Montaje de ordenadores	Barcelona

Ahí “Autor ciudad” no dependía de forma completa de la clave, porque solo dependía de Autor y la clave es (Autor, Libro).

Para eliminar esta dependencia, se creará una tabla de Autores y se moverá el campo “Autor Ciudad” a esa tabla. Quedaría así:

Tabla Autor_Libros:

Autor	Libro
Manuel	Informática para amas de casa
Manuel	Programación web para novatos
José	Montaje de ordenadores

Tabla Autor:

Autor	Autor ciudad
Manuel	Sevilla
José	Barcelona

Ahora, si se quisiera saber la ciudad de un autor, simplemente tendrían que unirse las tablas Autor_Libros y Autor y se obtendría el valor.

Así se eliminan duplicados de la tabla principal y en caso de tener que cambiar la ciudad de un autor, solamente habría que cambiar un registro.



Actividades

17. Normalice la siguiente tabla:

Id_Ciudad	Id_País	Ciudad_nombre	Pais_nombre
300	22	México	México
301	22	Tijuana	México
111	10	Toledo	España
200	10	Murcia	España

3FN: Tercera forma normal

Aquí se intenta eliminar dependencias transitivas entre campos que no son clave candidata. Siguiendo el ejemplo del apartado anterior:

Campeonato	Ganador	Nacionalidad
2004 Formula One World Drivers' Championship	Michael Schumacher	Alemana
2005 Formula One World Drivers' Championship	Fernando Alonso	Española
2006 Formula One World Drivers' Championship	Fernando Alonso	Española

La Nacionalidad dependía del Ganador y ambos dependían de la clave primaria (por lo que está en 2FN).

Para solucionar el problema se haría algo similar a lo aplicado para la 2FN: crear una tabla de Ganadores y mover el campo Nacionalidad a esa tabla. Quedaría así:

Tabla Campeonatos

Campeonato	Ganador
2004 Formula One World Drivers' Championship	Michael Schumacher
2005 Formula One World Drivers' Championship	Fernando Alonso
2006 Formula One World Drivers' Championship	Fernando Alonso

Tabla Campeonato_Ganador

Ganador	Nacionalidad
Michael Schumacher	Alemana
Fernando Alonso	Española

Si se quisiera obtener la nacionalidad del ganador de un campeonato simplemente tendrían que unirse las tablas para obtener el valor.

FNBC: Forma normal de Boyce-Codd

Aquí se trataba de tener cuidado con claves candidatas múltiples que se interpusieran entre sí. Siguiendo el ejemplo del apartado anterior:

Profesor	Asignatura	Alumno
Paco	Lengua	José
Luisa	Filosofía	Marta
Rafael	Matemáticas	Luis

Las claves candidatas eran (Profesor, Asignatura) y (Asignatura, Alumno). Se ve que se interponen, por lo que no está formalizada. Para solucionarlo se crearán dos tablas:

Profesor_Asignatura

Profesor	Asignatura
Paco	Lengua
Luisa	Filosofía
Rafael	Matemáticas

Asignatura_Alumno

Asignatura	Alumno
Lengua	Jose
Filosofía	Marta
Matemáticas	Luis

Aquí lo que se busca es separar tablas que estarían mejor organizadas por separado que en una única tabla.

Es difícil determinar que una tabla no está en FNBC a la hora de crear el modelo lógico de datos, es mucho más sencillo de apreciar cuando la tabla ya tiene datos y pueden verse. En ese caso se ve rápidamente que hay datos que no se relacionan correctamente entre sí.

En muy raros casos una tabla que está 3FN no estará en FNBC. La manera más sencilla de saber si no está en FNBC es listar las claves candidatas y si dos de ellas comparten algún campo, es que no lo está.

4FN: Cuarta forma normal

Aquí había que encargarse de eliminar dependencias multivaluadas en las tablas. El ejemplo del apartado anterior era demasiado exagerado, para mejorarlo seguramente tendrían que crearse claves primarias numéricas.

Siguiendo el ejemplo del apartado anterior:

Producto	Color	Talla
Calcetines tobilleros	Negro	38-40
Calcetines tobilleros	Negro	40-42
Calcetas	Blanco	40-42

Aquí por cada producto hay varios colores y por cada producto y color hay varias tallas. En este ejemplo, lo ideal sería asignar un valor numérico a la relación entre producto y color, porque si no, la segunda tabla acabaría quedando igual.

Tabla Producto

Producto_ID	Producto
1	Calcetines tobilleros
2	Calcetas

Tabla Producto_Color

Producto_Color_Id	Producto_Id	Color
1	1	Negro
2	2	Blanco

Tabla Producto_Color_Talla

Producto_Color_Id	Talla
1	38-40
1	40-42
2	40-42

Si no se asignara un valor numérico tampoco pasaría nada. La tabla quedaría igual que al principio, pero estaría formalizada, puesto que tendría una tabla para cada producto, otra para cada color de producto y una tabla para cada talla de color de producto.

5FN: Quinta forma normal

En esta forma normal se trataba de plasmar las restricciones del problema real a la base de datos. Es decir, que se vean las restricciones de datos implícitas en los datos, las cuales no se verían de otra manera.

Siguiendo el ejemplo del apartado anterior:

Médico	Tratamiento	Paciente	Fecha
Dr. Torreiglesias	Anestesia	Manuel	20/09/2013
Dr. Torreiglesias	Vacuna	Marcos	21/09/2013
Dra. Gómez	Radiografía	Alberto	18/09/2013
Dra. Díaz	Anestesia	María	19/09/2013

Aquí la restricción era que un médico solo podía hacer una serie de tratamientos. Para que quede reflejado simplemente habría que crear una tabla extra que relacione médicos con tratamientos, así podremos saber qué tratamientos puede realizar cada médico.

Quedaría así:

Tabla Medico_Tratamiento_Paciente_Fecha

Médico	Tratamiento	Paciente	Fecha
Dr. Torreiglesias	Anestesia	Manuel	20/09/2013
Dr. Torreiglesias	Vacuna	Marcos	21/09/2013
Dra. Gómez	Radiografía	Alberto	18/09/2013
Dra. Díaz	Anestesia	María	19/09/2013

Tabla Medico_Tratamiento

Médico	Tratamiento
Dr. Torreiglesias	Anestesia
Dr. Torreiglesias	Vacuna
Dr. Torreiglesias	Dieta
Dra. Gómez	Radiografía
Dra. Gómez	Radioterapia
Dra. Díaz	Anestesia

Como puede apreciarse, esta tabla contiene todas las posibles combinaciones.



Importante

Hay valores que luego no se usan en la tabla, pero definen la lista de valores y pueden útiles para los nuevos registros que añadan.



Actividades

18. Normalice la tabla que creó en la primera actividad de este apartado en la cual creó una tabla con el inventario de una tienda.
-



Aplicación práctica

Se acaba de reunir con un cliente, el cual le pide que le desarrolle una aplicación que le permita automatizar la gestión de su negocio.

La aplicación quiere gestionar los activos de una empresa de limpieza. La empresa cuenta con maquinaria, productos y vehículos. De la maquinaria, quiere almacenarse su número de identificación único, su marca y su modelo. Cada máquina puede o no tener una próxima fecha de calibración. Se desea saber cuál es esa fecha y además, contar con un histórico de todas las calibraciones de una máquina. Es posible que una máquina no necesite de calibraciones. De cada producto se quiere saber su nombre y su precio; de cada vehículo, su matrícula, marca y modelo y además, la fecha de la próxima inspección técnica, junto con un histórico de cada inspección, con fecha y resultado de la inspección.

Cada vehículo, además de inspecciones técnicas periódicas, puede sufrir averías, se desea contar con un histórico con la lista de revisiones y averías

de un vehículo, así como con una descripción del problema, el nombre del taller, precio total y precio sin IVA y la fecha cuando se produjo la reparación.

- Dibuje un diagrama entidad/relación que represente este problema.
- Transforme el diagrama en un esquema relacional.
- Normalice las tablas resultantes hasta 3FN.

SOLUCIÓN

Esquema entidad/relación:



Esquema relacional:

- maquinaria (maquinaria_id, maquinaria_marca, maquinaria_modelo, maquinaria_proxima_calibracion)
- maquinaria_calibración (calibracion_id, calibracion_fecha, maquinaria_id)
- producto (producto_id, producto_nombre, producto_precio)
- vehículo (vehiculo_matricula, vehiculo_marca, vehiculo_modelo, vehiculo_proxima_inspección)
- vehículo_inspección (inspeccion_id, inspeccion_fecha, inspeccion_resultado, vehiculo_matricula)
- vehículo_revisión (revision_id, revision_fecha, revision_descripcion, revision_taller, revision_precio, vehiculo_matricula).

Normalización:

1. Las tablas se encuentran en 1FN porque no hay campos multivaluados.
2. Las tablas se encuentran en 2FN porque todos los atributos de las tablas

- dependen de forma completa de la clave primaria.
3. Las tablas se encuentran en 3FN porque no hay dependencias transitivas entre columnas.

Cabe destacar que el modelo de la maquinaria y de los vehículos puede que se encuentre denormalizado. Tendría sentido separarlo a tablas propias solo si se prevé que se necesite información relativa a los modelos o que sea necesario hacer listados o búsquedas por modelos concretos.

6. El modelo físico de datos. Ficheros de datos

En este apartado se explicará qué son los ficheros de datos. El propósito que se busca con ellos es transformar un **modelo de datos** en **archivos** convencionales. Estos archivos pueden ser binarios o de texto plano, dependiendo de las necesidades.

En el apartado 7 titulado “Transformación de un modelo lógico en un modelo físico de datos” veremos cómo transformar un modelo de datos en archivos de texto plano en los que almacenar los datos del modelo.

Los ficheros de datos tienen un gran número de limitaciones de fiabilidad, concurrencia y gestión de restricciones de integridad. En los inicios de la informática eran el único medio que había de almacenar información. Con el paso de los años fueron evolucionando para cubrir más necesidades y se idearon los Sistemas Gestores de Bases de Datos, que soportan el modelo relacional de forma nativa y son más escalables.

En este capítulo se explicará cómo manejar simples ficheros de datos para utilizar un modelo de datos. Y en el siguiente capítulo explicaremos en profundidad qué son los Sistemas Gestores de Bases de Datos, qué ventajas ofrecen, sus características y cómo utilizarlos.

6.1. Descripción de los ficheros de datos

Los ficheros de datos son unos archivos cuyo propósito es almacenar los datos de un modelo de datos en archivos simples.

Los ficheros de datos pueden ser binarios o contener texto plano, las diferencias entre estos dos tipos de fichero se estudiarán más adelante. Este capítulo se centrará en los ficheros de texto plano, puesto que son más utilizados y más fáciles de utilizar y de explicar.

Se vienen utilizando desde los inicios de la informática. En 1890 la oficina del censo de los Estados Unidos utilizó un sistema de tarjetas perforadas para automatizar el control del censo de la población. Una máquina se encargaría de leer las tarjetas perforadas y dependiendo de dónde se encontraran los agujeros, se podría saber la fecha de nacimiento, nombre, apellidos, etc., de cada persona. Este se podría decir que fue el primer uso que se le dio a los ficheros de datos. Cada tarjeta se correspondía con un fichero con datos sobre cada persona.

También se han venido utilizando desde hace mucho tiempo, por ejemplo, en las consolas de videojuegos para guardar el progreso en la partida de los jugadores.

Un ejemplo de fichero de datos puede ser el siguiente:

```
Id;Nombre;Departamento;Salario  
32;José Porras;Contabilidad;23000  
55;Pepe Arévalo;Dirección;100000
```

Como se aprecia, la primera línea del archivo contiene el nombre de cada atributo. Cada línea siguiente contiene un registro. Cada columna está separada por un punto y coma (;).

Este fichero podría estar siendo utilizado para guardar una lista de empleados de una empresa.

Este es un sistema muy fácil de utilizar, puesto que el **formato** de los ficheros de datos **es libre**. Es el diseñador de los mismos el que define el formato que tendrá. En vez de puntos y comas se podrían utilizar comas o tabuladores.



Nota

En vez de tener una primera línea con los nombres de las columnas, puede omitirse y obviarse qué nombre tiene cada una.

El siguiente apartado ofrecerá muchas variantes de ficheros de datos. El formato es libre, pero es aconsejable dejarse guiar por algún formato ya establecido para hacerlos estándar.

El sistema es muy sencillo y los datos pueden ser modificados con cualquier programa de edición de texto.

Cualquier usuario podría manipular ese fichero sin mucho conocimiento.

La contrapartida es que se presentan muchas restricciones:

- El formato debe permanecer intacto. Si se cambia el formato, las aplicaciones que usen el archivo dejarían de funcionar. Cambios en el fichero implican cambios en las aplicaciones que lo usen.
- Escalabilidad: puede contarse con un fichero de *log* de accesos a la aplicación. Es posible que durante los primeros cien mil accesos todo funcione perfectamente pero, ¿si hay 20 millones de accesos seguirá funcionando correctamente? Esto es algo que no se garantiza. Por un lado, el programador de la aplicación que utilice el fichero debe asegurarse de que los accesos al archivo son eficientes y de que no es necesario leer el fichero entero y colocarlo en la memoria RAM para su manipulación, de que los accesos son solo los justos y necesarios, etc.
Además, ¿qué pasaría si la aplicación que utilice ese archivo la utilizan millones de usuarios a través de Internet? ¿Será posible que miles de personas modifiquen el archivo al mismo tiempo? El administrador de sistemas tendría que asegurarse de que el archivo está siempre accesible. Llegaría un momento en que sería insostenible.
- Conurrencia: piense qué pasaría en el siguiente caso: el usuario (A) lee el archivo, a continuación el usuario (B) lee el archivo, después el usuario (A) modifica el archivo. Por último, el usuario (B) modifica el archivo. A no ser que la aplicación implementara un sistema de concurrencia propio, las modificaciones del usuario (A) se perderían por completo y serían sobrescritas por el usuario (B).
- Restricciones de integridad: la aplicación se debe encargar de que los datos mantengan la coherencia, de que no haya duplicados y de que las relaciones entre datos se mantengan. En un fichero de datos sería posible tener empleados con el mismo DNI, pedidos asociados a productos que no existen, etc. Recae en la aplicación toda la responsabilidad de que los datos sean correctos.

Aunque se han mencionado algunos de sus inconvenientes, siguen siendo útiles. No

siempre un Sistema Gestor de Bases de Datos es necesario. Los ficheros pueden ser útiles en los siguientes casos, por ejemplo:

- **Transferencia de gran cantidad de información:** los ficheros de datos por lo general ocupan poco espacio y permiten transferir datos entre varias localizaciones de forma muy rápida y eficiente.
- **Comunicación entre varios sistemas diferentes:** por ejemplo, un programa escrito en Python que necesite comunicarse con otro escrito en C. O un navegador web utilizando Javascript que desee comunicarse con un servidor utilizando AJAX.
- **Gestión de datos en entornos con restricciones de rendimiento:** por ejemplo, en las consolas antiguas, cada byte de almacenamiento era valioso. La libertad al usar el formato que se desee y el pequeño tamaño de los ficheros los hacía ideales para almacenar la información.
- **Gestión de datos muy simples sencilla:** si los datos apenas tienen relaciones y son muy simples, la opción correcta puede ser tener un archivo que contenga esos datos. Por ejemplo, una libreta de direcciones se puede almacenar en un fichero porque es muy simple, solamente contiene personas con su nombre, *e-mail*, teléfono, dirección y poco más.

6.2. Tipos de ficheros

Como ya se ha mencionado, los ficheros de datos se pueden dividir entre los ficheros de texto plano y los ficheros binarios.

Además de esa división, pueden agruparse según muchos criterios de acuerdo a su formato.

Tipo de dato almacenado

Según el tipo de dato almacenado:

- **Binarios:** los datos se almacenan en binario. Por lo general, solo el programa que los creo será capaz de manipularlos. Ofrecen total libertad al programador, puesto que además de texto, pueden almacenar estructuras informáticas internas (como *arrays*, objetos, estructuras, variables, etc.).
- **De texto plano:** los datos se guardan en ficheros de texto. Cualquier programa de edición de texto puede abrirlos. Generalmente, cada línea corresponde a un

registro. Son los más fáciles de utilizar y de manipular puesto que es sencillo ver qué contienen.

Esta unidad se centra en archivos de texto, puesto que son los más fáciles de utilizar y de explicar.

Los ficheros de texto plano pueden agruparse según diferentes criterios.

Formato de columnas

Como se vio en el ejemplo, el tipo de fichero más común está compuesto por líneas que se corresponden con cada registro y por columnas. Hay varias maneras de definir cuál es cada columna: ancho fijo, delimitadores de columnas y lenguajes de etiquetado.

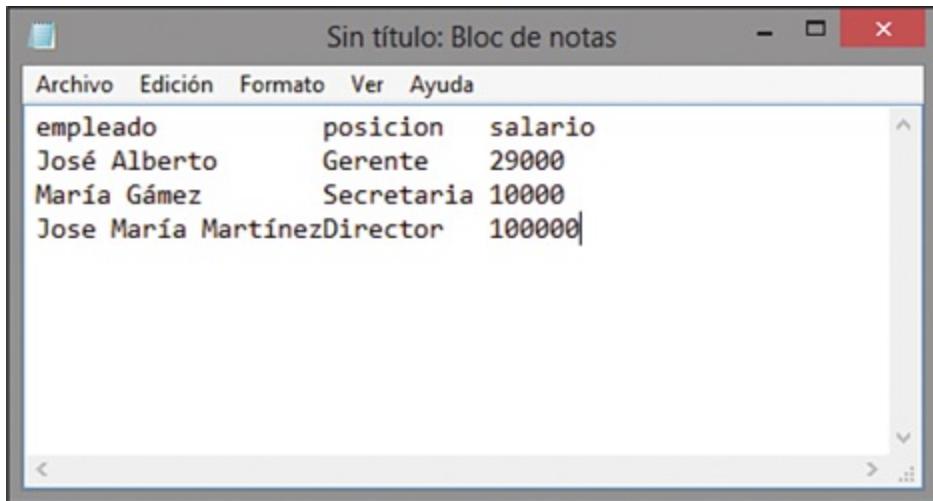
Ancho fijo

Cada columna tiene un ancho especificado. Por ejemplo, 20 caracteres. El espacio sobrante suele rellenarse con espacios (que pueden ir a la derecha o a la izquierda) o si se trata de un número, con ceros a la izquierda hasta completar el ancho. Si un dato tiene un tamaño superior, su valor será truncado.

La principal ventaja de este formato es que el rendimiento es máximo. El programa que lea el fichero simplemente tiene que contar el número de caracteres para obtener los valores; no tiene que preocuparse de nada más.

Además, si estos ficheros están comprimidos, hay técnicas de compresión muy avanzadas que detectan los espacios inútiles del archivo y reducen el tamaño final considerablemente.

Su principal inconveniente es que no son muy flexibles y no aprovechan bien el espacio en disco. Si un dato ocupa más que el ancho de columna, no se puede almacenar completo. Si un dato ocupa menos, se está desperdiando espacio en disco con bytes inútiles. Esto se puede mejorar utilizando técnicas de compresión, pero entonces se perdería algo de velocidad de lectura.



empleado	posicion	salario
José Alberto	Gerente	29000
María Gámez	Secretaria	10000
Jose María Martínez	Director	100000

Cada columna tiene un tamaño específico. Resulta bastante fácil de leer tanto para las personas como para las computadoras, aunque tiene limitaciones de manipulación de datos.

Delimitadores de columna

Cada columna está delimitada por un carácter. Por ejemplo, una coma, un punto y coma o un tabulador.

Su ventaja es que es un sistema flexible. Cada columna puede ocupar todo el espacio que necesite y no hace falta desperdiciar espacio llenando ceros o espacios en blanco.

Su mayor inconveniente es que son más difíciles de gestionar por la aplicación que utilice el fichero.

Debe implementar un sistema de **escape de caracteres** para asegurarse de que el carácter delimitador de columna no es interpretado incorrectamente.



Ejemplo

Piense en un fichero separado por columnas en el que un registro sea:

Nombre, aficiones, edad
Paco Pérez, televisión, fútbol, correr, 20

En este caso, la segunda columna no podría ser interpretada correctamente. Se consideraría que la edad sería “fútbol”. La aplicación debe considerar los caracteres de escape para poder saber si se trata de un fin de columna o de datos

dentro de la columna.

Esto se puede arreglar así:

“Paco Pérez”, “televisión, fútbol, correr”, “20”

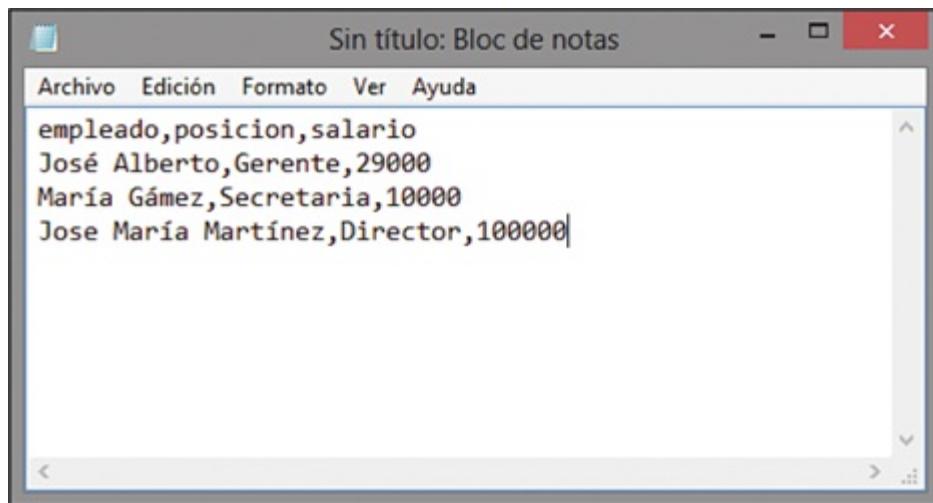
Como se aprecia, cada columna está dentro de comillas, por lo que se sabe que lo que esté dentro de comillas debe ser considerado como dato dentro de la columna.

Otra manera puede ser la siguiente:

Paco Pérez, televisión\, fútbol\, correr, 20

Se escapa el carácter de coma poniendo una barra invertida delante. La aplicación solo considerará las comas que no estén escapadas.

Independientemente del sistema, todas estas consideraciones no son necesarias en ficheros de ancho fijo.



Cada columna se separa por un delimitador. Un poco más difícil de leer pero con menos limitaciones.

Lenguajes de etiquetado

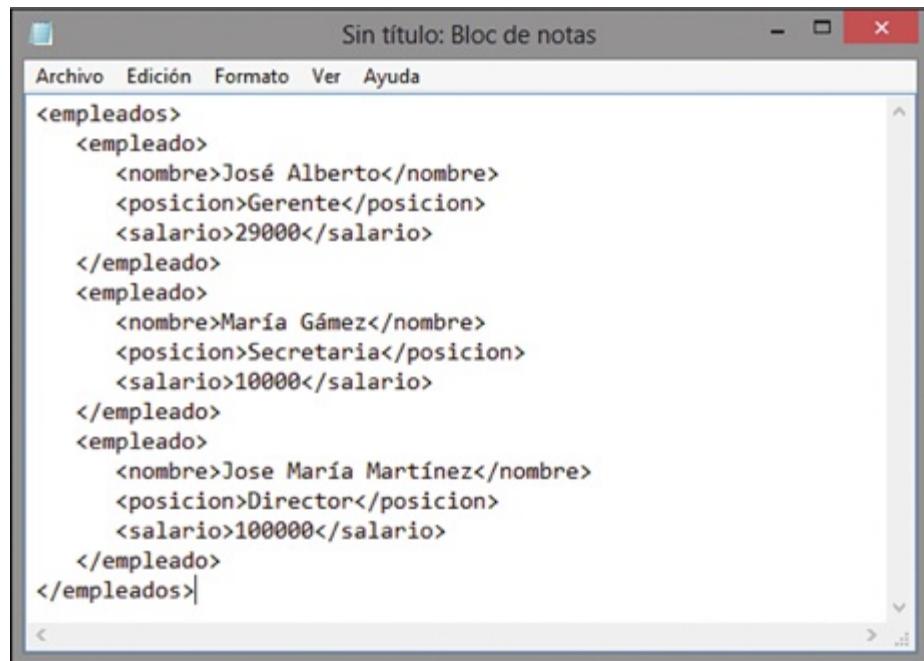
Como XML, en este caso, toda una sintaxis específica es utilizada para delimitar los datos. Cada lenguaje tendrá una sintaxis muy diferente, aunque el XML es el más extendido.

Su ventaja es que siguen un estándar fijado, por lo tanto, son consistentes y

pueden ser leídos desde cualquier aplicación que implemente ese protocolo.

Su desventaja es que añaden mucha complejidad a los ficheros. Tratar con ellos implica mucho más trabajo y además, los ficheros ocupan más porque requieren de caracteres extra para definir cada dato.

Además del XML hay otro lenguaje muy extendido en aplicaciones web que es el formato JSON (Java Script Object Notation). Este formato es muy simple. Es el formato que utiliza la función eval() de Javascript para convertir texto en código fuente de Javascript. Cada vez es más utilizado porque ocupa menos espacio que el XML, es más fácil de leer y es más fácil de utilizar desde Javascript.



The screenshot shows a Windows Notepad window titled "Sin título: Bloc de notas". The menu bar includes Archivo, Edición, Formato, Ver, and Ayuda. The main content area contains the following XML code:

```
<empleados>
  <empleado>
    <nOMBRE>José Alberto</nOMBRE>
    <posICION>Gerente</posICION>
    <salARIO>29000</salARIO>
  </empleado>
  <empleado>
    <nOMBRE>María Gámez</nOMBRE>
    <posICION>Secretaria</posICION>
    <salARIO>10000</salARIO>
  </empleado>
  <empleado>
    <nOMBRE>Jose María Martínez</nOMBRE>
    <posICION>Director</posICION>
    <salARIO>100000</salARIO>
  </empleado>
</empleados>
```

Todo un sistema de etiquetas para representar los datos. Muy fácil de leer por humanos. Más complicado de leer y manipular por computadores, pero prácticamente sin restricciones.



```
{  
    "empleados": [  
        {  
            "nombre": "José Alberto",  
            "posicion": "Gerente",  
            "salario": "29000"  
        },  
        {  
            "nombre": "María Gámez",  
            "posicion": "Secretaria",  
            "salario": "10000"  
        },  
        {  
            "nombre": "Jose María Martínez",  
            "posicion": "Director",  
            "salario": "100000"  
        }  
    ]  
}
```

Utiliza la sintaxis de código Javascript para representar los datos. Fácil de leer por personas y muy fácil de leer y manipular por aplicaciones web, sobre todo.



Nota

Hay más formatos para compartir ficheros pero no son mayoritarios. Suelen cubrir necesidades específicas de distintos programas informáticos.

6.3. Modos de acceso

El modo de acceso es lo que más puede hacer variar el rendimiento a la hora de utilizar ficheros de datos. Puede distinguirse entre los siguientes modos de acceso los más comunes:

- **Acceso secuencial:** se lee el archivo desde el principio hasta que se encuentra el dato o los datos que se buscaban. Funcionaría como una cinta de cassette. No es óptimo puesto que hay que leer datos que no se van a utilizar.
- **Acceso directo:** permite ir a una posición concreta del fichero directamente. Es muy rápido a la hora de obtener datos, puesto que va directo a la parte del

fichero de interés.

- **Acceso indexado:** en este caso, el archivo contiene un índice que permite determinar dónde se encuentra cada fila de forma directa. Por ejemplo, un CD de audio tiene acceso indexado, porque es posible saber dónde está cada pista de audio de forma rápida.
- **Acceso por direccionamiento calculado:** se puede conocer dónde se encuentra un dato mediante cálculos informáticos. No se necesita recorrer todo el archivo, solo una parte.

Dentro del modo de acceso se pueden establecer varias diferencias:

- Si el fichero se lee directamente desde donde esté alojado (disco duro, internet, red local, *pendrive*, etc.) o si se aloja en memoria durante su utilización. Esto influirá bastante en el rendimiento puesto que los datos alojados en memoria se leen muy rápido. También se corre el riesgo de que en caso de apagón, los datos modificados que no se hayan guardado en el soporte original se pierdan.
- Si para manipular registros hay que leer el archivo entero y crear una copia que incluya esos cambios o si se pueden modificar directamente sobre el archivo físico los datos afectados.
- Si es necesario que no haya “huecos” en el fichero. Si se borra un registro, puede dejarse un hueco vacío que se llenará más adelante si se añade uno nuevo o por el contrario, puede que haya que crear una nueva copia del archivo y mover todos los datos de debajo una línea más arriba.

6.4. Organización de ficheros

A la hora de organizar ficheros, lo ideal es hacer que cada entidad se aloje en un archivo separado. Puede crearse un archivo grande que contenga toda la información necesaria, pero aumentaría la complejidad en el manejo y el rendimiento podría caer, sobre todo si se trabaja con archivos de texto.

Piense qué pasaría si tuviera que mezclar empleados, departamentos y nóminas en un mismo archivo de texto. Tendría que diferenciar muy bien en qué parte del archivo empiezan los empleados y dónde los departamentos.

Por eso la manera óptima de organizar los ficheros de datos es según su tipo: que cada tipo de dato vaya en un archivo separado. Todos los ficheros deberían ser del mismo tipo (ancho fijo, delimitado, XML, etc. para ser consistentes, pero cada fichero tendría columnas diferentes.



Nota

Se pueden organizar los ficheros de forma libre, no hay ninguna restricción explícita. Simplemente se debe ser consciente de cómo se van a utilizar los datos y adoptar la solución óptima.

7. Transformación de un modelo lógico en un modelo físico de datos

Ya se ha visto qué son los ficheros de datos, sus ventajas, inconvenientes y los tipos de fichero que hay. Ahora, se aprenderá a transformar un modelo lógico en un modelo físico de datos.

Tras haber dibujado el diagrama entidad/relación, haber utilizado las reglas de transformación para convertirlo en un esquema relacional y haber normalizado este esquema, se habrá obtenido el modelo lógico de datos.

Un modelo lógico es el paso anterior a la implementación del modelo relacional en un sistema informático real.

El modelo lógico listaba todas las tablas necesarias y sus columnas. Las relaciones se representaban o bien mediante tablas extra o mediante columnas extra llamadas claves foráneas.

Al pasar del modelo lógico a un modelo de ficheros físico, se deben seguir los siguientes pasos:

1. Determinar qué formato de fichero se utilizará. Una vez se tenga el modelo lógico se hará una idea de qué datos va a almacenar. Deberá utilizar el formato que mejor cumpla con las restricciones de rendimiento, accesibilidad y tamaño de la aplicación.
2. Cada tabla del modelo lógico será un archivo separado. Si el modelo lógico es realmente grande, pueden agruparse los archivos en subcarpetas. El nombre del archivo será el nombre de la tabla. Sin caracteres extraños. Los espacios se sustituirán por guiones bajos y los caracteres extraños serán eliminados del nombre.

3. Dentro de cada archivo, dependiendo del formato del fichero, podrá necesitar incluir el nombre de las columnas en la primera línea o no. Si este es el caso, se hará en cada archivo y se tratará de que las columnas que sean clave primaria vayan al principio.
4. Toda la gestión de la integridad en las relaciones y del tipo de dato que puede alojar cada columna será responsabilidad de la aplicación que utilice los archivos, por lo que no hay que hacer nada más.



Sabía que...

Aunque no se puedan especificar las relaciones y las cláusulas de integridad de forma implícita en el fichero de datos, es posible tener un modelo que cumpla todas las formas normales.

Si se establecen correctamente las tablas, columnas y dependencias entre ellas, no debe haber ningún problema.



Aplicación práctica

Se necesita realizar una aplicación informática. Se trata de una aplicación móvil, bastante sencilla. Los requisitos son los siguientes:

- La aplicación servirá para gestionar listas de tareas.
- Una lista de tareas tiene un nombre.
- Una lista de tareas está compuesta por varias tareas. Cada tarea tiene un título y se desea saber cuándo una tarea está completada o no.

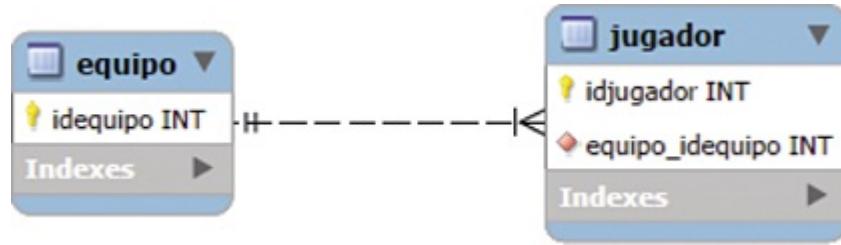
Dada la simpleza y dado que utilizar un Sistema Gestor de Bases de Datos sería bastante complejo en este entorno, se ha decidido utilizar un sistema de ficheros de datos para almacenar la información.

Diseñe el modelo conceptual y el modelo lógico para, a continuación, crear una estructura de ficheros que permita a una aplicación cumplir con los requisitos iniciales.

Esto permitirá a los programadores de la aplicación poder ver de un vistazo

cómo es el modelo de datos y, además, le servirá de estructura para empezar a escribir el código fuente de la misma.

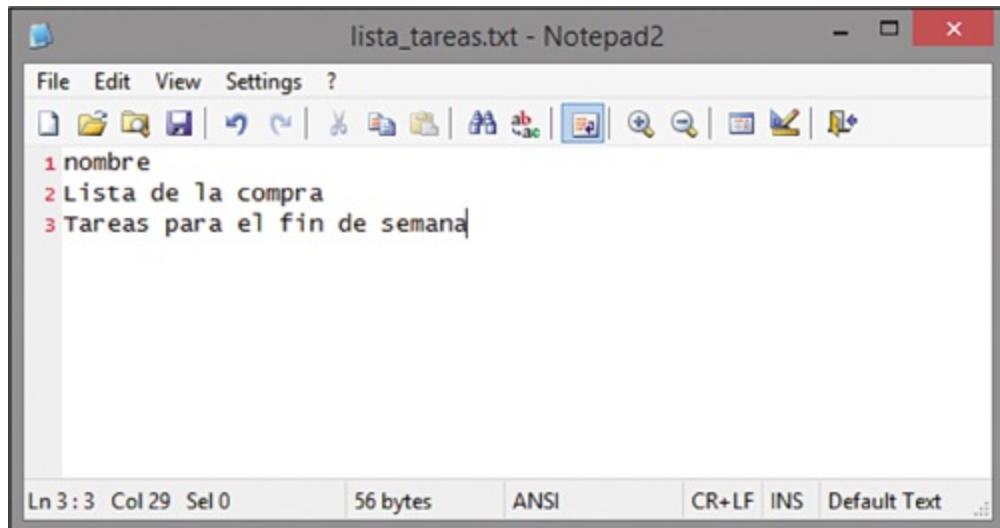
SOLUCIÓN



El modelo lógico sería:

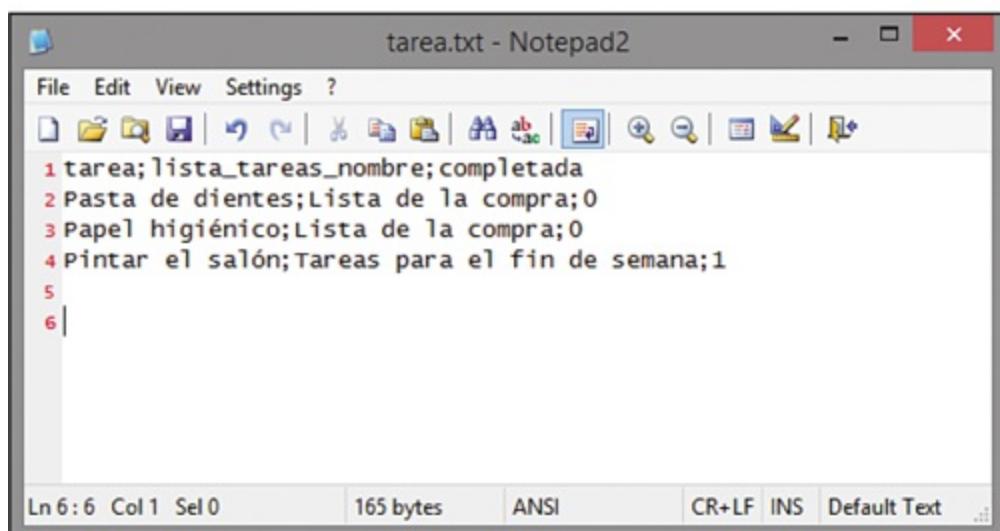
```
Lista_tareas (nombre)
Tarea (tarea, lista_tareas_nombre, completada)
```

Un sistema de archivos que sirviera para esto podría ser el siguiente:



```
File Edit View Settings ?
1 nombre
2 Lista de la compra
3 Tareas para el fin de semana
```

Ln 3 : 3 Col 29 Sel 0 56 bytes ANSI CR+LF INS Default Text



```
File Edit View Settings ?
1 tarea;lista_tareas_nombre;completada
2 Pasta de dientes;Lista de la compra;0
3 Papel higiénico;Lista de la compra;0
4 Pintar el salón;Tareas para el fin de semana;1
5
6
```

Ln 6 : 6 Col 1 Sel 0 165 bytes ANSI CR+LF INS Default Text

8. Herramientas para la realización de modelos de datos

Dibujar un modelo entidad/relación es una tarea bastante engorrosa porque por lo general, debe improvisarse dónde colocar cada entidad y si el modelo es muy grande, puede que no sea legible. Se requiere de mucho tiempo y cuanto más grande sea, más difícil resulta.

Hay muchas herramientas que ayudan a dibujar el modelo desde el ordenador. Simplemente soltando y arrastrando entidades, relaciones y escribiendo el nombre de las columnas visualmente.

Uno de los más conocidos es *Microsoft Visio*. El problema es que es un programa de pago y que solo funciona bajo *Windows*, aunque es posiblemente el más completo.

Aquí se utilizará *MySQL Workbench*, que es una herramienta gratuita y que además tiene versiones para *Windows*, *Linux* y *Mac OS*. Está enfocado principalmente a las bases de datos de *MySQL*, pero esto no afecta para nada al realizar modelos relacionales.



Nota

Posibles alternativas a MySQL Workbench son las siguientes:

- *Gliffy*: <<http://www.gliffy.com>> aplicación web gratuita para crear diagramas desde cualquier navegador.
 - *Libre Office Draw*: alternativa libre a *Microsoft Visio*. Incorporado en la suite ofimática Libre Office.
-

Aquí se prefiere este programa porque es multiplataforma, libre, gratuito y porque dado que el libro se orientará hacia *MySQL*, con este programa puede pulsarse un botón y crear la base de datos en *MySQL* directamente desde el modelo entidad/relación.

8.1. Instalación

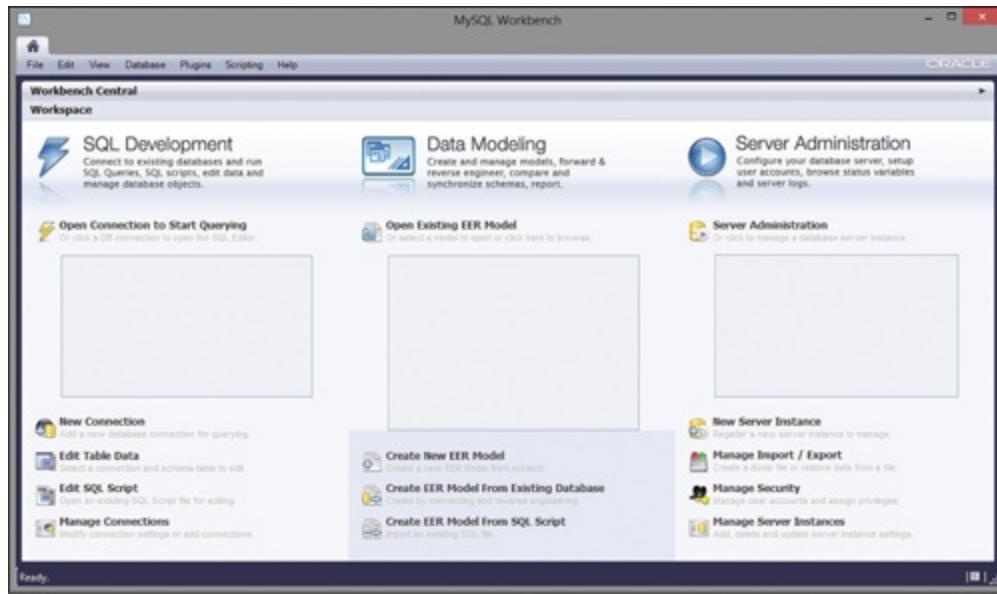
Se utiliza un motor de búsqueda para buscar *MySQL Workbench* y se descarga el programa desde la web de mysql.com.

Dependiendo del sistema operativo que se utilice, se seguirán unos pasos u otros.

Para las capturas de pantalla de la aplicación en este libro se mostrará el programa funcionando en *Windows*, pero todas las opciones deben ser muy similares en cualquier sistema operativo. Las capturas corresponden a la versión 5.2.45.

Primeros pasos

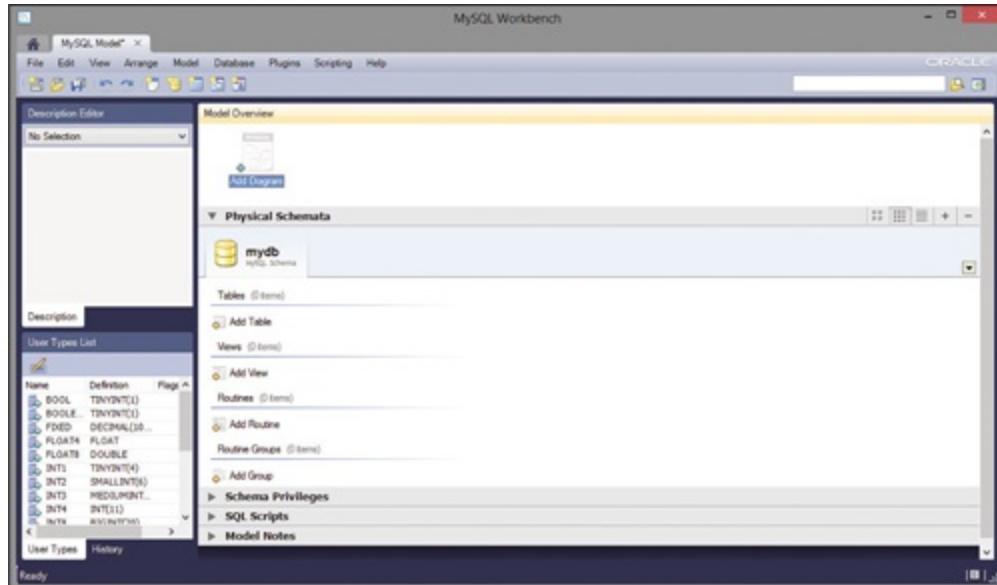
El programa está en inglés de momento. Aunque es muy fácil de utilizar y se entiende bastante bien.



Primera vez que se inicia MySQL Workbench.

La sección que más interesa para este capítulo es la de la columna central **Data Modeling**. Aquí es donde pueden crearse los modelos de entidad/relación. También está accesible en **File -> New Model**.

Para crear el primer modelo de entidad/relación se debe hacer clic en **Create New EER Model**. La primera opción.



Pantalla que aparece tras hacer click en *Create New EER Model*.

Aparece una página intermedia. Deberá hacerse doble clic en **Add Diagram**.

Entidad/relación vacío

Ahora habrá que enfrentarse con una cuadrícula vacía. Aquí es donde se irá arrastrando y soltando tablas y relaciones.

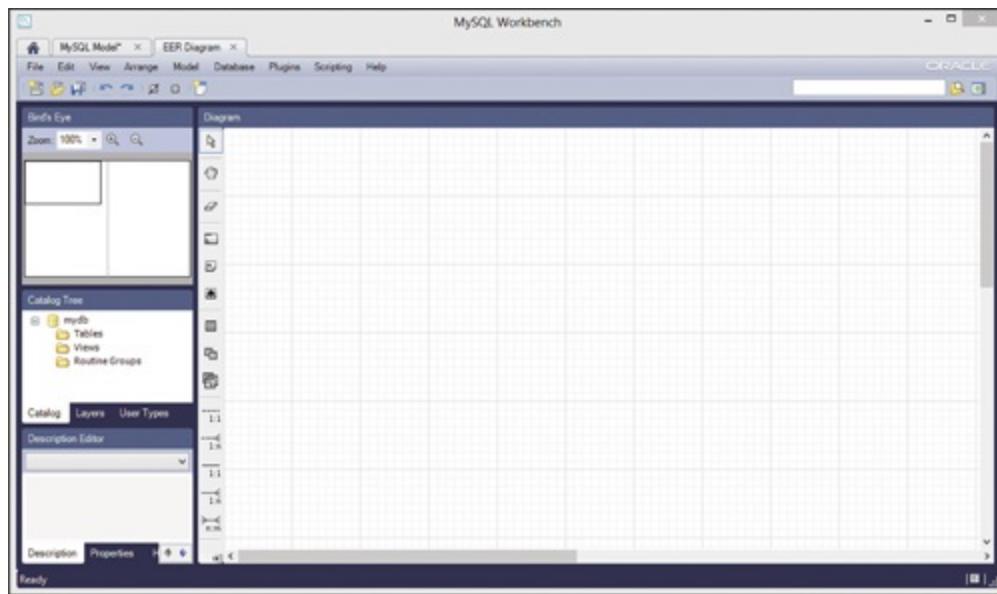
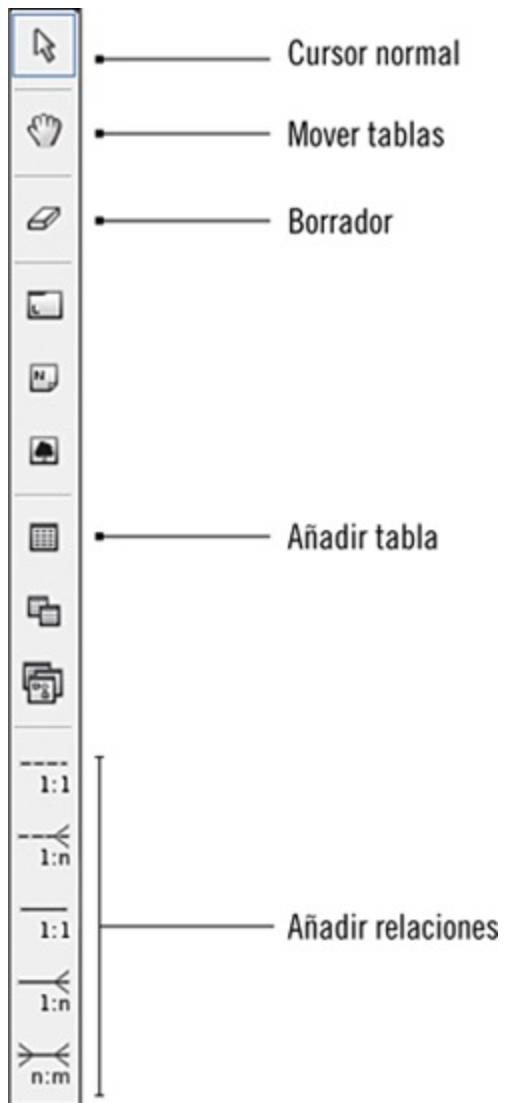


Diagrama entidad/relación vacío

En la columna de la izquierda es donde aparecen las principales opciones para añadir tablas y relaciones.

A continuación, se verán los significados de las principales opciones:



Opciones básicas más utilizadas

En la sección de añadir relaciones, la diferencia entre las líneas continuas y las discontinuas es para identificar si una relación es por identificación o no. Si una relación 1: N se establece con la línea continua quiere decir que la segunda entidad es débil y necesita la clave primaria de la primera identidad para formar su propia clave.



Importante

Normalmente, se utilizará siempre la línea discontinua.

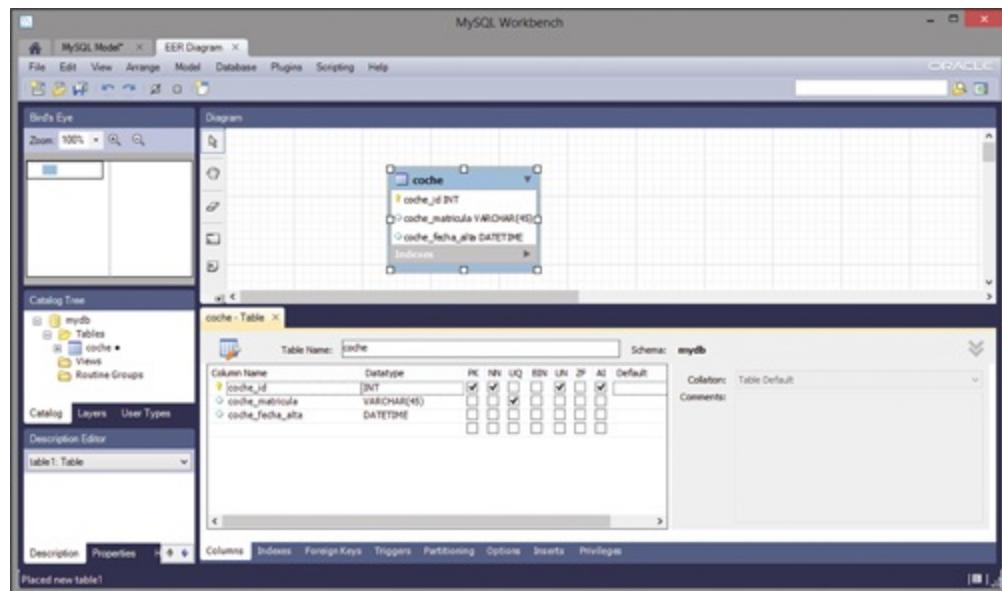
Crear tablas

Para crear una tabla se hace clic en el botón de **Place a new table**. Aparecerá vacía. Debe hacerse doble clic sobre ella y aparecerá debajo un panel con sus propiedades.

Desde ahí, habrá que modificar el nombre de la tabla, pudiendo especificar las columnas que tiene. Igualmente es posible especificar su tipo de dato.

También se podrán elegir otras opciones:

- PK: Primary Key. Especificar si esa columna pertenece a la clave primaria o no.
- NN: Not Null. Si no admite nulos. Por defecto se admiten nulos. Debe marcarse esta opción para no permitirlos. La clave primaria no puede contener nulos.
- UQ: Unique. Si el campo es clave única o no.
- BIN, UN, ZF y AI son opciones avanzadas que no es necesario especificar aquí. Se tratarán al estudiar SQL.



Crear una tabla

No deben incluirse las claves foráneas aquí. Las relaciones se definirán después y se crearán automáticamente.

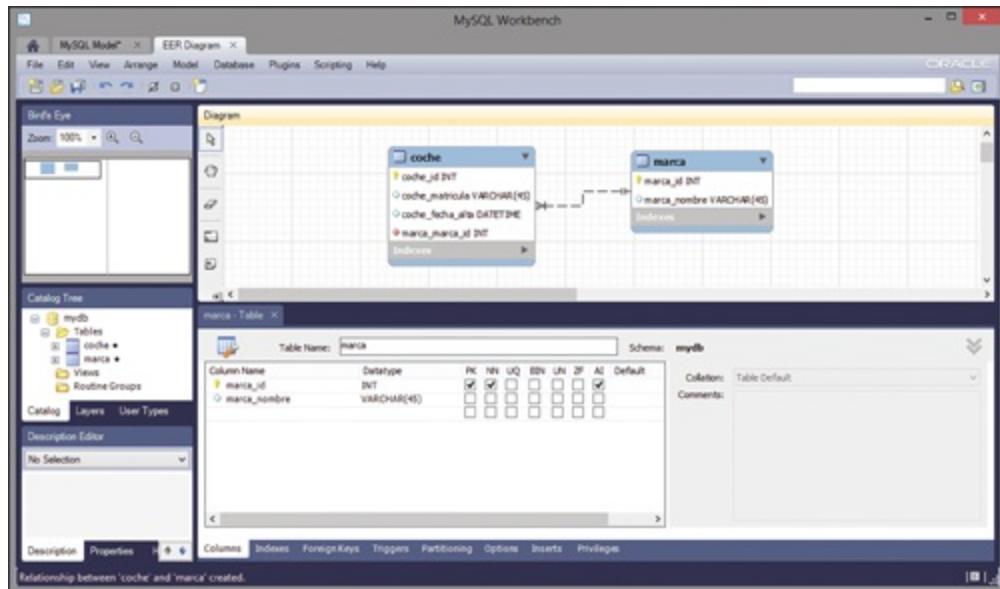
Crear relaciones

Tras crear esta tabla de coches, se creará una tabla de marcas, para relacionar ambas entre sí.

Para ello se hará clic en el tipo de relación que se vaya a utilizar. En este caso, la de 1: N discontinua.

A continuación, debe clicarse en las dos tablas que estén involucradas en la relación, pero el orden en que se haga es muy importante. Primero debe hacerse clic en la entidad 2 y luego en la entidad 1. Es decir, en este caso primero se hará clic en “coche” y luego en “marca”.

Piense que debe hacer clic primero en la tabla que necesitará una clave foránea y después, en la tabla a la que se referenciará la clave foránea.

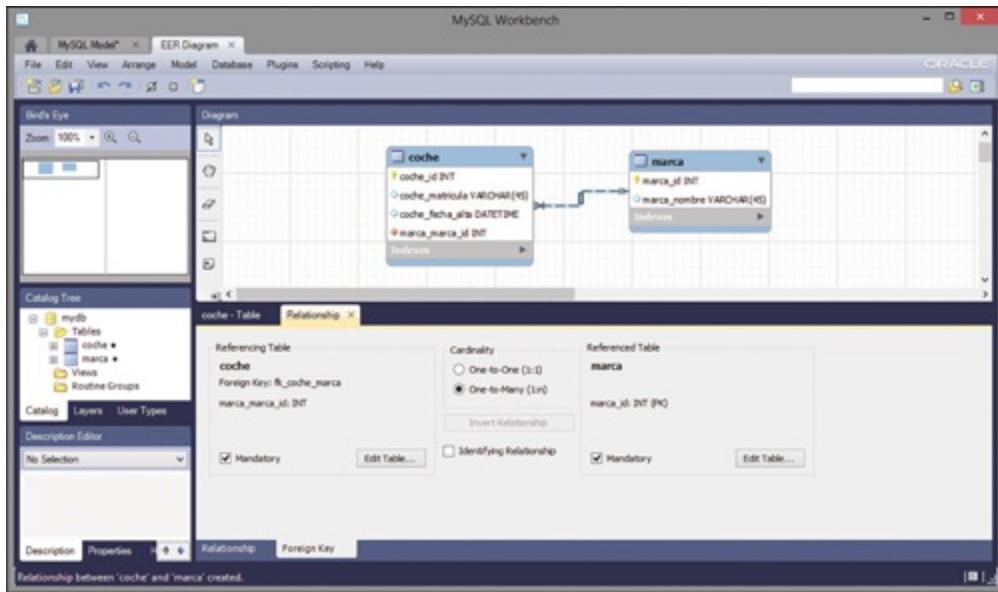


Crear una relación

Se aprecia que ya aparece la relación en pantalla. También puede que necesiten editarse las propiedades de la relación, sobre todo para definir las cardinalidades.

¿Qué pasaría si un coche pudiera tener marca o no? Sería una relación de 0: N en la cual la clave foránea debería admitir nulos.

Puede realizarse de forma manual editando la tabla “coche” y deseleccionando la opción **Not Null**. También puede hacerse doble clic en la relación para editar sus propiedades.



Propiedades de una relación

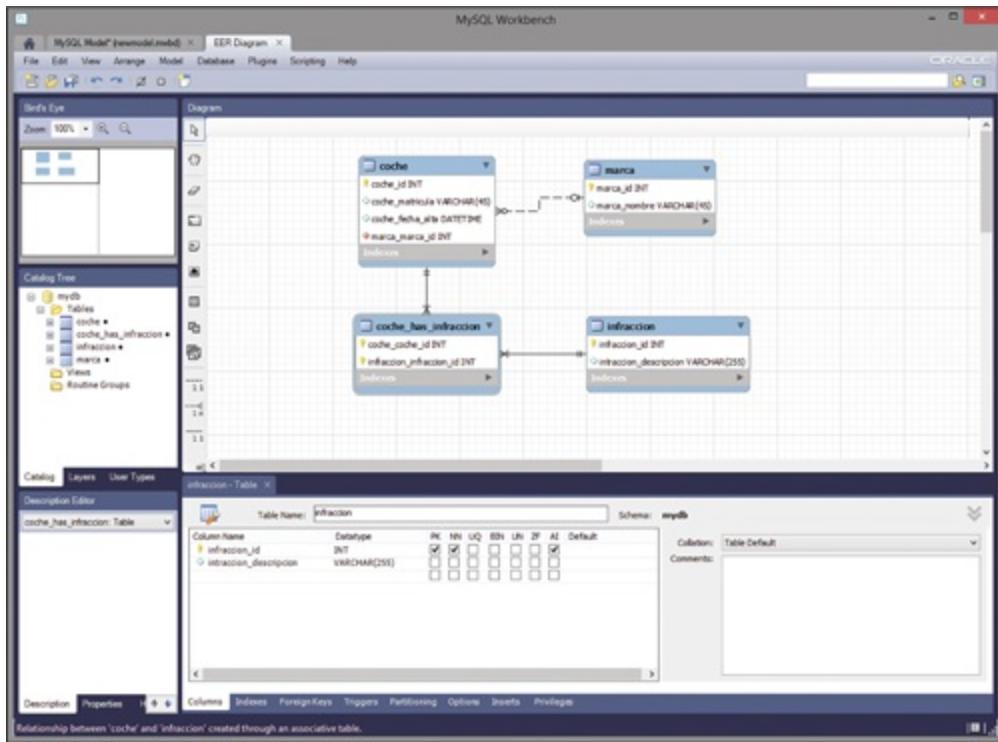
Pueden especificarse las propiedades de cada extremo de la relación. La opción de **Mandatory** (requerido) determina la cardinalidad mínima. Si no es requerido uno de los extremos, la relación cardinalidad mínima será de cero. Por ejemplo, si un coche puede no tener marca, quiere decir que la referencia de marca no es requerida. En este caso, supuestamente no sería posible tener marcas si no se tienen coches de esa marca; eso también puede ser modificado.

Dentro del panel de propiedades de la relación también puede cambiarse el tipo de relación, para que sea 1:1 o sea entre entidades fuertes o no.

Relaciones de muchos a muchos

Al utilizar una relación de muchos a muchos se verá que automáticamente se creará la tabla intermedia de relación. En vez de utilizar la notación de las patas de gallo en ambos extremos de la relación, se crea la tabla de relación. La tabla contendrá las claves primarias de las tablas involucradas.

Por defecto, su nombre será el nombre de la tabla, guion bajo y el nombre de la columna. El nombre de la tabla por defecto es tabla1, guion bajo, “has”, guion bajo tabla2.



Relación de muchos a muchos



Nota

Se le puede cambiar el nombre editando la tabla.

Tras haber establecido la relación, haciendo doble clic en cada parte de la relación se establecerán las cardinalidades.

Exportar el modelo de datos

Mientras se realiza el modelo de datos debe guardarse una copia para asegurarse de no perder el trabajo realizado. Se guardará por defecto en un archivo con extensión “mwb”. Este archivo solo servirá para editar el modelo de datos. Lo normal es que cuando se haya terminado, quiera imprimirse el dibujo o tenerlo en una imagen.



Recuerde

Este archivo solo servirá para editar el modelo de datos.

Puede exportarse el dibujo en formato PNG para poderlo utilizar como una imagen normal. Para ello vaya a **File -> Export -> Export as PNG**.

Si lo que quiere es imprimirla, lo más normal será generar un PDF. Para ello vaya a **File -> Print to PDF**.



Aplicación práctica

Su jefe le ha pedido que cree, utilizando *MySQL WorkBench*, un diagrama entidad/ relación visual que le permita ver rápidamente cómo está diseñada su base de datos.

Esta base de datos ya está creada y lleva tiempo funcionando, pero no se hizo este diagrama en su día, y ahora es útil para poder decidir las posibles mejoras que se le puedan hacer y para servir de documentación a los programadores.

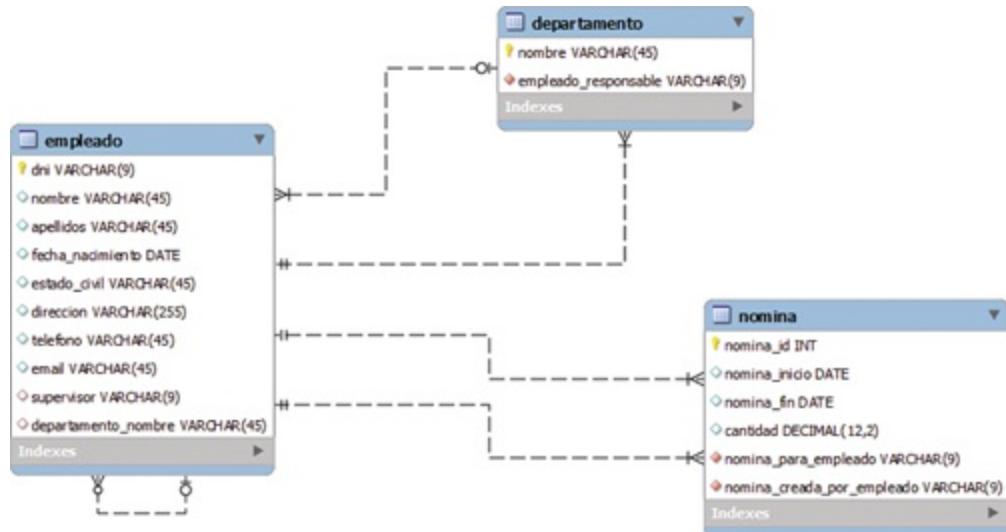
La base de datos tiene las siguientes características:

- **Se cuenta con empleados, departamentos y nóminas.**
- **Cada empleado tiene un DNI, nombre, apellidos, fecha de nacimiento, estado civil, dirección, teléfono y e-mail.**
- **Cada empleado puede tener un supervisor. Un empleado solo tiene un supervisor. Un empleado puede ser supervisor de varios empleados. Es posible que un empleado no supervise a nadie o que nadie le supervise.**
- **Cada departamento tiene un nombre y un empleado que es responsable del mismo.**
- **Los empleados forman parte de departamentos. Es posible que un empleado no pertenezca a ninguno (el jefe o alguien recién llegado, por ejemplo). Un empleado solo puede pertenecer a un departamento en un preciso instante.**
- **Una nómina contiene una fecha de inicio, fecha de fin, fecha de creación y cantidad a pagar bruta.**
- **Una nómina pertenece a un empleado. Además, se desea saber qué empleado creó esa nómina. Es decir, la nómina es para un empleado y la crea otro empleado.**

Dibuje el diagrama entidad/relación utilizando *MySQL WorkBench* y,

continuación, exporte los resultados en formato PDF.

SOLUCIÓN



9. Resumen

Cuando conozca los requisitos de una aplicación informática, debe seguir una serie de procedimientos para poder realizar una base de datos que permita desarrollar esa aplicación deseada.

El primer paso consistirá en dibujar un modelo conceptual utilizando diagramas entidad/relación. Esta será una representación gráfica del problema a resolver. Deberá identificar las entidades y las relaciones entre ellas.

Una entidad puede relacionarse con cero o con muchas entidades. Puede incluso relacionarse consigo misma.

Los diagramas entidad/relación pueden ser dibujados a mano o utilizando algún software informático, lo cual facilitará mucho el trabajo. *MySQL Work-Bench* es una buena opción para esto.

Una vez tenga el modelo conceptual, se pasará al modelo lógico, en el cual habrán de seguirse una serie de reglas de transformación para convertir un diagrama

entidad/relación en un esquema relacional, el cual se compone de tablas y atributos.

Este esquema relacional debe ser normalizado, para asegurarse que posteriormente, al explotar la base de datos, no aparezcan deficiencias, anomalías o problemas de rendimiento.

Para ello, se seguirán las convenciones de normalización, las seis formas normales.

Una vez hecho esto, puede considerarse completado el diseño lógico. Este diseño lógico ya está listo para ser implementado en cualquier soporte físico que se desee. Típicamente será un Sistema Gestor de Bases de Datos, pero si el modelo de datos no es muy complejo, es posible que se desee implementarlo utilizando un sencillo sistema de archivos de texto. La utilización de uno u otro dependerá de la complejidad de la aplicación y de las limitaciones técnicas que se tengan.



Ejercicios de repaso y autoevaluación

1. De las siguientes frases, indique cuál es verdadera o falsa.

Un dato es la unidad mínima de información.

- Verdadero
- Falso

Un modelo de datos es una representación gráfica de datos utilizados en una aplicación.

- Verdadero
- Falso

Se dice que un campo tiene un valor nulo cuando realmente se desconoce su valor o cuando aún no se le ha dado un valor.

- Verdadero
- Falso

2. ¿Qué diferencia hay entre una clave primaria, una clave candidata y una superclave?

3. Se necesita crear una web que gestione la cartera de valores de los usuarios registrados. Cada usuario cuenta con un nombre (único), contraseña y dirección de e-mail (única). Cada empresa cotizada cuenta con un identificador único y un nombre completo.

- a. Un usuario tiene una cartera, la cual está compuesta de valores. Cada valor corresponde a una empresa, un número de títulos y un valor de compra. a. Dibuje el diagrama de entidad relación que permitiría crear una base de datos que gestione esa información.
- b. Cree el modelo lógico de datos tras haber completado el modelo entidad/relación. Identifique las claves candidatas de la tabla de

usuarios y explique por qué utiliza como clave primaria una u otra.

- 4. Se necesita ampliar una aplicación para que solo determinadas partes de la misma sean accesibles por los usuarios, dependiendo de sus permisos. Se cuenta con usuarios (nombre, contraseña, fecha de creación). La aplicación se compone de secciones (nombre de sección). Cada sección puede tener subsecciones dentro, en una estructura anidada. Se desea controlar a qué secciones puede acceder cada usuario. Cree un modelo entidad/relación que permita crear un modelo de datos que contemple este caso.**
-
-
-

- 5. Cree un diagrama entidad/relación para un foro de Internet. Este es el enunciado:**

El foro se divide en subforos. Solo se permite un nivel de anidamiento. Un subforo se identifica por su nombre. Dentro de cada subforo, los usuarios (nombre, contraseña, e-mail, fecha creación. Nombre y e-mail son únicos) abren conversaciones (título, fecha creación). Una conversación tendrá varios mensajes. Siempre tendrá al menos uno (el mensaje incluido por el creador de la conversación). Un mensaje puede ser independiente o estar enviado en respuesta a otro mensaje de la misma conversación. Cada mensaje tiene un ID auto numérico generado por el sistema.

- 6. Dibuje el diagrama entidad/relación del siguiente problema:**

- Se cuenta con países, los cuales tienen nombre y código identificativo de tres caracteres.
- Cada país tiene una serie de estados/comunidades autónomas

(dependiendo del país). Se desea guardar el nombre de cada estado. Hay países sin estados, en este caso se guardaría en la base de datos un estado del mismo nombre que el país, con el fin de hacer funcionar la aplicación correctamente. Por lo que cada país tendrá al menos un estado.

- Cada estado tiene una serie de condados o provincias (dependiendo del país). De las provincias se quiere guardar su nombre y unas coordenadas GPS para saber dónde se encuentran.
 - Pista: cada nombre de estado o condado no es único, porque se puede repetir en otro país. Las relaciones deben hacerse mediante entidades débiles por identificación.
-
-
-
-

7. Normalice la siguiente tabla argumentando cada paso.

Cliente	Teléfono	Área nombre	Área código	Área país
John Murphy	084988438 081288484	Cork	COR	Irlanda
Ryan Murray	074747747	Londres	LON	UK
Marisa Mayers	0748848	Londres	LON	UK

8. ¿Se encuentra la siguiente tabla normalizada? Razone la respuesta:

Torneo	Ganador	Finalista	Resultado
Indian Wells	Rafael Nadal	Juan Martín del Potro	4-6, 6-3, 6-4
Miami	Andy Murray	David Ferrer	2-6, 6-4, 7-6(1)
Montecarlo	Novak Djokovic	Rafael Nadal	6-2, 7-6(1)
Madrid	Rafael Nadal	Stanislas Wawrinka	6-2, 6-4
Roma	Rafael Nadal	Roger Federer	6-1, 6-3

9. ¿Se encuentra la siguiente tabla normalizada? Razone la respuesta:

Temporada	Jugador	Nacionalidad	Goles
2006/07	Kaká (AC Milán)	Brasil	10
2007/08	Cristiano Ronaldo (Manchester United FC)	Portugal	8
2008/09	Lionel Messi (FC Barcelona)	Argentina	9
2009/10	Lionel Messi (FC Barcelona)	Argentina	8
2010/11	Lionel Messi (FC Barcelona)	Argentina	12
2011/12	Lionel Messi (FC Barcelona)	Argentina	14
2012/13	Cristiano Ronaldo (Real Madrid CF)	Portugal	12

10. Determine las claves candidatas y la clave primaria óptima de la tabla de noticias.

Título	URL	Texto	Fecha
Carrera de atletismo	http://www.example.com/2010/carrera-atletismo.html	Una carrera de atletismo en la ciudad. Ganó Usain Bolt	27/08/2010
Concurso de tartas	http://www.example.com/2012/concurso-tartas.html	El próximo jueves tendrá lugar un concurso de tartas en la plaza del pueblo.	05/03/2012

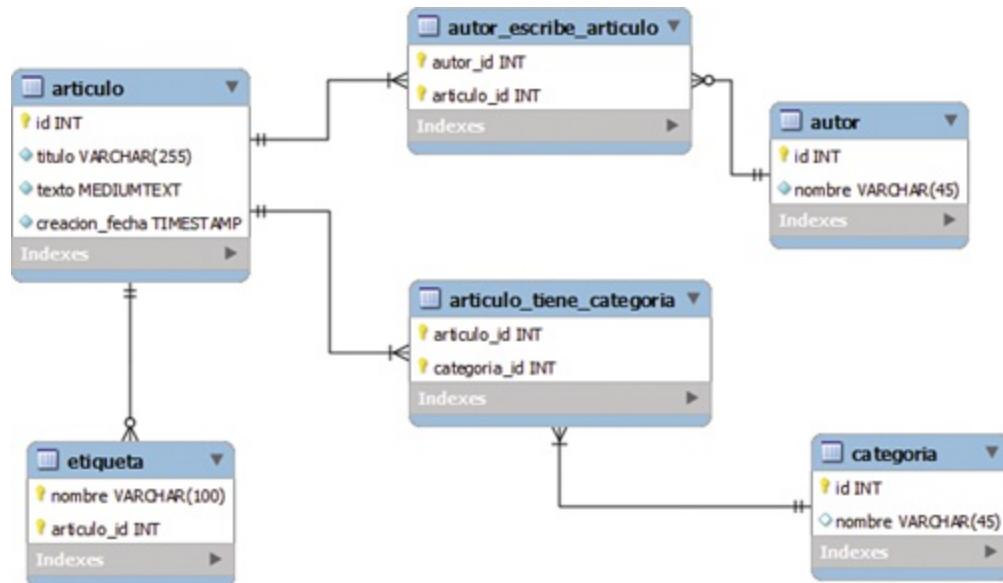
11. Determine las claves candidatas de la siguiente tabla de piezas.

Departamento	Código	Nombre	Dimensiones	Precio neto
AUDI	023	Recambio limpiaparabrisas	40 x 5	7.99
AUDI	020	Filtro aire	20 x 15	2.99

SEAT	023	Catalizador	50 x 30	129
VW	080	Catalizador	45 x 35	150

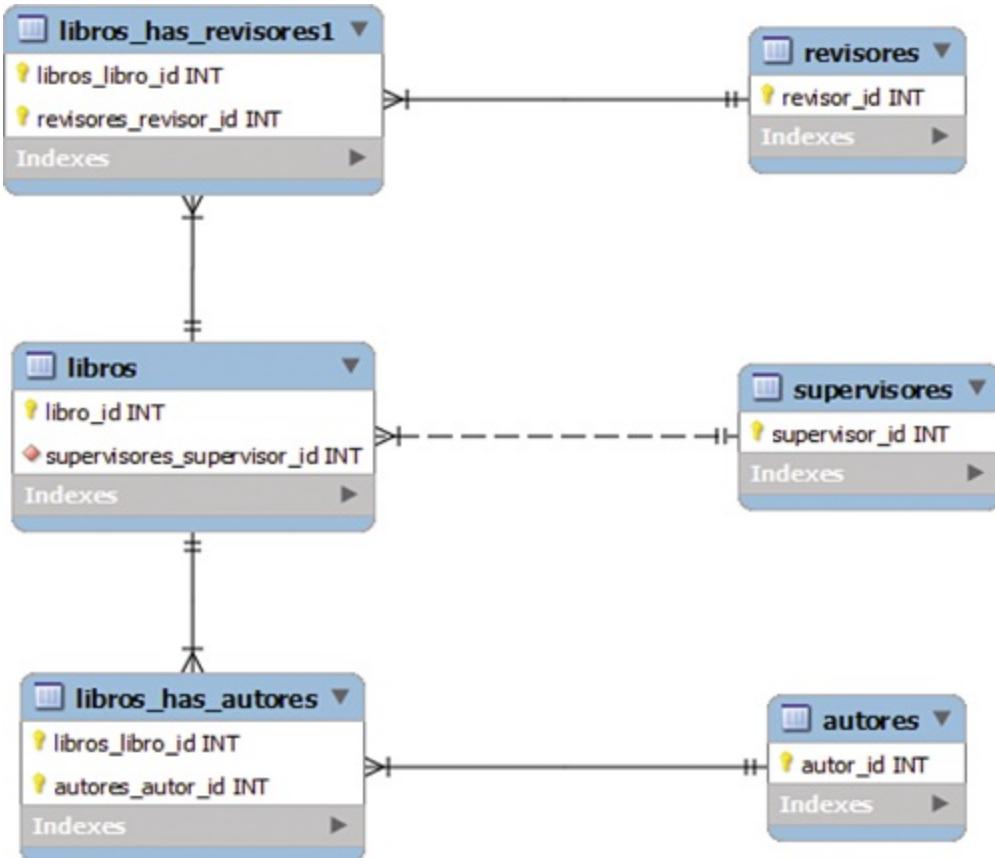
12. Cree la un fichero de datos que sirva para guardar los datos de transacciones con tarjetas de crédito. Los campos serán: ID transacción, número de tarjeta, fecha caducidad, código de seguridad (tres dígitos), cantidad y fecha. Utilice un fichero de texto delimitado por tabuladores.
-
-
-
-

13. Escriba el enunciado que habría supuesto crear este diagrama de entidad/relación:



14. Escriba el enunciado que habría supuesto crear este diagrama de entidad/relación:

Tenga en cuenta que solo se lista la clave primaria de cada entidad para que sea más sencillo. Solo es necesario que se centre en las relaciones.



15. Dibuje, utilizando MySQL WorkBench, el siguiente enunciado:

Va a desarrollarse un programa de reproducción multimedia. Para ello, se cuenta con el siguiente documento explicativo:

- Hay canciones. Las cuales tienen un título, artista, álbum, duración, ruta al archivo del sistema y tamaño en bytes.
- De cada artista se quiere guardar su nombre, nacionalidad y la ruta del archivo que contiene una foto del mismo (opcional). Se desea poder distinguir si un artista es un cantante solitario o un grupo.
- De cada álbum quiere saberse su nombre, el artista que lo creó y el año de lanzamiento.
- El álbum puede estar creado por un artista, pero que una de las canciones del álbum la cante otro artista. Hay una doble relación entre canciones y artistas, una a través del álbum y otra a través de la canción por sí misma.
- Se quiere que las tablas se encuentren bien normalizadas, por lo que las

tablas de canciones, artistas y álbumes deben estar relacionadas entre sí correctamente.

Capítulo 2

Sistemas de Gestión de Bases de Datos (SGBD)

1. Introducción

Ya se han visto los modelos de datos y la manera de convertir modelos lógicos de datos en ficheros de datos. El problema de los ficheros de datos es que tienen muchas limitaciones, sobre todo relacionadas con la escalabilidad. Por eso durante años de evolución de la informática, se fue viendo que los ficheros de datos eran claramente insuficientes y que era necesario un sistema más sofisticado que se encargara de trabajar con los datos de forma unificada.

Para ello, se crearon los sistemas gestores de bases de datos (a partir de ahora se nombrarán como SGBD, por sus siglas). Los SGBD representan una evolución enorme con respecto a los ficheros de datos.

Proporcionan interfaces para permitir que cualquier programa se conecte a ellos, encapsulan el tratamiento físico de los datos, eliminando complejidad innecesaria, incluyen mecanismos de escalabilidad y están altamente optimizados.

2. Definición de SGBD

Un SGBD (o DBMS *Data Base Management System* por sus siglas en inglés) es un conjunto de programas informáticos altamente sofisticados que gestionan bases de datos y proporcionan interfaces para comunicarse con aplicaciones externas.

Un SGBD puede ser utilizado para diferentes tipos de modelos de datos. Los más habituales son los SGBD relacionales, que implementan el modelo relacional de forma nativa. Hay otros tipos de SGBD, los cuales se tratarán más adelante.

Para que un SGBD sea relacional deberá tratar de cumplir las **12 reglas de Codd** explicadas en el primer capítulo.



Importante

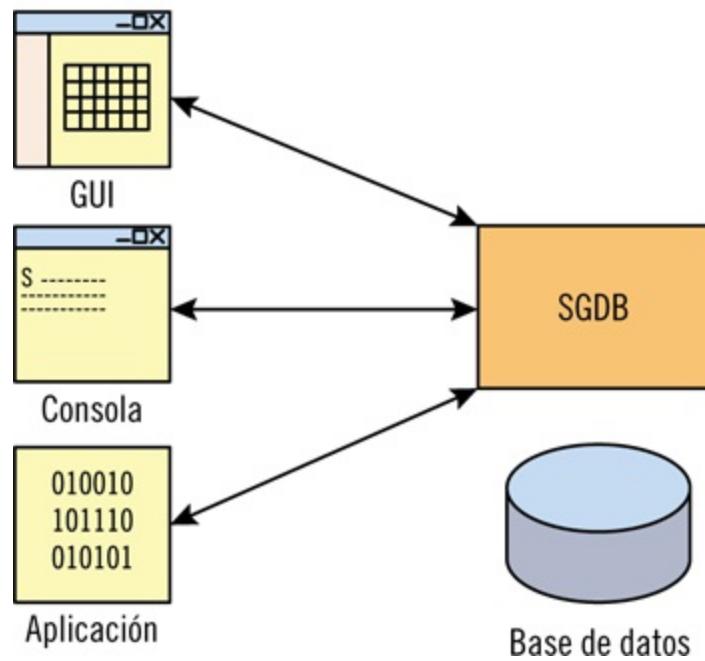
No todos los SGBD cumplen rigurosamente todos los puntos.

Básicamente, lo que se busca en un SGBD con respecto a un sistema de ficheros es lo siguiente:

- **Separación entre el modelo de datos y las aplicaciones que lo utilicen:** si se modifican los datos no deberían rescribirse las aplicaciones que utilicen esa base de datos.
- **Eliminación de redundancia e inconsistencia:** dado que todos los datos se encuentran centralizados, es posible no tener que copiar lo mismo en varios sitios. Además, como hay una interfaz única para modificar los datos, no es posible que varios programas los modifiquen de forma distinta.
- **Seguridad:** es posible limitar el acceso de los usuarios a solo ciertas partes de la base de datos. Proporciona todo un sistema de permisos para controlar qué puede hacer cada usuario.
- **Concurrencia:** es difícil hacer que un sistema de ficheros permita modificaciones concurrentes (simultáneas) y es fácil obtener copias inconsistentes de los datos. Un SGBD cuenta con una serie de herramientas para gestionar todo esto de forma transparente.
- **Integridad:** un SGBD es capaz de gestionar las restricciones de un modelo relacional de forma nativa, por lo que no es necesario que cada aplicación que use esos datos tenga que implementarlo a su manera. Esto elimina la posibilidad de que se incumplan las reglas de integridad y elimina complejidad a los programas que utilicen la base de datos.
- **Encapsulamiento del tratamiento físico de los datos:** es el SGBD el que se encarga de saber dónde guardar cada byte en el disco duro. El administrador de bases de datos (DBA por sus siglas en inglés) solo se encarga de definir el modelo de datos y trabaja desde un nivel superior. No necesita saber cómo se guardan los ficheros internamente para utilizarlos.

Este libro se centrará en los SGBD relacionales. Hay otros tipos de SGBD, aunque los que más relevancia tienen son los que implementan el modelo relacional de forma nativa.

Esquema de un SGBD



Un SGBD debe implementar las siguientes características:

- **Lenguaje de definición de datos (DDL *Data Definition Language*):** sistema que permita crear las bases de datos, sus tablas, tipos de datos, restricciones, etc. Además, debe proporcionar una manera de acceder al diccionario de datos (siguiendo la regla 4 de Codd).
- **Lenguaje de manipulación de datos (DML *Data Manipulation Language*):** sistema para que mediante consultas se puedan manipular los datos (siguiendo la regla 5 de Codd).
- **Gestión de permisos (DCL *Data Control Language*):** debe contar con mecanismos de seguridad que controlen qué permisos tiene cada usuario de la base de datos.
- **Gestión de restricciones de integridad:** debe gestionar las restricciones de los datos de forma inherente. No debe ser posible manipular datos incorrectamente de forma que se pierda la coherencia de los mismos.
- **Gestión de concurrencia:** sistema complejo que permite que varios usuarios manipulen la base de datos de forma simultánea sin corromper la base de datos y manteniendo el rendimiento.
- **Gestión de copias de seguridad:** debe ser posible recuperar la base de datos en caso de fallo en el sistema.

A continuación, se verá con más profundidad cada elemento de un SGBD, cómo se

gestiona el almacenamiento físico de los archivos, el mecanismo de consultas, etc.



Ejemplo

Ejemplos de SGBD:

- *MySQL*: bases de datos relacionales, software libre. El sistema libre más utilizado del mundo y el que utilizaremos en este libro. Es libre cuando se utiliza en proyectos de software libre, de lo contrario habría que pedir permiso o comprar una licencia especial.
 - *Oracle Database*: sistema de pago ampliamente utilizado en el mundo empresarial. Fue pionero en su época. Dispone de versiones gratuitas muy limitadas en funcionalidad.
 - *SQL Server*: sistema de pago creado por Microsoft. Hace competencia a Oracle.
-



Actividades

1. Busque los tres principales sistemas gestores de bases de datos relacionales que son software libre.
 2. En su opinión, ¿cuáles son las tres principales ventajas de un SGBD con respecto a un sistema de ficheros de datos?
-

3. Componentes de un SGBD. Estructura

Un SGBD consiste en una serie de programas informáticos de alta complejidad y sofisticación. Cada SGBD contará con un número de subprogramas distinto para cumplir con necesidades específicas, aunque todos tienen un denominador común y es que poseen un sistema que gestiona el almacenamiento físico de los datos; otro que comunica las consultas realizadas por los usuarios de la base de datos con el gestor de almacenamiento; y un diccionario de datos que permite a los usuarios saber qué tablas hay, las cláusulas de integridad declaradas, qué columnas tiene cada tabla, etc.



Recuerde

Con esos tres sistemas el programador podrá explotar una base de datos de forma adecuada.

3.1. Gestión de almacenamiento

Los datos que maneja un SGBD son almacenados físicamente en el soporte físico que utilice (disco duro, unidad externa, etc.). El administrador de bases de datos define el modelo de datos y es el SGBD el que se ocupa de guardar esos datos físicamente. Para ello, cuenta con un componente informático de alta complejidad que se encarga de ello.

Las principales funciones de un gestor de almacenamiento son estas:

- Convertir consultas del usuario en operaciones lógicas en el sistema de archivos físico.
- Controlar el búfer de la memoria principal, es decir, gestionar la memoria RAM del sistema para mejorar el rendimiento.
- Asegurarse de que las restricciones de integridad se cumplen.
- Sincronizar acciones simultáneas de varios usuarios.
- Controlar los sistemas de copia de seguridad y recuperación.



Actividades

-
3. Busque información sobre los distintos motores de almacenamiento de MySQL. Sobre todo *MyISAM* e *InnoDB*.
-

3.2. Gestión de consultas

El usuario se comunica con la base de datos mediante consultas, generalmente consultas de SQL, el estándar.

El SGBD se encargará de leer esas consultas, compilarlas y comunicarse con el motor de almacenamiento para manipular los datos deseados.

Las consultas son convertidas a código ejecutable por el ordenador y, además, son optimizadas para que sean lo más rápidas posible.

La gestión de consultas sigue estos pasos:

1. El usuario envía una consulta al servidor.
2. El servidor comprueba la sintaxis básica de la consulta para comprobar que sea correcta. En caso de no serlo, devuelve un error.
3. Despues de cerciorarse de que la sintaxis es correcta, se hacen comprobaciones generales que no requieran de mucho esfuerzo para la base de datos, como ver que las columnas y tablas especificadas existen, que los tipos de datos son correctos, etc. En caso de haber un problema, se devuelve un error.
4. Con la sintaxis correcta y los datos coherentes, se pasa el proceso al **optimizador de consultas**, que es una pieza de software altamente sofisticada que busca la manera más rápida de satisfacer esa consulta. Cada SGBD tiene un optimizador diferente y lo que uno puede hacer muy rápido, es posible que otro lo haga más lento.
5. El SGBD, con el método más óptimo ya escogido, va al sistema de almacenamiento y ejecuta la consulta.
6. Con la consulta ejecutada, se devuelven los datos al usuario o un error, en caso de que haya un problema.

Es posible que entre el paso 1 y el paso 2 se interponga una **caché**, que es un sistema que guarda los resultados de las últimas consultas realizadas para que se ejecuten más rápido. Cada SGBD tiene sistemas de caché diferentes, pero todos tienen un propósito común: evitar ejecutar la consulta en sí y retornar los resultados directamente.

Cada SGBD soporta unas consultas diferentes. Todas suelen utilizar el SQL, aunque con sus peculiaridades, por eso cada SGBD tiene un motor de consultas diferente.



Recuerde

Las consultas son convertidas a código ejecutable por el ordenador y, además, son

optimizadas para que sean lo más rápidas posible.



Actividades

4. ¿Quién actúa antes, el optimizador de consultas o el gestor de almacenamiento?
 5. Todos los gestores de almacenamiento funcionan igual, ¿lo único que cambia es la manera en que guardan los datos en el disco duro? Razone la respuesta.
-

3.3. Motor de reglas

El SGBD debe proporcionar un sistema que permita explorar las bases de datos y sus características. Un SGBD se compone de bases de datos que, a su vez, tienen tablas con columnas. Si el SGBD no contara con un sistema para explorar la lista de elementos, solo la persona que creó las bases de datos conocería sus características.

Para eso existe el motor de reglas o diccionario de datos, que es una interfaz entre el SGBD y el programador con la lista de propiedades de cada base de datos. De esta manera cualquier usuario puede saber qué tareas puede llevar a cabo con la base de datos.



Importante

El motor de consultas se conectará a este diccionario para evaluar que las consultas sean correctas.

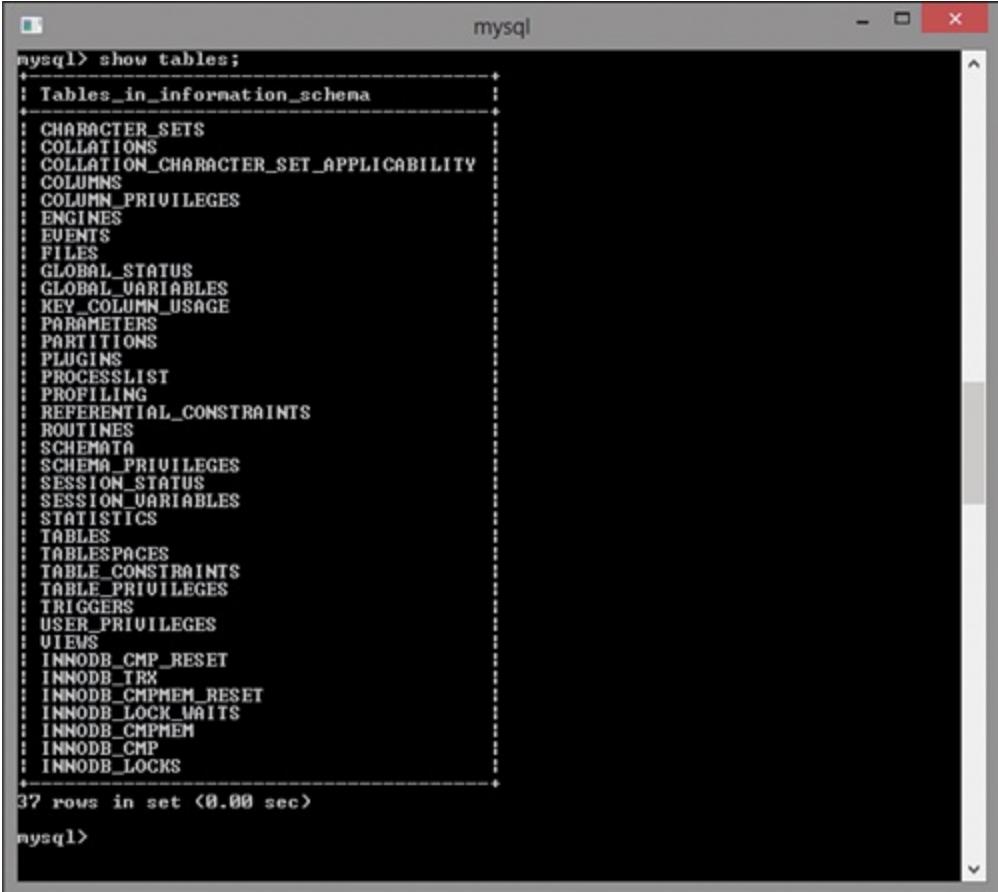
Un diccionario de datos debe proporcionar la siguiente información:

- Lista de bases de datos del sistema.
- Para cada base de datos, la lista de tablas que la contienen y sus propiedades.
- Para cada tabla, la lista de columnas con sus propiedades.
- Lista de restricciones de integridad y relaciones que hay entre las tablas.
- Listado de índices de cada tabla.
- Lista de usuarios y sus permisos.

- Listado de procedimientos almacenados, *triggers* y eventos.
- Cada SGBD tendrá información extra particular, como pueden ser estadísticas, registros de acceso, etc. Depende de lo que cada SGBD implemente.

La manera de acceder a esta información es mediante consultas de SQL tradicionales, por lo que el diccionario de datos es tratado de forma parecida a una base de datos convencional. Normalmente, el SGBD crea una base de datos especial con esta información. Cada tipo de dato está guardado en una tabla o vista.

Es tarea del usuario de la base de datos conocer la manera de acceder a estos datos en el SGBD que esté utilizando. Las diferencias entre uno y otro son grandes y apenas comparten elementos en común.



```
mysql> show tables;
+-----+
| Tables_in_information_schema |
+-----+
| CHARACTER_SETS                |
| COLLATIONS                   |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                      |
| COLUMN_PRIVILEGES             |
| ENGINES                      |
| EVENTS                       |
| FILES                        |
| GLOBAL_STATUS                |
| GLOBAL_VARIABLES              |
| KEY_COLUMN_USAGE              |
| PARAMETERS                    |
| PARTITIONS                   |
| PLUGINS                      |
| PROCESSLIST                  |
| PROFILING                    |
| REFERENTIAL_CONSTRAINTS      |
| ROUTINES                     |
| SCHEMATA                     |
| SCHEMA_PRIVILEGES            |
| SESSION_STATUS                |
| SESSION_VARIABLES             |
| STATISTICS                   |
| TABLES                       |
| TABLESPACES                  |
| TABLE_CONSTRAINTS            |
| TABLE_PRIVILEGES              |
| TRIGGERS                     |
| USER_PRIVILEGES               |
| VIEWS                        |
| INNODB_CMP_RESET              |
| INNODB_TRX                   |
| INNODB_CMPMEM_RESET           |
| INNODB_LOCK_WAITS             |
| INNODB_CMPMEM                |
| INNODB_CMP                   |
| INNODB_LOCKS                 |
+-----+
37 rows in set (0.00 sec)

mysql>
```

Ejemplo de diccionario de datos de MySQL



Ejemplo

En MySQL se accede al diccionario de datos mediante una base de datos especial llamada *information_schema*. Ahí se encuentran alrededor de 37 tablas con distinta información relativa a la base de datos.

4. Terminología de SGDB

A continuación, se muestra una lista de palabras comunes aplicables a los SGBD y su significado:

- **Transacción:** es un conjunto de consultas que se envían a la base de datos y que se ejecutan en conjunto. Si una falla, ninguna se ejecuta.
- **SQL (*Standard Query Language*):** lenguaje para ejecutar consultas en la base de datos.
- **Diccionario de datos:** sección accesible de una base de datos en la que se encuentra la definición del modelo de datos: lista de tablas, lista de columnas, usuarios del sistema y sus permisos, lista de restricciones, etc.
- **DBA (*Data Base Administrator*):** administrador de bases de datos, es la persona que se encarga de instalar y mantener la base de datos.
- **ACID (*Atomicity, Consistency, Isolation, Durability*):** siglas en inglés de atomicidad, consistencia, aislamiento y durabilidad. Son las características básicas de un sistema de transacciones.
- **Cascada:** cuando cambios en una clave primaria afectan a las claves foráneas que la referencian en otras tablas.
- **Servidor:** el sistema informático en el que se encuentra instalado el SGBD. Puede ser un solo ordenador o varios, si el sistema es distribuido.
- **Cliente:** la aplicación informática que se conecta a la base de datos y la manipula.
- **Concurrencia:** la habilidad que tiene un SGBD para gestionar a varios usuarios que se conectan simultáneamente a la base de datos y realizan operaciones sobre los mismos objetos.
- **Conexión:** una sesión de un usuario conectado al SGBD. Una conexión solo puede realizar una consulta en un momento determinado, aunque un mismo usuario se puede conectar al mismo servidor de forma simultánea varias veces.
- **Consistencia:** cumplimiento de las restricciones de integridad. Tener claro que la copia de los datos es coherente. El SGBD debe encargarse de que siempre haya consistencia y de que en caso de no haberla, sea por culpa de un uso

malintencionado del usuario y no porque el SGBD falló.

- **DDL (Database Definition Language):** conjunto de consultas e instrucciones que permiten al DBA crear una base datos con sus tablas, columnas y restricciones.
- **DML (Database Manipulation Language):** conjunto de consultas e instrucciones que permiten al DBA y a los usuarios de la base de datos manipular una base de datos.
- **DCL (Database Control Language):** conjunto de consultas e instrucciones que permiten al DBA gestionar los usuarios y los permisos que estos tienen para acceder a cada parte de la base de datos.
- **Lock (bloqueo):** cuando un usuario accede a los datos y bloquea el acceso a los mismos a los demás usuarios.
- **Índice:** un índice es una manera de acceder de forma más rápida a los datos, creando para ello una especie de diccionario mediante el cual es posible averiguar en qué lugar físico se encuentra cada dato rápidamente sin tener que rastrear una tabla completa.
- **Consulta:** una sentencia que se envía al SGBD especificando qué se quiere hacer.
- **Escalabilidad:** la virtud que tiene un SGBD de permitir ampliar sus capacidades, tanto de tamaño como de acceso a medida que vaya creciendo. La mayoría de bases de datos funcionan bien cuando 5 usuarios las utilizan simultáneamente. Pocas funcionan bien cuando 100 usuarios la utilizan.



Actividades

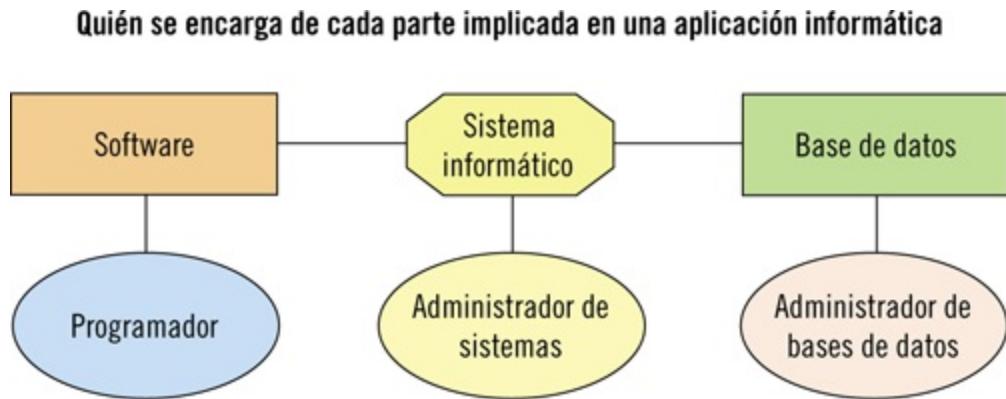
6. ¿Qué diferencia hay entre la escalabilidad y la concurrencia?

5. Administración de un SGDB

Los SGBD proporcionan herramientas de alto nivel para gestionar un modelo de datos. Es necesario contar con una o varias personas que conozcan a fondo el SGBD en cuestión y se encarguen de administrarlos correctamente, para eso están los DBA (*Data Base Administrators*, por sus siglas en inglés, administradores de bases de datos).

Comúnmente, en aplicaciones sencillas con pocos requisitos, el mismo programador

será el que se encargue de gestionar la base de datos. En organizaciones más grandes es común contar con uno o varios DBA que se encarguen de que la base de datos se mantenga activa las 24 horas del día, de optimizarla y de que los clientes del servidor de bases de datos la utilicen de forma correcta.



5.1. El papel del DBA

Un DBA (administrador de bases datos) es el responsable máximo de que la base de datos se comporte de forma eficiente, de que no se pierdan datos y de que los usuarios utilicen la base de datos de forma correcta.

Es un puesto de gran responsabilidad si la base de datos contiene datos críticos o requiere de altos requisitos de disponibilidad. Es, además el único usuario que posee todos los permisos disponibles para gestionar la base de datos sin restricciones, por lo que se encargará de crear a los demás usuarios.

Piense en el DBA como en el usuario que tiene acceso *root* al sistema. Puede haber varios DBA, aunque hay que mantener el número lo más bajo posible, porque es un puesto de alta responsabilidad y lo ideal es que lo gestione el menor número de personas posible.

La lista de habilidades con las que un DBA debe contar es esta:

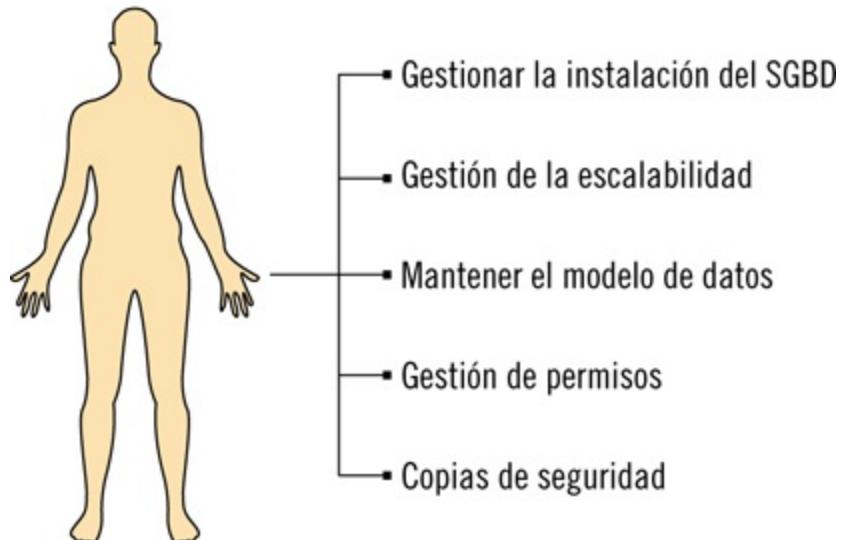
- Amplio conocimiento de teoría de bases de datos: reglas de Codd, modelo relacional, etc.
- Amplio conocimiento de diseño de base de datos: creación de modelos entidad/relación, etc.
- Experiencia trabajando con el SGBD en cuestión: si se trabaja con Microsoft SQL Server, debe tener experiencia de haber trabajado con ese sistema.

- Conocimiento completo del SQL, el lenguaje de consultas para comunicarse con la base de datos: cada SGBD cuenta con sus particularidades dentro del SQL, el DBA debe conocer el *dialecto* de SQL que utilice el SGBD que vaya a administrar.
- Buenas habilidades comunicativas: el DBA debe proporcionar documentación a los programadores además de explicarles la forma óptima de acceder a la base de datos.
- Conocimiento general sobre computación distribuida, de sistemas operativos y de sistemas de almacenamientos de archivos.

Entre sus obligaciones destacan las siguientes:

- Instalar y actualizar el servidor de bases de datos cuando sea necesario, escogiendo la versión óptima para cada necesidad.
- Planificar las necesidades de almacenamiento futuras, asignando el espacio suficiente para que la aplicación funcione en el futuro.
- Modificar la estructura de la base de datos si los programadores requieren cambios en la misma. El DBA debe escoger la manera en la que, cumpliendo con los requisitos del programador se obtenga la solución más óptima y eficiente para la base de datos.
- Gestión de permisos de usuarios. Además de monitorizar el uso que hacen los usuarios de la base de datos para detectar posibles anomalías y comunicárselas a los programadores.
- Monitorizar la base de datos y optimizarla. Si el DBA detecta que algunos accesos son ineficientes, deberá buscar la manera de que sean eficientes.
- Planificación de copias de seguridad y de recuperación de las mismas en caso de fallo en el sistema.
- Gestión de datos archivados. El DBA puede requerir archivar algunos datos que no van a ser utilizados por la aplicación pero que hay que guardar en caso de que hagan falta en el futuro para generar informes, por ejemplo.

Tareas principales del DBA



Actividades

7. ¿Un DBA es un puesto de trabajo que puede abarcar desde una preocupación más para un programador normal hasta un puesto a jornada completa y alta responsabilidad en una empresa? Razone su respuesta.

5.2. Gestión de índices

¿Qué es un índice? Aplicado a bases de datos, un índice es un sistema informático que se encarga de *averiguar* dónde se encuentra un dato sin necesidad de recorrer todos los demás datos para encontrarlo. Piense en una biblioteca en la que los libros se encuentran ordenados por género y dentro de cada género, por fecha de publicación.

Si se quiere obtener un libro escrito en 2010 en el género de ciencia ficción simplemente se tendría que mirar el índice para ver en qué estantería se encuentra ese libro. Esto ahorraría mucho trabajo porque no habría que ir libro a libro viendo si es el que se busca.

Dentro del estante de libros escritos en 2010 de ciencia ficción se encontrarán varios libros, pero solamente habrá que recorrer unos pocos. No más de 100 seguramente.

Imagine que los libros se encontraran ordenados por género y título ordenado

alfabéticamente. En ese caso podría acudirse a la sección de ciencia ficción y realizar una búsqueda más exacta, se ahorraría aún más tiempo, porque no tendrían que revisarse varios libros, se iría directamente.

Aplicado a las bases de datos el concepto es el mismo: se crea un sistema que permita saber dónde van a estar los datos sin tener que ir uno a uno. Es el DBA el que se encarga de indexar una tabla.



Recuerde

Puede haber varios DBA, aunque hay que mantener el número lo más bajo posible, porque es un puesto de alta responsabilidad y lo ideal es que lo gestione el menor número de personas posible.

Un índice puede estar compuesto por una o varias columnas.

El número de posibles valores que un índice puede contener se llama **cardinalidad**. Esto determina el número de comprobaciones que hará falta realizar para encontrar un dato concreto. En el ejemplo anterior, cuando se indexaba por año, la cardinalidad era mucho más baja que cuando se ordenaba por nombre.

Es el DBA el que debe elegir la combinación óptima que permita los accesos más rápidos. La máxima al crear índices es que **aumente la velocidad al buscar datos** pero **disminuye significativamente la velocidad al modificarlos**, puesto que hay que regenerar el índice de tal manera que siga estando actualizado.

En el ejemplo de la biblioteca todos los libros se encuentran en estanterías. En un sistema informático esto es mucho más complejo y los datos ni siquiera necesitan estar ordenados físicamente en el disco duro.

Lo importante es contar con un sistema que permita determinar dónde se encuentra un dato sin necesidad de buscar uno a uno.

Diferentes SGBD proporcionan diferentes tipos de índices y se gestionan internamente de forma diferente.

Lo que es común a todos es que el índice más importante es **la clave primaria**. La clave primaria siempre se encuentra indexada. Como se ha apuntado, cada SGBD gestiona los índices de forma diferente, en algunos casos incluso los datos se ordenan

físicamente según el valor de la clave primaria de forma correlativa, por lo que acceder a un dato sabiendo su clave primaria es la manera más rápida posible de hacerlo.

Cabe destacar que es posible indexar por el valor completo de un campo o agrupar. Es decir, si hay un campo nombre, pueden indexarse solamente los primeros 3 caracteres. De esta manera se obtiene un acceso bastante rápido sin perjudicar demasiado la gestión del índice.

A continuación, se muestra un ejemplo de cómo un índice podría ayudar a buscar datos en una tabla:

Cliente	Empresa	Ciudad	Fecha alta
José	ACME	Ciudad Real	2013-05-20
Marcos	ACME	Málaga	2013-05-10
Luis	ACME	Valencia	2010-05-10
Margarita	Toys Harris Ltd.	Málaga	2012-12-20
María	Toys Harris Ltd.	Málaga	2010-01-08
José María	Toys Harris Ltd.	Madrid	2009-09-20

Suponiendo que el único índice se encuentra en {Cliente} (la clave primaria).

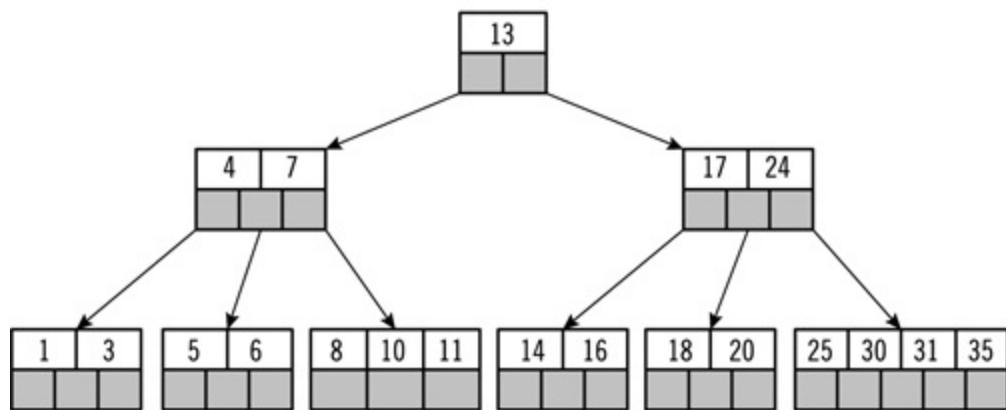
- Se quiere acceder a todos los clientes de la empresa ACME:
 - Se necesitaría hacer un escaneo completo de la tabla. Se escanearían 6 registros y solo se devolverían 3.
 - Si se añadiera un índice en {Empresa} podrían devolverse los 3 registros directamente.
- Se quiere acceder a todos los clientes de “Toys Harris Ltd.” que sean de Málaga.
 - Si se contara con el índice en {Empresa}, se podría ir a los tres registros que son de esa empresa y luego filtrar. Se escanearían 3 registros y se devolverían 2.
 - Si se creara un índice en {Empresa, Ciudad} se podría ir directamente a los

registros de nuestro interés.

Este es un ejemplo con una tabla diminuta, no se apreciaría rendimiento alguno al utilizar índices, pero con tablas de millones de registros o que ocupen muchos megabytes de almacenamiento en disco, la diferencia sería enorme.

El DBA debe conocer cómo va a ser utilizada su base de datos, decidir qué índices son necesarios y crearlos. Deberá, además, decidir la cardinalidad de los mismos para encontrar el equilibrio entre el rendimiento de consulta y el rendimiento de modificación.

Ejemplo de estructura de un índice B-Tree. Cuando se indexan varias columnas, se va creando un árbol con las posibles combinaciones



Actividades

8. ¿Qué efecto se busca con un índice?

5.3. Seguridad

El DBA debe encargarse de que la base de datos sea utilizada únicamente por los usuarios que él espera. Además, debe asegurarse de que esos usuarios realizan solamente las operaciones que se supone que deben realizar.



Ejemplo

No debe ser posible que un usuario ajeno a la empresa se conecte a la base de datos y se dedique a borrar tablas.

Entre las operaciones de seguridad que el DBA deberá manejar destacan las siguientes:

- Asegurarse de que solo los usuarios correctos se pueden conectar a la base de datos: el DBA creará los usuarios necesarios y solo ellos se podrán conectar. No serán aceptados usuarios anónimos.
- Asegurarse de que solo los clientes correctos se pueden conectar: para ello se podrá utilizar una gestión por IP para determinar si ese cliente está autorizado o no.
- Asegurarse de que los usuarios solo realizan las operaciones que pueden hacer: para eso dará o quitará permisos a determinados usuarios para acceder a unas u otras tablas.
- Evitar que *malware* acceda a la base de datos corrompiéndola o sustrayendo información privilegiada.
- Encargarse de que un aumento inesperado en el uso de la base de datos no haga que esta deje de estar disponible. Para ello podrá necesitar establecer límites en el número de conexiones por cada usuario o tener preparado una manera de escalar los recursos informáticos para satisfacer la demanda de trabajo.
- Estar preparado para que en caso de accidente o fallo informático, la base de datos sigue estando accesible al público.
- Comprobar que no es posible corromper los datos o crear inconsistencias en los mismos. Para ello, es posible que se necesiten establecer nuevas restricciones de integridad o eliminar permisos a determinados usuarios.

Además de estos puntos, es posible que sean necesarios otros métodos más complejos como la encriptación de datos sensibles o de mejorar los métodos de autenticación que por defecto proporciona el SGBD para hacer comprobaciones extra.

El terreno de la encriptación es bastante amplio, pero el DBA debe tener algunos conocimientos básicos del mismo. Por ejemplo, el DBA nunca debe permitir que la contraseña de un usuario se guarde como texto en una tabla, deberá estar encriptada.

Hay otros métodos más complicados para evitar que los datos sean visibles. Por ejemplo, es posible encriptar los datos de una tarjeta de crédito de tal manera que solo puedan verse si se sabe la contraseña del usuario que la posee (posible para una web de casa de apuestas online, por ejemplo).

El DBA deberá, con los requisitos de la aplicación en la mano, decidir los requisitos de encriptación y seguridad que sean necesarios.



Nota

No es lo mismo crear una base de datos para un blog de astronomía que una base de datos de un bróker online que permita comprar y vender acciones en tiempo real.



Actividades

9. Determine si la siguiente frase es verdadera o falsa y argumente su respuesta: “lo importante al empezar a implementar una base de datos es que funcione correctamente y que el modelo de datos esté correcto. La seguridad debe ser tenida en cuenta más adelante, cuando el sistema ya se esté empezando a utilizarse”.
-

5.4. Respaldos y replicación de bases de datos

Los respaldos y la replicación son conceptos **estrechamente ligados**. Los respaldos no son más que copias de seguridad de los datos. La replicación consiste en duplicar los datos de la base de datos en varios sistemas informáticos de tal manera que haya redundancia de datos, lo cual aumenta la seguridad —si algo se pierde, está en otro sitio también— y la velocidad —si un servidor está ocupado, puede actuar otro—.

Están estrechamente ligados porque una manera muy segura de hacer copias de seguridad es tener un servidor de copia de seguridad que esté replicado con el servidor de bases de datos principal.

La replicación solo es necesaria cuando los accesos a la base de datos sean masivos o se prevea que lo sean a corto plazo. Las copias de seguridad, en cambio, deben estar presentes desde el minuto uno.

Respaldos

El DBA debe encargarse de que haya copias de seguridad más o menos actualizadas en caso de fallo en el sistema. Además, tendrá que conocer el protocolo a la hora de recuperar esas copias de seguridad para garantizar que es posible utilizarlas.

Dentro del mundo de las bases de datos existen varios tipos de copias de seguridad:

- **Copias exactas:** se realiza una copia desde cero de toda la base de datos. Es la más sencilla de hacer y recuperar en caso de que el volumen de datos sea pequeño.
- **Copias incrementales:** partiendo de una copia de seguridad, se le añaden solamente los cambios producidos desde la última copia. Así se ahorra espacio en disco por no necesitar de dos copias, aunque añade más trabajo a la hora de restaurar. A medida que se hacen más copias incrementales, se van añadiendo los cambios encima. De esta manera puede restaurarse hasta el punto que se quiera.
- **Copias diferenciales:** partiendo de una copia de seguridad exacta, se le añaden solamente los cambios producidos desde la última copia. La diferencia con las incrementales es que no se pueden añadir nuevos cambios a la misma copia de seguridad.
- **Copia en espejo continua:** cada cambio en la base de datos se va replicando automáticamente en otro servidor que actúa como un espejo. En caso de fallo, simplemente tendría que usarse el servidor espejo temporalmente mientras se arregla al servidor principal.

El DBA deberá decidir qué tipo de copia de seguridad es el óptimo en cada momento.

Dentro de las copias de seguridad completas hay varios formatos:

- **SQL:** la copia se guarda en formato SQL. Simplemente lanzando esas consultas en la base de datos puede crearse todo desde cero.
- **Texto:** cada tabla se guarda en un fichero de texto separado. El rendimiento es máximo, pero es engorroso de recuperar y mantener.
- **Binarias:** las tablas se guardan tal cual se almacenan físicamente en el disco duro. El rendimiento es total, pero es fácil que queden corruptas. Si se actualiza el SGBD pueden dejar de funcionar, no son muy portables, etc.
- **Otros formatos:** a veces lo que se pretende es exportar la base de datos para utilizarla en otro sistema o por si se quiere realizar una migración. Por ejemplo, pasar de una base de datos en *MySQL* a una en *Oracle*.

Siempre hay que recordar que es tan importante realizar una copia de seguridad como saber que luego se podrá utilizar para lo que se desea.



Recuerde

Los tipos de copias de seguridad son: exactas, incrementales, diferenciales y en espejo continua. Las copias completas pueden presentarse en distintos formatos como SQL, texto, binarias y otros formatos.



Aplicación práctica

Se encuentra administrando la base de datos de un instituto. Actualmente no se realizan copias de seguridad de ningún tipo. Solamente se hacen de vez en cuando manualmente.

La base de datos ocupa unos 10 MB en total. Solamente se modifica en horario lectivo, durante la noche el servidor se encuentra prácticamente inactivo.

La copia de seguridad se restauraría solamente en caso de desastre informático. No suele ser habitual que se utilicen.

Escoja y explique el método de copia de seguridad más apropiado para este entorno.

SOLUCIÓN

Una copia exacta sería la opción correcta. Se programaría para que se ejecutara a media-noche, por ejemplo. Es fácil de hacer, es portable y no daría problemas. Lo máximo de que se podría perder serían 24 horas de datos.

Esta copia podría almacenarse en la nube o en algún servidor externo a fin de evitar que se pierda en caso de accidente.

Replicación

Cuando una base de datos tiene altísimos requisitos de almacenamiento o de acceso rápido a la misma, aparecen dos opciones:

- a. Ampliar el ordenador donde se encuentra: añadirle más RAM, más disco duro, mejorar la conexión de red, etc. A esto se le llama escalado vertical porque va desde abajo arriba. El ordenador amplía su hardware para mejorar su rendimiento.
- b. Añadir nuevos ordenadores iguales y dividir el trabajo entre ellos. Este es el escalado horizontal.

Podrá imaginar que la opción a) tiene multitud de limitaciones, además de que el número de ordenadores que pueden conectarse es infinito, ampliar un ordenador sí es algo finito. Por ejemplo, el supercomputador más potente actualmente es el Tianhe-2, creado por la Universidad Nacional de Tecnología de Defensa en China y vale millones de euros. Sin embargo, Google utiliza un sistema distribuido de ordenadores baratos que en total tiene más potencia, vale menos y es mucho más fácil de mantener.

Siempre hay un punto en el cual deja de compensar ampliar un ordenador y empieza a compensar utilizar una red distribuida. Hay que tener en cuenta que ampliar un ordenador es mucho más sencillo en los primeros pasos. Por ejemplo, si la base de datos es muy grande y el disco duro es de solo 250 GB, se añade otro de 1 TB y es posible seguir utilizándose durante varios años más.



Nota

La ventaja además es que no hay que realizar apenas tareas de administración del sistema, porque todo sigue igual que antes pero con más capacidad.

Donde es más difícil escalar verticalmente es cuando lo que se convierte en un cuello de botella son los accesos a la base de datos. Cuando el almacenamiento es lo que escasea, aumentar el disco es sencillo; cuando el problema es que algunas consultas son lentas, es fácil determinar qué pasa y arreglarlo, pero cuando lo que se necesita es que, por ejemplo, 20.000 usuarios puedan hacer apuestas en directo por un partido de fútbol, la cosa cambia.

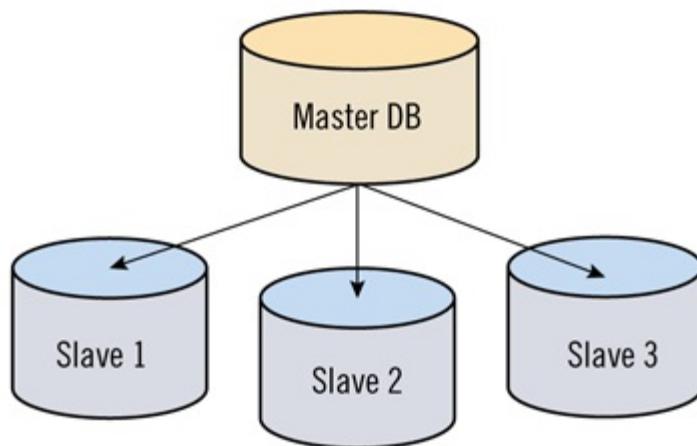
Casos que resuelve la replicación:

- **Distribución de datos:** es útil para mantener los datos en otra localización geográfica.

Ejemplo: tener dos copias una en Estados Unidos y otra en Singapur para reducir los tiempos de latencia en la red, aumentando con ello el rendimiento.

- **Balanceo de carga:** es útil para dividir el trabajo entre varios servidores. Un cliente envía una consulta al servidor principal, esta pasa por el balanceador de carga y le envía la consulta al servidor que se encuentre menos ocupado en ese momento.
- **Copias de seguridad:** como se vio anteriormente, es posible crear una copia de seguridad instantánea en otro servidor para prevenir posibles fallos.
- **Alta disponibilidad y recuperación de fallos:** si uno de los servidores falla, no pasa nada porque se pueden utilizar los demás. Simplemente, habría una pequeña bajada de rendimiento durante el tiempo en que los administradores de sistemas arreglan el servidor roto.

Al replicar los datos, múltiples servidores pueden satisfacer una petición, por lo que se balancea la carga y aumenta la fiabilidad



Un sistema de replicación funciona así:

- a. El servidor maestro recibe una modificación y la guarda en un fichero de *log*.
- b. El servidor esclavo copia el fichero de *log* del maestro.
- c. El servidor esclavo ejecuta los cambios que se hicieron en el maestro.

Dependiendo de las necesidades, pueden encontrarse varios problemas o casos de uso. Por ejemplo, ¿qué pasa si se reciben tantas modificaciones que el servidor maestro se colapsa? En ese caso se necesitaría un sistema multi-maestro en el que varios servidores puedan recibir modificaciones. Ese es un sistema realmente complejo y difícil de manejar, aunque raramente es necesario.

¿Qué pasa en el tiempo que transcurre entre que el maestro recibe la modificación y

todos los clientes la aplican? Dependiendo de las necesidades, ese lapso de tiempo puede ir desde unos minutos a pocos milisegundos.

Lo más importante es que el sistema de replicación mantenga siempre una copia consistente de los datos, que tanto en el maestro como en todos los esclavos haya la misma información.

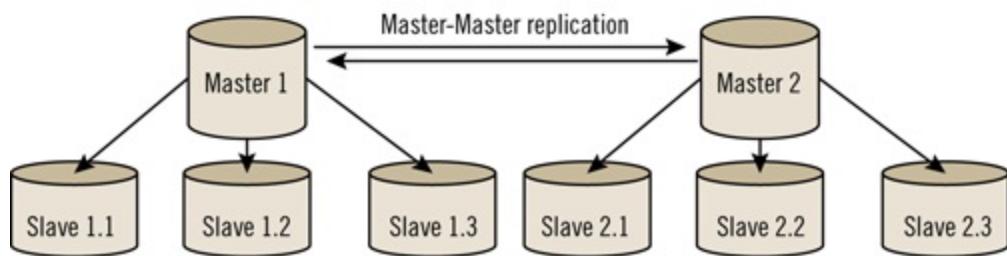


Importante

Puede haber un pequeño retraso, pero la consistencia debe existir siempre.

Este sistema aumenta mucho el rendimiento al consultar la base de datos, porque varios esclavos tienen una copia de la base de datos, por lo que cualquiera de ellos puede responder a esa consulta.

Esquema de replicación que incluye maestro-esclavo y maestro-maestro



El DBA deberá determinar si un sistema de replicación es necesario o no y de implementarlo en caso afirmativo. La replicación será totalmente transparente a los usuarios de la aplicación y a los programadores de la misma.



Importante

Será imposible saber qué esclavo ejecutó la operación.

Además, el DBA deberá medir el *delay* (retraso de replicación para saber exactamente cuánto puede tardarse en tener todos los esclavos replicados. Este es un dato muy importante para la empresa y determinará en gran medida las limitaciones de la aplicación.

6. Gestión de transacciones en un SGBD

Uno de los principales objetivos de un SGBD es la fiabilidad, estar seguros de que cada dato es guardado donde debe, que no hay inconsistencia y que no se corrompen los datos.

Una de las herramientas que más ayudan a evitar la corrupción de datos son las transacciones, un sistema que agrupa operaciones de tal forma que se asegura que se cumplen en su totalidad.

Piense qué pasaría en el siguiente caso: un bróker online en el cual es posible comprar y vender acciones. Al vender una acción, el proceso comienza vendiendo el valor en el mercado, a continuación se abona esa cantidad en la cuenta corriente del vendedor. ¿Qué pasa si entre que se vende el valor y se guarda la cantidad se va la corriente eléctrica? Se perdería todo ese dinero en el limbo y requeriría mucho trabajo manual saber que hay una cantidad que se le debe al cliente.

Para esto se inventaron las transacciones, para asegurarse de que las operaciones solo se ejecutan si son completas.

6.1. Definición de transacción

¿Qué es una transacción? Una transacción es un conjunto de operaciones dependientes las unas de las otras que se ejecutan en una base de datos. Para que una transacción se ejecute es necesario que todas las operaciones que la componen se ejecuten satisfactoriamente. Si hay un fallo en mitad del proceso, la base de datos no guardará ningún cambio y será como si no hubiera pasado nada. No se deja nada a medio hacer.

Es el programador de la aplicación el que se encarga de agrupar las consultas en transacciones según las necesidades de la misma. El DBA puede asesorar, pero no es su responsabilidad.

El programador se encarga de realizar una serie de consultas y, a continuación, hará *Commit* (confirmar cambios y es ahí cuando los cambios se aplicarán físicamente en la base de datos. El programador podrá también hacer *Rollback* (restablecer cambios para dejar todo como estaba y descartar la transacción en caso de que algo falle o alguna condición no se cumpla.

La mayoría de SGDB cuenta con un sistema de transacciones integrado que permite realizar todas estas operaciones mediante el SQL. Hay veces en las que las transacciones no se utilizan por defecto pero están permitidas, para ello hay que especificar cuándo se necesitan y cuándo no.

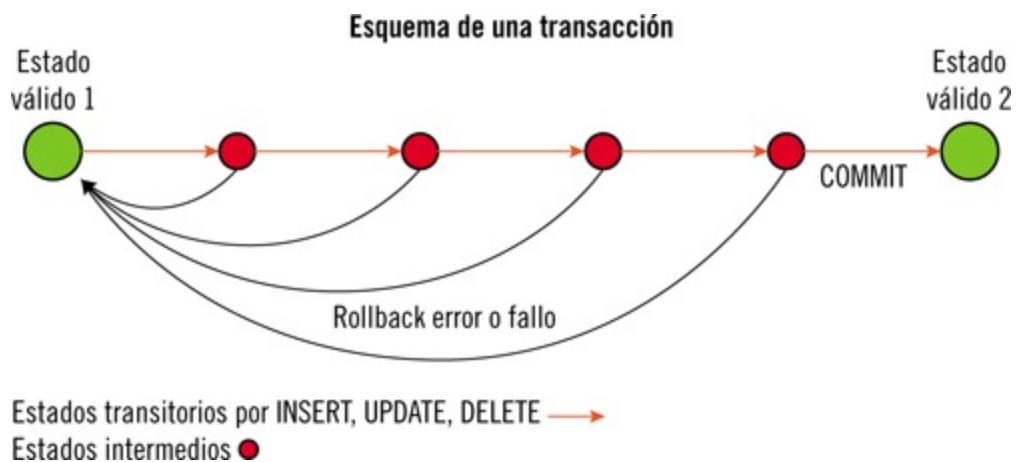


Ejemplo

En MySQL, la opción *Autocommit* está activada en la configuración por defecto. Cada vez que se haga una consulta de manipulación de datos se hará un *Commit* inmediatamente después, a no ser que se indique lo contrario.

Lo que se busca en un sistema transaccional es que cumpla con las cuatro reglas ACID (por sus siglas en inglés):

- **Atomicity (atomicidad):** cada transacción debe ejecutarse completamente o de lo contrario no ejecutarse en absoluto. Si una parte de la transacción falla, todo falla y los datos no se ven afectados.
- **Consistency (consistencia):** esta propiedad garantiza que cualquier transacción llevará a la base de datos desde un estado correcto hasta otro estado correcto, es decir, que la base de datos no contendrá errores tras aplicar una transacción. No permitirá violar restricciones de integridad.
- **Isolation (aislamiento):** esta propiedad garantiza que varias ejecuciones simultáneas de transacciones serán tratadas correctamente. Esta es una propiedad esencial de cara a la concurrencia de una base de datos.
- **Durability (durabilidad):** cuando una transacción se envía (se hace *Commit*), esos datos se guardarán de forma permanente en el soporte físico que utilice el SGBD. No importa que se vaya la corriente eléctrica o haya un fallo del sistema, los datos se deben guardar y si no se puede garantizar que todos han sido guardados, se dará por inválida la transacción y será ignorada.



Se pasa de un estado válido a otro estado válido. En cualquier momento, si hay un fallo o se hace ROLLBACK, se vuelve al estado inicial. Entre una operación y otra, se dice que la transacción se encuentra en “estado de transición”, que es cuando los cambios todavía no se han producido.



Actividades

10. ¿Qué es lo más importante de una transacción: la fiabilidad o la velocidad?

6.2. Componentes de un sistema de transacciones

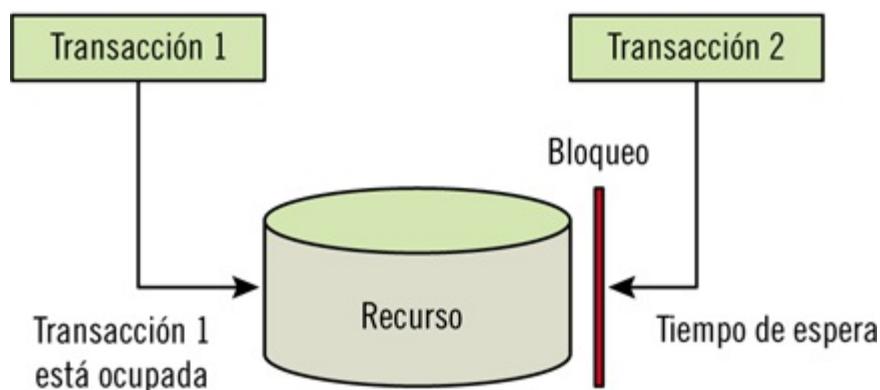
Un sistema de transacciones debe proporcionar un número de componentes que aseguren que el proceso funcione correctamente. Cada componente corresponde a una pieza de software altamente compleja con miles de líneas de código y que sería prácticamente imposible de implementar mediante un sistema de ficheros de datos simple:

- **Sistema de bloqueos:** dos transacciones pueden necesitar acceder a una misma porción de una base de datos simultáneamente. En este caso, la primera transacción debe bloquear esa porción para asegurarse de que no se pierden los cambios. La segunda transacción deberá esperar a que la primera termine para poder ejecutar los cambios. La gestión de bloqueos es realmente compleja y es común que una base de datos caiga debido a un problema en la gestión de los mismos, de tal manera que una tabla quede inutilizada temporalmente porque se han recibido muchos bloqueos.

Cada SGBD implementa unos tipos de bloqueos diferentes, unos más eficientes y otros menos. Como resumen puede identificarse entre:

- **Bloqueos a nivel de tabla:** cuando una transacción modifica una tabla, esta queda bloqueada hasta que la transacción termine. Las demás transacciones tendrán que esperar a que termine. Estos bloqueos son muy ineficientes en la gestión de la concurrencia, aunque bastante fáciles de implementar. Crea grandes cuellos de botella si una tabla recibe modificaciones muy frecuentes.
- **Bloqueos a nivel de fila:** cuando una transacción modifica una tabla, solo las filas afectadas por la modificación son bloqueadas. Si las demás transacciones van a hacer operaciones en la misma tabla pero que afectan a otros registros, estos cambios serán admitidos en tiempo real. Solo en el caso de que dos transacciones intenten modificar las mismas filas al mismo tiempo se producirá un cuello de botella.
Estos bloqueos son muy eficientes aunque son difíciles de implementar. Requieren de un software complejísimo para funcionar.

Esquema del bloqueo de un recurso



- **Sistema de rollback:** sistema que permite volver atrás en caso de que una transacción falle o se cancele. El sistema deberá crear una imagen del estado de la base de datos antes de que la transacción se produjera. En caso de fallo, esta debe ser utilizada automáticamente.
- **Sistema de rollforward:** una de las reglas de los SGBD era que en caso de error informático, nada debía ocurrir, que la copia de los datos debía permanecer siempre intacta. Este sistema lo que hace es guardar la lista de transacciones a medida que se van realizando. En caso de fallo, se recuperará una copia de seguridad, pero esta copia solo contendrá los cambios efectuados en el momento en que se hizo. El sistema de rollforward guarda todas las transacciones realizadas, por lo que será posible que, partiendo del estado inicial de la copia de seguridad, se ejecuten todas las transacciones realizadas

hasta el momento del fallo informático. De esta manera no se perderán datos.

6.3. Tipos de protocolos de control de la concurrencia

Hay varias maneras de gestionar un sistema concurrente, según las necesidades. Cada SGBD concurrente suele contar con uno o varios de estos sistemas:

- **Sistema de bloqueos:** cuando una transacción modifica una porción de los datos, estos quedan bloqueados. Si otra transacción modifica esa misma porción de datos, tendrá que esperar a que la primera los libere. Hay posibles cuellos de botella si muchas transacciones tratan de modificar los mismos datos durante mucho tiempo. Es posible implementar sistemas que eviten bloqueos mediante otros sistemas de concurrencia, aunque todos deben implementar una manera de detectar cuándo se puede leer o modificar un objeto.
- **Sistema de *timestamps* (marcas de tiempo):** a cada transacción se le asigna un identificador único cuyo valor representa el momento en el tiempo en que la transacción se ejecuta. Puede ser la hora del sistema o un identificador auto numérico proporcionado por el SGBD. Lo importante es que un valor superior corresponde a un momento de tiempo que ocurre después de un valor inferior. Cada objeto en la base de datos tiene un *timestamp* de lectura y otro de escritura.

En caso de leer un objeto:

- Si una transacción empezó antes del *timestamp* de escritura del objeto, significa que algo ha cambiado desde que se inició la transacción, por lo que la transacción debe ser cancelada o reiniciada.
- Si una transacción empezó después del *timestamp* de escritura, todo está correcto por lo que se puede leer el objeto y estará actualizado. Se actualizará el *timestamp* de lectura del objeto con el valor del *timestamp* de la transacción.

En caso de escribir un objeto:

- Si la transacción empezó antes del *timestamp* de lectura del objeto, significa que otra transacción ha leído esos datos. No se puede permitir que se guarde nada porque entonces la otra transacción tendrá una copia inválida de los datos. La transacción debe ser reiniciada.
- Si la transacción empezó antes del *timestamp* de escritura del objeto significa que algo cambió en el objeto desde que la transacción empezó. En

- ese caso, la transacción no ejecuta la modificación y continúa como si nada.
- Si no ocurre ninguno de los casos anteriores, el objeto se modifica y se le asigna a su *timestamp* de escritura el valor del *timestamp* de la transacción.

Este caso está pensado para evitar bloqueos, aunque añade complejidad e invalida muchas transacciones. En realidad hay un bloqueo pero es prácticamente instantáneo, que es el tiempo que transcurre mientras el *timestamp* de un objeto está siendo modificado.

- **Gráfico de precedencia:** aquí se evitan bloqueos mediante un sistema de precedencia. Cada transacción se considera un nodo. Cada nodo tiene un orden de precedencia. El nodo 1 necesita esperar que el nodo 2 termine de escribir para poder leer.

De esa manera, cuando múltiples transacciones acceden a un objeto, se va creando un orden que se deberá seguir para que cada transacción se ejecute correctamente y con la copia de los datos correcta.

Este es un sistema muy complejo desde el punto de vista informático. Se genera un orden en las operaciones y es importante que el sistema que controle la concurrencia se asegure de que no haya bucles infinitos.

Hay más métodos de control de la concurrencia pero se han visto los tres más comunes. El método más común es el de los bloqueos.



Nota

Normalmente, la parte de software del SGBD que maneja la concurrencia es una de las más complejas y la que marca la diferencia entre un sistema u otro.



Actividades

11. Investigue el método que implementa *MySQL* para gestionar la concurrencia (en particular con su motor de datos llamado InnoDB).
-

6.4. Recuperación de transacciones

Si una transacción falla, es importante que haya un mecanismo que permita devolver

a la base de datos al estado en que se encontraba antes. Este es un concepto muy importante porque una transacción siempre debe pasar desde un estado válido de la base de datos a otro estado válido. Si hubo operaciones erróneas, es posible que la base de datos quedara en un estado no válido.

Cuando una transacción es reiniciada (*rollback*) las demás transacciones concurrentes deben ser reiniciadas también.



Importante

Una transacción fallida puede hacer que otras transacciones fallen también.



Aplicación práctica

Le llega una incidencia que dice que la aplicación informática de la empresa va muy lenta.

Está confirmado que no es un problema de la aplicación, sino un cuello de botella producido por la base de datos.

La aplicación es consultada por muchos usuarios, pero las modificaciones de datos son poco usuales y van rápidas. Lo que va lento es la consulta de datos.

¿Cuáles serían las 5 primeras cosas que comprobaría para tratar de identificar y solucionar el problema?

SOLUCIÓN

Ver que las tablas se encuentran normalizadas y que no hay ningún fallo de diseño en el modelo de datos.

Ver si la utilización de un índice mejoraría la velocidad de las consultas.

Comprobar si el sistema informático en el que el SGBD se encuentra no está saturado por otras aplicaciones.

Ver si el SGBD no se encuentra optimizado: no está utilizando toda la memoria

que podría, etc.

Ver sí sería posible ampliar las capacidades del sistema informático fácilmente: añadir más RAM, discos duros más rápidos, etc. En caso contrario, habría que pensar en implementar un sistema de replicación.

7. Soluciones de SGBD

Soluciones de SGBD existen muchas, tantas como necesidades específicas se puedan tener. Los SGBD relacionales son los más extendidos, sin embargo, hay otras necesidades en las que otros tipos de SGBD pueden ser requeridos, como cuando es necesario almacenar componentes complejos como objetos en 3D, trabajo con datos matemáticos o relaciones entre componentes informáticos en vez de con datos. También es posible que todo un sistema relacional sea demasiado complejo para resolver un problema concreto y se usen otras herramientas más sencillas y, posiblemente, más eficientes.

En los siguientes apartados se estudiarán algunas peculiaridades de estos SGBD no tan usuales y para qué pueden ser utilizados.

7.1. Distribuidas

En este caso, es posible que el SGBD relacional además tenga opciones para distribuir los datos entre diversos servidores.

En este tipo de sistema lo que se busca es la capacidad de escalar horizontalmente el sistema de tal manera que la misma copia de la base de datos se guarde en varios sistemas informáticos separados.

Los grandes SGBD relacionales del mercado (*Oracle*, *MySQL*, *Microsoft SQL Server*) ya proporcionan métodos para distribuir una base de datos entre varios servidores esclavos.

7.2. Orientadas a objetos

Hay veces en las que lo que se quiere guardar va más allá de datos estructurados simples. No se precisa una tabla de usuarios con su nombre, e-mail y contraseña, sino que hace falta guardar una colección de modelos en tres dimensiones que se relacionen entre ellos y que tengan un comportamiento u otro dependiendo de cada registro.

En ese caso habrá que utilizar un sistema de base de datos orientado a objetos. Ya no habrá tablas y registros, sino clases y objetos.

Cada entidad formará una clase y cada registro tendrá unas propiedades y unos métodos (acciones). Cada clase puede heredar propiedades y métodos de una clase superior.

Las bases de datos orientadas a objetos son productos complejos y que requieren mucho conocimiento. Suelen ser utilizadas en grandes aplicaciones matemáticas, de modelación de objetos en 3D, etc.

Poseen numerosas ventajas cuando se realiza un trabajo intenso con objetos, pero presentan problemas en casos realmente menos complejos. Por lo general, un SGBD relacional es más rápido, más fiable y más fácil de usar.

7.3. Orientadas a datos estructurados (XML)

Es posible utilizar bases de datos basadas en el estándar XML. Esto es útil generalmente para bases de datos orientadas a documentos.



Ejemplo

Si se cuenta con una lista de patentes de Estados Unidos, es posible que simplemente con una base de datos basada en XML sea suficiente.

Están a medio camino entre un SGBD y un sistema de ficheros de datos. Añaden una capa de complejidad encima por tratarse del almacenamiento físico de los datos, de tal manera que el usuario solo debe trabajar con datos XML.

7.4. Almacenes de datos (*data warehouses*)

Son un tipo de base de datos especial que se utiliza solamente para guardar datos.

Estos datos tan solo se leerán ocasionalmente para realizar informes, para consultar datos anteriores o para tratar de recuperar información perdida mediante otro medio.

La particularidad de este sistema es que no permite ni modificación ni borrado de datos; solo guardado y lectura.



Nota

Este sistema está optimizado para que las inserciones de datos tengan un rendimiento máximo.

Se utilizan para guardar históricos de cambios, registros de operaciones realizadas, etc., que puedan ser necesarios en el futuro para realizar informes a largo plazo, por motivos de auditoría o para recuperar desastres informáticos.

Estas bases de datos pueden ser utilizadas en conjunto con otra base de datos relacional que guarde la información que cambia frecuentemente o encontrarse aisladas.

Se suelen utilizar para centralizar el guardado de históricos de cambios de varias aplicaciones de una misma empresa en un lugar común, para poder así realizar informes unificados.

Pueden tener tratamiento de relaciones y de inviolabilidad de restricciones, pero no están enfocadas hacia ello, su principal cometido es la gran escalabilidad en tamaño, velocidad de escritura y simplicidad.



Actividades

12. Busque un sistema SGBD orientado a objetos, otro en XML y un data *warehouse*.
-

8. Criterios para la selección de SGBD comerciales

La elección de un SGBD u otro constituye una de las decisiones empresariales más importantes que se pueden llevar a cabo en el apartado técnico de una empresa porque

limitará mucho las decisiones futuras.

En el siguiente capítulo se tratará en mayor profundidad el SGBD *MySQL*, por las razones siguientes:

- **Gratis para aplicaciones no comerciales:** la versión Community Edition es de uso libre y gratuito siempre y cuando se respete la licencia GPL.
- **Es código abierto:** Es posible acceder al código fuente del proyecto y realizar modificaciones.
- **Ampliamente utilizado:** es el SGBD código abierto más utilizado y está situado entre los más usados en general.
- **Estándar en programación web:** dado que el libro se centra en acceso a datos desde aplicaciones web y *MySQL* es lo más utilizado en ese ámbito, es quizás lo más apropiado.
- **Multiplataforma:** funciona en todos los sistemas operativos del mercado. Por ejemplo, si se tratara de *Microsoft SQL Server* solo podría usarse desde *Microsoft Windows*.

Este sistema gestor de bases de datos es utilizado por los grandes de Internet: Google, Facebook, Yahoo, Wikipedia y Youtube lo utilizan, por poner solo unos cuantos ejemplos.



Nota

No obstante, no siempre *MySQL* es la mejor opción para todo, a la hora de escoger un SGBD u otro hay que fijarse en características concretas que los diferencian.

Este libro se centra en SGBD relacionales, dado que son los más utilizados.

Lo habitual es elegir entre un sistema libre o un sistema de pago con soporte técnico en primer lugar. A partir de ahí, se escogerá el que tenga las características que se necesiten.

Normalmente, la determinación de usar uno u otro vendrá de la mano de estos factores:

- **Rendimiento:** que para el uso que le se le va a dar, la velocidad sea la máxima posible.

- **Escalabilidad:** relacionada con la anterior. Saber cómo de sencillo será escalar la base de datos de tal forma que admita más usuarios y más datos.
- **Interpretación del modelo relacional:** unos SGBD manejan unas restricciones de integridad que otros no. Por ejemplo, en Oracle es posible especificar que un valor numérico no sobrepase el número 20. En *MySQL* eso no es posible.
- **Interpretación del estándar SQL:** cada SGBD utiliza el modelo SQL y lo adapta para poder añadir nuevas funcionalidades. Quizá haya un conjunto de funcionalidades que un SGDB tiene y que se precisen.
Ejemplo: en Oracle puede utilizar el operador INTERSECT y en *MySQL* no, pero *MySQL* proporciona la utilidad INSERT ON DUPLICATE KEY UPDATE que Oracle no tiene.
- **Soporte técnico:** hay veces en que el soporte técnico de una gran empresa es lo que determina el uso de un sistema u otro. En un sistema propietario creado por Microsoft, IBM u Oracle es posible contar con un soporte técnico que se encargue de solucionar los problemas en persona. En un sistema código abierto como *MySQL* o *PostgreSQL* habrá que leer documentaciones online y arreglarlo nosotros mismos o contratar a un DBA experimentado (a menos que se haya contratado soporte técnico en el caso de *MySQL*).
- **Coste:** este puede ser también el factor decisivo. Una instalación de MS SQL Server varía de precio según el uso que se le vaya a dar, aunque nunca valdrá menos que unos cientos de dólares. En los sistemas de pago además, suelen cobrar por cada característica que se le quiera añadir. El paquete completo costaría miles de dólares al mes.



Nota

INTERSECT consiste en combinar dos consultas SQL de tal forma que solo se utilicen los registros que satisfacen ambas consultas.

INSERT ON DUPLICATE KEY UPDATE consiste en realizar una operación de inserción de datos y, en caso de existir ya ese registro, realizar una modificación. Es muy útil para contadores, por ejemplo. Puede organizarse en una tabla (fecha, visitas); la primera visita del día crearía un registro, las siguientes aumentarían en uno el contador.

A continuación, se listarán algunos de los SGBD relacionales más comunes, citando para qué suelen servir:

- **MySQL**: sistema libre y gratuito para uso no comercial. Ampliamente utilizado en programación web. Utilizado por millones de webs en todo el mundo.
- **PostgreSQL**: sistema libre y gratuito. Su principal cometido es ser una alternativa a *MySQL*. Por lo general, es más lento que *MySQL*, pero incorpora más capacidades de gestión de integridad y escalabilidad.
- **SQLite**: es libre también. Su principal característica es que se guarda en un archivo único y es portable de un lugar a otro. Es útil para aplicaciones informáticas de escritorio o en dispositivos móviles (es el estándar en Android).
- **Oracle Database**: uno de los sistemas más potentes aunque es de pago. Suele utilizarse en aplicaciones empresariales importantes. Su precio varía según el uso que se le vaya a dar y las características que se necesiten.
- **MS SQL Server**: producto de Microsoft que hace la competencia a *Oracle Database*. útil sobre todo en entornos en los que se suele trabajar con herramientas de Microsoft (*Windows*, *.NET Framework*, *Windows Server*, etc.).
- **IBM DB2**: producto de IBM para gestionar bases de datos relacionales. Su utilización es similar a la de *Oracle Database*.

Hay otros productos pero o bien son menos conocidos o ya no se utilizan mucho.

Puesto	SGBD	Modelo de datos	Puntuación
1.	Oracle	SGBD relacional	1583.84
2.	MySQL	SGBD relacional	1331.34
3.	Microsoft SQL Server	SGBD relacional	1207.00
4.	PostgreSQL	SGBD relacional	177.01
5.	DB2	SGBD relacional	175.83
6.	MongoDB	Catálogo de documentos	149.48
7.	Microsoft Access	SGBD relacional	142.49
8.	SQLite	SGBD relacional	77.88
9.	Sybase	SGBD relacional	73.66
10.	Teradata	SGBD relacional	54.41

Clasificación general de los SGBD más populares según el estudio realizado por <<http://db-engines.com>> mensualmente.



Aplicación práctica

Se encuentra en una reunión en la empresa. Se quiere decidir qué SGBD utilizar a partir de ahora, debido a que se va a informatizar gran parte de la empresa. Usted es el miembro de la reunión con mayores conocimientos informáticos y deberá decidir qué sistema utilizar.

La empresa es un almacén de piezas de electrodomésticos. Al día se procesan alrededor de 10.000 pedidos. Los requisitos que se han podido identificar son:

- Es fundamental que no se pierda ningún pedido, porque de lo contrario se perderá mucho dinero.
- La empresa cuenta con recursos económicos para afrontar estos cambios, aunque no sobra el dinero.
- Actualmente todos los ordenadores utilizan sistemas Linux.
- Es necesario tener un buen soporte técnico en caso de que haya problemas. No pueden permitirse tener a un empleado dedicado a ello, prefieren llamar a un técnico cuando algo falle.

¿Qué SGBD utilizaría y por qué?

SOLUCIÓN

Oracle Database, porque cumple con todos los requisitos. Es altamente fiable, cuenta con soporte especializado y funciona en *Linux*, mientras que *Microsoft SQL Server* no. Técnicamente cualquier SGBD podría haber sido utilizado a excepción de *SQLite*, pero *Oracle Database* sería la opción que menos problemas daría.



Actividades

13. Investigue en Internet el precio para desplegar una base de datos mediante *Oracle Database* y *Microsoft SQL Server*.
14. Busque *Oracle*, *SQL Server* y *MySQL* en un portal de búsqueda de trabajo online para hacerse una idea de cómo está el mercado actualmente.

9. Instalación del SGBD *MySQL*

A continuación, se tratará el SGBD *MySQL*. Es uno de los más utilizados en programación web y es código abierto.

Dependiendo del sistema en que se trabaje, se tendrá que instalar de una manera u otra. Desde sistemas *Linux* se puede utilizar “apt-get install mysql” o “yum install mysql” desde línea de comandos.



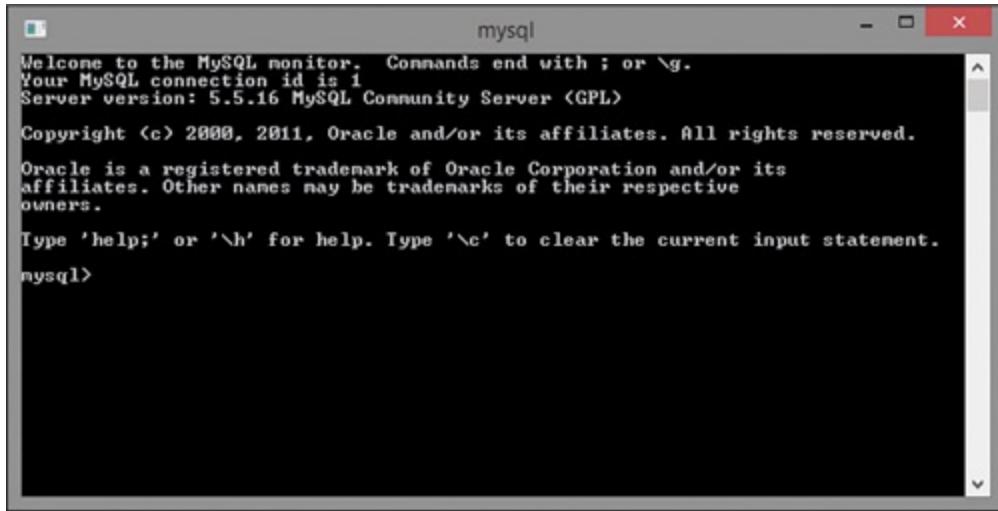
Nota

Muchas instalaciones de *Linux* ya lo incorporan.

Desde *Windows* se puede descargar un instalador desde <<http://dev.mysql.com/downloads/>>. Desde ahí, habrá que seleccionar “MySQL Community Server”, que es la opción libre más utilizada y después simplemente hay que seguir los pasos de la web y del instalador.

Una vez se haya instalado *MySQL*, será posible conectarse a él desde el cliente por línea de comandos. Desde *Linux* es más sencillo, desde la línea de comandos se teclea “mysql –uroot” y ya está.

Desde *Windows*, el instalador de *MySQL* habrá creado un acceso directo a la consola de *MySQL*. Haciendo doble clic se abrirá.



```
mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 5.5.16 MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Al abrirlo lo que se encuentra es esto. Al principio se ve el número de versión de *MySQL* e información sobre los términos de la licencia y demás. *Mysql>* es el cursor de la consola. Ahí es donde se introducen las consultas de SQL.

A lo largo del manual se insertan capturas de pantalla de la consola de *MySQL* desde *Windows*, pero el comportamiento es idéntico en cualquier sistema.



Importante

Asegúrese de ser capaz de conectarse al cliente de *MySQL* por línea de comandos para poder realizar los ejercicios y seguir los ejemplos del próximo capítulo, porque se utilizará *MySQL* en ellos.

10. Conexión a *MySQL* desde un lenguaje de servidor

Una de las características más interesantes de los SGBD es que proporcionan interfaces mediante las cuales es posible comunicarse con ellos desde prácticamente cualquier plataforma.

La manera más común de comunicarse con un SGBD es mediante un lenguaje de programación. Aquí se estudiará cómo conectarse a *MySQL* utilizando el lenguaje de servidor PHP. La combinación de PHP+*MySQL* es una de las más habituales en lo que a desarrollo web se refiere.

PHP cuenta con el conjunto de funciones para conectarse con *MySQL* de forma nativa, no es necesario instalar ninguna librería o conector como sí podría ser necesario en Java o .NET.

Los pasos a seguir cuando se comunica con un SGBD suelen ser los siguientes:

1. Conectarse al SGBD. Se dice dónde está el servidor, el usuario y contraseña y se realiza la conexión.
2. Un SGBD está compuesto por múltiples bases de datos. En el segundo paso se selecciona a qué base de datos se desea conectar.
3. Enviar una consulta SQL al servidor utilizando la conexión que se ha abierto.
4. Recibir los datos que el SGBD devuelve para esa consulta. Puede devolver un *resultset* (lista de filas y columnas) o un número (para indicar éxito/error).

Ahora se mostrará un ejemplo muy básico de cómo conectarse mediante PHP a un *MySQL*. Para ello, primero, abra la consola de *MySQL* y ejecute las siguientes consultas de SQL, ello le permitirá probar con una base de datos ya creada:

```
CREATE DATABASE prueba_php;
USE prueba_php;

CREATE TABLE provincias (
    id_provincia smallint(6) DEFAULT NULL,
    provincia varchar(30) DEFAULT NULL
) ENGINE=InnoDB;

INSERT INTO provincias (id_provincia, provincia)
VALUES
(2,'Albacete'), (3,'Alicante/Alicant'), (4,'Almería'), (1,'Araba/Álava'), (33,'Asturias'),
(5,'Ávila'), (6,'Badajoz'), (7,'Balears, Illes'), (8,'Barcelona'), (48,'Bizkaia'),
(9,'Burgos'), (10,'Cáceres'), (11,'Cádiz'), (39,'Cantabria'), (12,'Castellón/Castelló'),
(51,'Ceuta'), (13,'Ciudad Real'), (14,'Córdoba'), (15,'Coruña, A'), (16,'Cuenca'),
(20,'Gipuzkoa'), (17,'Girona'), (18,'Granada'), (19,'Guadalajara'), (21,'Huelva'),
(22,'Huesca'), (23,'Jaén'), (24,'León'), (27,'Lugo'), (25,'Lleida'), (28,'Madrid'),
(29,'Málaga'), (52,'Melilla'), (30,'Murcia'), (31,'Navarra'), (32,'Ourense'),
(34,'Palencia'), (35,'Palmas, Las'), (36,'Pontevedra'), (26,'Ríoja, La'),
(37,'Salamanca'), (38,'Santa Cruz de Tenerife'), (40,'Segovia'), (41,'Sevilla'),
(42,'Soria'), (43,'Tarragona'), (44,'Teruel'), (45,'Toledo'), (46,'Valencia/valència'),
(47,'Valladolid'), (49,'Zamora'), (50,'Zaragoza');
```

Ahora, hay que conectarse a la base de datos “prueba_php”, la cual se encuentra en la máquina local (*localhost*):

```

1 <?php
2 // conectarse a mysql
3 $conexion = mysql_connect("localhost", "root", "");
4
5 // seleccionar la base de datos
6 mysql_select_db("prueba_php", $conexion);
7
8 // enviamos una consulta al servidor
9 $sql = "SELECT id_provincia, provincia FROM provincias";
10 $resultados = mysql_query($sql, $conexion);
11
12 // recorremos cada resultado
13 echo "<ul>";
14 while ($fila = mysql_fetch_assoc($resultados))
15 {
16     echo "<li>" . $fila["provincia"] . "</li>";
17 }
18 echo "</ul>";
19
20 mysql_close($conexion);
21

```

En los comentarios del código se explica cada paso, pero se añadirá una explicación más detallada:

- Primero se crea un objeto de conexión. `Mysql_connect` recibe primero el nombre del servidor (`localhost`, una dirección IP o una URL), luego el nombre de usuario y, por último, la contraseña. En una instalación por defecto de *MySQL*, la única cuenta que hay es `root`. Si se poseen otros datos, deben cambiarse ahí.
- Luego, se selecciona la base de datos que va a utilizarse. El segundo parámetro es el objeto de conexión aplicable. Es posible tener más de una conexión abierta al mismo tiempo, por eso es necesario especificarla.
- A continuación, se crea una cadena de texto con la consulta SQL que se quiere lanzar. Se hace así para que en caso de que falle la consulta, se pueda recurrir a `$sql`, verla de primera mano y probarla.
- Luego con “`mysql_query`” se envía la consulta al servidor, la cual devuelve un *resultset* en la variable `$resultados`.
- Hay varias formas de recorrer los resultados de una consulta. Una de las más sencillas es utilizando “`mysql_fetch_row`”, la cual va recorriendo el *resultset* como un cursor, devolviendo el valor de la fila actual y moviendo el cursor hacia la siguiente fila.
- La variable `$fila`, en cada iteracción del bucle contendrá un *array* asociativo cuyos índices tienen el nombre de cada columna devuelta por *MySQL*.
- **Siempre debe cerrarse la conexión** antes de que termine el *script*. PHP trataría

de hacerlo automáticamente, pero se corre el riesgo de que esto no sea así y se acabe colapsando la base de datos si hay muchas conexiones abiertas de usuarios que no están haciendo nada. Para ello, se utiliza “mysql_close”.

Se ha visto un ejemplo básico de cómo conectarse. Sin embargo, tan pronto como la aplicación crezca, este código será inmanejable y puede suceder que se encuentre con más de una conexión al mismo tiempo y teniendo que copiar y pegar mucho código.

Para ello, lo habitual es crear una **clase de conexión a base de datos**, la cual contendrá la conexión actual y una serie de métodos para comunicarse con ella más fácilmente.

Una clase sencilla de ese estilo podría ser la siguiente:

```

1 <?php
2 class Db {
3     public $conexion;
4
5     function __construct()
6     {
7         $this->conexion = mysql_connect("localhost", "root", "");
8         mysql_select_db("prueba_php", $this->conexion);
9     }
10    function __destruct()
11    {
12        mysql_close($this->conexion);
13    }
14    function consulta($sql)
15    {
16        $resultados = mysql_query($sql, $this->conexion);
17
18        $salida = array();
19        while ($fila = mysql_fetch_assoc($resultados))
20        {
21            $salida[] = $fila;
22        }
23        return $salida;
24    }
25    function insercion($sql)
26    {
27        if (mysql_query($sql, $this->conexion))
28        {
29            return mysql_insert_id($this->conexion);
30        }
31
32        return NULL;
33    }
34    function modificacion($sql)
35    {
36        mysql_query($sql, $this->conexion);
37        return mysql_affected_rows($this->conexion);
38    }
39    function borrado($sql)
40    {
41        mysql_query($sql, $this->conexion);
42        return mysql_affected_rows($this->conexion);
43    }
44    function error()
45    {
46        return mysql_error($conexion);
47    }
48 }

```

Para utilizar esta clase, simplemente habría que hacer algo como:

```

50 $db = new Db();
51
52 $filas = $db->consulta("SELECT provincia FROM provincias WHERE LENGTH(provincia) > 5");
53
54 // recorremos cada resultado
55 echo "<ul>";
56 foreach ($filas as $fila)
57 {
58     echo "<li>" . $fila["provincia"] . "</li>";
59 }
60 echo "</ul>";
61
62

```

La clase de conexión es útil cuando se quiere empezar a utilizar MySQL con PHP,

pero tiene mucho margen de mejora.

10.1. Consultas SQL que contienen información introducida por el usuario

Una de las reglas básicas en programación informática es “**nunca te fíes de la entrada del usuario**”, siempre puede haber algún usuario malicioso o despistado que haga que tu programa deje de funcionar y sea inseguro.

Es muy habitual que las consultas de SQL lanzadas desde PHP contengan datos que dependan de la situación en que se encuentre (ese es el fin de las páginas web dinámicas).



Ejemplo

Crear un buscador en el que el usuario introduzca una palabra y devuelva las provincias que se parezcan.

Suponiendo que se tiene un formulario que apunta al siguiente *script* de PHP, mandando un parámetro GET llamado “búsqueda”:

```
$sql = "SELECT provincia FROM provincias WHERE provincia LIKE '%".$_GET["busqueda"]."%';  
$filas = $db->consulta($sql);
```

Ahí se indagan provincias que contengan lo que el usuario haya buscado. Pueden surgir los siguientes problemas:

- **SQL invalidada.** El usuario busca algo que contiene una comilla simple: la consulta quedaría algo así como: *SELECT provincia FROM provincias WHERE provincia LIKE '%malaga%'* lo cual tiene una sintaxis inválida.
- **SQL Injection.** Este es uno de los problemas de seguridad más comunes en programación web y ocurre por despiste o desconocimiento de los programadores de la web. Consiste en que el usuario es capaz de modificar los parámetros de tal forma que *jueguen* con el resultado de la consulta. Por ejemplo, un usuario podría buscar “cuenca’ OR provincia LIKE ‘ “ y hacer que la consulta quedara como *SELECT provincia FROM provincias WHERE*

provincia LIKE '%cuenca' OR provincia LIKE '%' lo cual devolvería todos los resultados. Es un ejemplo muy básico, pero cuando aprenda el estándar SQL verá que se pueden hacer otras cosas mucho más potentes.

Ante estos casos, deben observarse las siguientes precauciones:

- Cuando una consulta contiene texto de dudosa procedencia (el usuario), debe aplicarse la función “`mysql_real_escape_string()`” sobre ellas, la cual convertirá comillas y eliminará cualquier posibilidad de ataque malintencionado.
- Si lo que se va a utilizar es un número, puede interesar hacer *casting* de la variable para forzar el tipo de dato que se quiere, de tal forma que si el valor es inválido, se utilice un cero o algún otro valor que sí sea seguro.

11. Resumen

Un SGBD es un conjunto de aplicaciones informáticas altamente complejas que sirven para facilitar las tareas de mantenimiento de los datos de una aplicación. Proporciona un sistema fiable y escalable en el que guardar los datos.

Son una evolución de los ficheros de datos, pero con gestión de restricciones de integridad, opciones de concurrencia, transacciones y lenguajes propios para comunicarse con ellos.

Con los SGBD surge una nueva figura que es la del DBA (administrador de bases de datos: una persona especializada y con experiencia trabajando con el SGBD que utilice la empresa. El DBA se encargará de crear las bases de datos, administrarlas y de asegurarse de que son escalables.

Dos aspectos básicos de los SGBD son las transacciones y la concurrencia. Una transacción es un conjunto de operaciones que se envían a la base de datos y que se gestionan como una sola. La concurrencia es la habilidad que tiene un SGBD para manejar transacciones simultáneas que trabajen sobre los mismos datos.

Elegir un SGBD es una decisión técnica de gran transcendencia en una empresa. Determinará la manera de escalar la base de datos, su rendimiento y la manera de usarla. Siempre hay que saber por qué se elige un sistema u otro.



Ejercicios de repaso y autoevaluación

1. De los siguientes nombres, diga cuál es un SGBD y cuál no:

- a. XML
- b. MySQL
- c. Oracle
- d. Oracle Database

2. Un SGBD es un conjunto de programas cuyo propósito es gestionar bases de datos relacionales.

- Verdadero
- Falso

3. ¿Qué diferencia hay entre el gestor de almacenamiento y el gestor de consultas de un SGBD?

4. Seleccione cuáles de las siguientes aptitudes debe tener un DBA.

- a. Amplio conocimiento sobre redes informáticas.
- b. Capacidad de comunicarse tanto con personas de perfil técnico como con personas no técnicas.
- c. Conocimiento del modelo relacional y sobre normalización.
- d. Conocimiento de algún lenguaje de programación.

5. ¿Qué se busca al crear un índice en una tabla?

- a. Evitar accesos repetitivos.
- b. Determinar la ubicación de un dato sin tener que leer los demás.
- c. Colocar los datos más utilizados en memoria para un acceso más rápido.
- d. Todas las respuestas anteriores son correctas.

6. ¿Qué es la cardinalidad de un índice?

- a. El número de pasos que debe dar el motor de almacenamiento para determinar dónde se encuentra un dato físicamente.
- b. El número de valores distintos por los cuales agrupa los datos que contiene.
- c. El tamaño que ocupa un índice en el soporte físico que se utilice (normalmente, un disco duro).
- d. Las respuestas a y b son correctas.

7. ¿Cuál de las siguientes situaciones supondría un riesgo en la seguridad de un SGBD y, por lo tanto, debe estar cubierta por el DBA?

- a. Borrado accidental de una tabla.
- b. Un servidor de base de datos recibiendo un volumen de peticiones inesperado.
- c. Un usuario leyendo datos de una tabla que se supone que es privada.
- d. Todas las respuestas anteriores son correctas.

8. Tanto la creación como la restauración de copias de seguridad es responsabilidad del DBA.

- Verdadero
- Falso

9. ¿Qué es más importante?

- a. Que las copias de seguridad sean fáciles de recuperar.
- b. Que las copias de seguridad se efectúen con mucha frecuencia.
- c. Que las copias de seguridad se guarden en un soporte físico distinto al que utiliza el SGBD.

10. ¿En qué consiste la replicación?

11. Los sistemas de replicación solo están disponibles en SGBD de alto rendimiento y de pago.

- Verdadero
- Falso

12. ¿Por qué son útiles las transacciones?

13. ¿Cuál de estas situaciones puede suponer un problema serio de rendimiento en un SGBD?

- a. Ausencia de índices.
- b. Muchas transacciones bloqueadas
- c. Muchos usuarios leyendo datos al mismo tiempo.
- d. Todas las respuestas anteriores son correctas.

14. A la hora de escoger un SGBD hay que centrarse principalmente en su precio y su rendimiento:

- Verdadero
- Falso

15. ¿Es cierto que, normalmente, cuando se escoge un SGBD no hay vuelta atrás y por eso es importante seleccionarlo correctamente desde el principio?

- a. Sí hay vuelta atrás, pero dependiendo de cuándo se haga el cambio puede requerir un costoso periodo de migración.
 - b. Sí hay vuelta atrás y es muy sencillo cambiarlo, porque todos utilizan SQL, que es estándar. Simplemente habría que cambiar los datos de conexión a la base de datos.
 - c. No hay vuelta atrás. Habría que hacerlo todo desde cero. Todos los SGBD utilizan SQL, pero los dialectos de este lenguaje son tan diferentes que no es viable una migración.
 - d. Hay vuelta atrás, pero no compensa. Cuando se escoge un SGBD, se puede decir que es una decisión definitiva.
-

Capítulo 3

Lenguajes de gestión de bases de datos. El estándar SQL

1. Introducción

Ya se ha visto en qué consiste el modelo relacional y qué aportan los sistemas gestores de bases de datos. En este capítulo se aplicarán todos estos conocimientos de forma práctica.

Los SGBD deben permitir ser manipulados mediante un sistema de consultas. En los SGBD relacionales el sistema utilizado es el SQL (*Structured Query Language* o Lenguaje estructurado de consultas, en español).

Mediante este lenguaje es posible realizar todo tipo de operaciones: crear tablas, añadir registros a esas tablas, modificarlos, eliminarlos, leer datos de varias tablas simultáneamente, etc.

Es **necesario tener gran destreza** en el manejo del SQL puesto que resulta fundamental para **interactuar** eficientemente **con las bases de datos**. Cada SGBD utiliza una versión ampliada del SQL estándar con el fin de añadir más opciones o atajos a opciones existentes. Como este manual está eminentemente enfocado a MySQL, primero se explicarán los elementos estándar de cada cosa y luego, las particularidades de MySQL.

2. Descripción del estándar SQL

El SQL (*Structured Query Language* o lenguaje estructurado de consultas) es un lenguaje estándar que sirve para comunicarse con fuentes de datos (normalmente, bases de datos relacionales).

Sus orígenes se remontan a finales de los años 70. Fue creado originalmente por IBM, pero Oracle lo implementó por primera vez en un programa comercial. Al principio se llamaba SEQUEL (*Structured English Query Language*).

En el año 1986 fue estandarizado por el ANSI (Instituto Nacional Estadounidense de

Estándares). A esta versión inicial se le llama SQL1 o SQL-86 (por el año de creación). En el primer estándar ya se incluyen la mayoría de elementos tratados en este libro.



Nota

Más adelante se fueron añadiendo nuevos tipos de datos, soporte para procedimientos almacenados, etc., pero la sintaxis se mantuvo.

Un ejemplo de una consulta de SQL puede ser la siguiente:

```
SELECT id, nombre, precio
FROM productos
WHERE fecha_alta >= '2013-01-01'
ORDER BY fecha_alta DESC;
```

Aquí se estaría devolviendo una tabla parecida a la siguiente:

Id	Nombre	Precio
3939	Plancha a vapor	49.95 €
6996	Batidora de varillas	12.99 €
4040	Tostadora	15 €

Lo que se pide son los campos id, nombre y precio de la tabla de productos que hayan sido introducidos después del 1 de enero de 2013 ordenados por orden cronológico, el más reciente primero y el más antiguo, después.

El SQL es un **lenguaje bastante natural** sobre todo para angloparlantes porque se asemeja mucho al lenguaje hablado. Se lee de izquierda a derecha y está estructurado por secciones (delimitadas por palabras clave como SELECT, FROM, WHERE). Las consultas **siempre terminan en punto y coma** (;) indicando el final de las mismas.

En este libro se escriben las palabras clave siempre en mayúscula y los nombres de campos y tablas en minúscula. De esta manera, lo que esté en minúscula quiere decir que ha sido definido por el DBA y lo que está en mayúscula significa que es una característica del SGBD o del SQL en general.

Se enfocará SQL hacia el SGBD *MySQL*, puesto que es **el más utilizado en desarrollo de aplicaciones web**. El SQL estándar por sí mismo no tiene sentido completo.



Nota

Lo importante del SQL es la sintaxis que propone, que es lo que se mantiene entre todos los SGBD.

Además, el SQL define una serie de funciones y tipos de datos comunes con el fin de facilitar la compatibilidad entre distintos SGBD, pero para tener una aplicación eficiente siempre necesitarán usarse funciones y tipos de datos que no están incluidos en el estándar.

Esto no quiere decir que mediante un SQL estándar no se pueda explotar una base de datos. Simplemente, se estarán desaprovechando muchas funcionalidades del SGBD empleado, aunque es bueno conocer los elementos comunes de cara a hacer aplicaciones compatibles con varios sistemas.

3. Creación de bases de datos

Un SGBD debe proporcionar tres lenguajes: **el DDL, DCL y el DML**. En este apartado se estudiará el DDL, el **lenguaje de definición de datos**. Este lenguaje se utiliza para crear bases de datos, sus tablas con sus columnas e índices y las claves foráneas que existan.

Todo ello se hace utilizando el lenguaje de SQL. Cada SGBD tiene una serie de tipos de datos adicionales para tratar con datos especiales (como puntos geográficos, datos binarios o de cara a simplificar las cosas).

Lo **primero** que siempre se precisa es **crear una base de datos**. En esa base de

datos es donde se crearán las tablas que vayan a utilizarse. Para ello, se usará la sentencia CREATE DATABASE seguida del nombre que se vaya a utilizar.

```
CREATE DATABASE tienda;
```



Importante

En MySQL no es posible editar el nombre de una base de datos creada, por lo que siempre habrá que tener esto en cuenta.

También hay que saber que no es recomendable utilizar nombres de base de datos con caracteres especiales. Deben eliminarse acentos, espacios y demás. Lo ideal son nombres en minúsculas con las palabras separadas por guiones bajos.

Como ya se ha indicado, editar el nombre de una base de datos no es posible; solo es posible eliminarla. Para ello, se utilizaría DROP DATABASE.

```
DROP DATABASE tienda;
```

No ejecute la sentencia de borrado; cree la base de datos “tienda” puesto que es la que se utilizará a partir de ahora a lo largo del libro.

Borrar bases de datos es una tarea muy arriesgada y hay que pensarlo dos veces antes de hacerlo porque no se pide confirmación y no hay vuelta atrás una vez se efectúa.

Las bases de datos **no tienen propiedades** por sí solas, solamente un nombre.



Nota

En MySQL sí tienen algunas propiedades extra como la colación por defecto (la manera de gestionar acentos y caracteres especiales) pero eso no pertenece al estándar y son características avanzadas del sistema.

Una vez creada la base de datos, hay que indicar al sistema que quiere usarse. Esto quiere decir que el SGBD sabrá que cuando se creen tablas o se realicen consultas, la base de datos a utilizar será esa. Para ello, se utilizará la sentencia USE (solo para MySQL).



USE tienda;

USE hace que la base de datos actual sea la que se indique. A partir de ese momento MySQL asumirá que esa es la base de datos a utilizar.

3.1. Creación de tablas. Tipos de datos

Una vez la base de datos está creada, lo que hay que hacer es **crear las tablas** y definir sus columnas. Para cada columna se debe especificar el **tipo de dato** que va a contener. Los tipos suelen ser números, texto, fechas, etc. A continuación, se muestra la lista de tipos de dato estándar:

Tipo	Uso
CHAR(N)	Cadena de caracteres de ancho fijo (de longitud N). Si sobran caracteres se rellenan con espacios.
VARCHAR(N)	Cadena de caracteres de ancho variable con una longitud máxima de N.
NCHAR(N)	Igual que CHAR pero con soporte de caracteres internacionales (no muy extendido)
NVARCHAR(N)	Igual que VARCHAR pero con soporte de caracteres internacionales (no muy extendido)
BIT(N)	Array de bits. Se guardan datos en binario. Muy poco extendido.

INTEGER	Número entero (sin decimales).
FLOAT, REAL, DOUBLE PRECISION	Números con decimales.
NUMERIC (precisión, escala) o DECIMAL (precisión, escala)	Números con precisión específica. DECIMAL (12, 3) significa un número de 12 cifras cuyas 3 últimas son decimales.
DATE	Una fecha.
TIME	Una hora.
TIMETZ	Una hora pero con detalles sobre la zona horaria usada.
TIMESTAMP	Una fecha junto a una hora.
TIMETSAMPTZ	Una fecha junto a una hora y con información sobre la zona horaria.

Estos son los tipos de dato estándar. *MySQL* implementa estos tipos y además incluye algunos extra.

A continuación, se tratarán los tipos de datos de *MySQL*.

Particularidades en tipos de datos de texto

Se admiten los tipos de datos TINYTEXT/MEDIUMTEXT/TEXT/LONGTEXT cuya longitud depende de su **tamaño en bytes** (no del número de caracteres). Son útiles para almacenar textos amplios (texto de noticias, descripciones, entradas de blog, etc.). Lo habitual es utilizar MEDIUMTEXT.

Se añaden los tipos de datos ENUM y SET. ENUM es una **enumeración de posibles valores** que puede contener un campo. Si se intenta añadir un valor que no está en esa enumeración, se devolverá un error. El tipo de dato SET es parecido, pero permite que se añadan **combinaciones entre los valores enumerados**. Es decir, un SET ('1', '2', '3') permitiría un valor que fuera "12", pero un ENUM ('1', '2', '3') no lo permitiría.



Nota

El ENUM está bastante extendido, el SET no.

Particularidades de los tipos de dato numéricos

Los números enteros se descomponen según su tamaño en TINYINT, SMALLINT, MEDIUMINT, INT y BIGINT. Esto se hace para **aumentar la eficiencia**, puesto que los tipos de datos más pequeños serán más fáciles de manejar por el SGBD, porque ocupan menos espacio.

Además, se permite eliminar el signo de los números mediante la palabra clave **UNSIGNED**. Así, si un número siempre incluye valores positivos (como un id auto numérico, por ejemplo), es posible quitarle el signo y tendrá una longitud máxima del doble (porque no se utilizan los valores negativos).

En la siguiente tabla se ven los valores mínimos y máximos que puede albergar cada tipo de dato numérico:

Tipo	Valor mínimo	Valor máximo
TINYINT	-128	127
TINYINT UNSIGNED	0	255
SMALLINT	-32768	32767
SMALLINT UNSIGNED	0	65535
MEDIUMINT	-8388608	8388607
MEDIUMINT UNSIGNED	0	16777215
INT	-2147483648	2147483647
INT UNSIGNED	0	4294967295
BIGINT	-9223372036854775808	9223372036854775807
BIGINT UNSIGNED	0	18446744073709551615

Esto es útil para cualquier número que sirva de contador. Por ejemplo, los kilómetros de un vehículo siempre serán mayores o iguales a cero.

Particularidades de los tipos de dato de fecha

Se introduce el tipo de dato DATETIME. El *timestamp* guarda fechas y horas pero

solo si son de después del 1 de enero de 1970. El campo DATETIME permite fechas anteriores. Esto es así porque el *timestamp* se utiliza para guardar **eventos de la aplicación** (que, por lo tanto, ocurren normalmente después de haber creado la tabla), no fechas por sí solas. Internamente, los *timestamps* se guardan con un valor numérico, calculando el número de segundos transcurridos desde el 1 de enero de 1970. Esto es así por el estándar de fechas de UNIX, el cual está ampliamente adoptado en la computación informática.

Tipos de dato binarios

Es posible que deseemos guardar datos binarios en la base de datos (como imágenes, por ejemplo). En la mayoría de los casos esto será una mala práctica que deberá evitarse, pero es bueno saber que es posible guardar datos en formato binario.



Nota

Estos datos se guardan de forma especial dentro del gestor de almacenamiento y son manipulados de diferente manera a veces.

Los tipos de datos binarios son TINYBLOB, MEDIUMBLOB, BLOB, LONG-LOB. La palabra BLOB es la abreviatura de “objeto binario grande”. Su nombre depende del número de bytes que ocupan.

Creación de tablas mediante SQL

Una vez se han visto todos los tipos de datos ya pueden **crearse tablas**. Para ello, se utilizará la siguiente sintaxis:

```
CREATE TABLE producto (
    id INT UNSIGNED PRIMARY KEY,
    titulo VARCHAR(100) NOT NULL DEFAULT '',
    precio DECIMAL(12, 2) NOT NULL DEFAULT 0,
    fecha_alta TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

Este es el primer ejemplo sobre cómo crear una tabla. Los requisitos que se buscan son estos:

- Nombre de la tabla. En este caso “producto”. No debe tener espacios ni caracteres extraños. En caso de tenerlos, se debería encerrar el nombre entre comillas invertidas ‘producto’.
- Cada nombre de columna separado por comas. La misma restricción de nombres se aplica. No deben tener caracteres extraños.
- Para cada columna, su tipo de dato, longitud (si se aplica), sus restricciones de integridad y su valor por defecto (opcional).

Observaciones a la hora de crear una tabla:

- **Siempre debe haber una clave primaria.** Es posible crear una tabla sin clave primaria y MySQL no dará ningún error, pero debe haberla. En este caso se especificará que “id” es la clave primaria, añadiendo las palabras “PRIMARY KEY” al final de la definición del campo.
- Algunos tipos de datos requieren que se especifique **su longitud** y otros no. Si un campo lo necesita y no se especifica, se recibirá un error.
- En algunos SGBD los campos por defecto admiten nulos y en otros no. En MySQL los campos **por defecto admiten nulos**. Si se quiere evitar que esto ocurra habrá que especificar que la columna es NOT NULL. Siempre que sea posible hay que elegir NOT NULL porque es más eficiente. Hay que utilizar los campos *nullables* solo cuando hagan falta. Para campos NOT NULL es conveniente añadir un valor por defecto para las nuevas filas. Así se evitan posibles errores al insertar datos después. Un campo *nullable* es aquel que admite nulos.
- El valor por defecto de un campo debe ser del **mismo tipo de dato** que el propio campo. Especificar DEFAULT NULL en un campo *nullable* sería redundante y no tendría sentido. Especificar DEFAULT NULL en un campo NOT NULL daría error y no podría crearse la tabla. Solo es posible especificar valores por defecto simples (números, texto o NULL). No se pueden utilizar funciones especiales. Tan solo hay una excepción para los campos de tipo *timestamp*, que admiten el valor CURRENT_TIMESTAMP, el cual contiene la fecha y hora en el momento de insertar el dato (útil para tener un campo con el dato de la fecha de creación de algo).



Consejo

Al crear tablas es bueno seguir patrones al listar las columnas. Por ejemplo, un patrón podría ser especificar la clave primaria primero, las claves foráneas después y, por último, los campos normales de la tabla.

Observaciones al crear campos de fecha

Al crear campos de tipo *timestamp* es posible utilizar la variable CURRENT_TIMESTAMP para los campos. Los campos *timestamp* tienen una **gran cantidad de restricciones** en MySQL, las cuales conviene conocer. Para ello primero se mostrará un ejemplo:

```
CREATE TABLE noticia (
    id INT UNSIGNED PRIMARY KEY,
    título VARCHAR(100) NOT NULL DEFAULT '',
    contenido MEDIUMTEXT NOT NULL DEFAULT '',
    fecha_alta TIMESTAMP NOT NULL DEFAULT '0000-00-00 00:00:00',
    fecha_modificación TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
    fecha_borrado TIMESTAMP NULL
);
```

En esta tabla se quiere mantener un registro de la fecha de creación del registro, fecha de modificación y fecha de borrado (se supone que cuando el campo está a NULL es porque no se ha borrado aún).

Solo es posible tener una columna de tipo TIMESTAMP que utilice CURRENT_TIMESTAMP. Las demás deben tener un valor por defecto.

Dado que la fecha de alta no admite nulos porque siempre debe tener un valor, debe especificarse un valor por defecto. Lo ideal es 0000-00-00 00:00:00 que significa que la fecha está vacía.

Además de permitir especificar CURRENT_TIMESTAMP como el valor por defecto, MySQL proporciona un método extra llamado **ON UPDATE CURRENT_TIMESTAMP** que hará que el campo actualice su valor a la fecha actual cada vez que el registro en cuestión reciba una modificación.



Actividades

1. Cree una tabla de nombre “hotel”. Debe tener los campos hotel_id,

hotel_nombre, hotel_direccion y hotel_telefono. La clave primaria es hotel_id, es un código de tres caracteres (como ABC, 1HA, etc.). El nombre, dirección y teléfono son campos de texto. Elija la longitud que desee.

Modificar tablas

Es muy habitual tener que realizar cambios en tablas ya creadas. No siempre los requerimientos finales de las aplicaciones serán los mismos, por lo que hay que estar preparados para hacer cambios en la estructura de las tablas para amoldarse a la situación.

Para modificar tablas se utiliza la sentencia **ALTER TABLE**. Tiene un funcionamiento bastante similar a CREATE TABLE, aunque dominarlo es bastante más complejo porque hay más palabras clave a memorizar.

Añadir campos a una tabla

Para crear un campo en una tabla, se utiliza ALTER TABLE ADD COLUMN. Puede probarse creando un campo llamado “categoría” en la tabla de noticias:

```
ALTER TABLE noticia ADD COLUMN categoria VARCHAR(100) NULL AFTER contenido;
```

Se empieza con ALTER TABLE seguido del nombre de la tabla a modificar. A continuación, se especifica la operación que se quiere realizar, en este caso se desea **añadir una columna**, así que se usa ADD COLUMN. Después se pone el nombre del nuevo campo y su tipo, de igual manera que si se estuviera creando la tabla, con la misma sintaxis. Al final se tiene una opción que sirve para **posicionar la columna** en un lugar concreto. En el modelo relacional se decía que el orden de las columnas era irrelevante, pero es cierto que el orden sirve a la hora de gestionar la base de datos para verlo todo más ordenado.

Si se quiere especificar un orden, al final de la definición del campo se añade AFTER seguido del nombre de columna que debe ir delante. Esta columna debe existir, de lo contrario se obtendría error. Si no se especifica nada, el campo se añade **al final**.

de la tabla. No es posible utilizar la palabra BEFORE, solo se puede utilizar AFTER. Como se ha indicado, es opcional.

Modificar una columna existente

Para modificar una columna se utiliza ALTER TABLE MODIFY COLUMN. Se probará modificando el campo “categoría” que se acaba de crear para hacer que solo tenga 50 caracteres y no admita nulos:

```
ALTER TABLE noticia MODIFY COLUMN categoria VARCHAR(50) NOT NULL;
```

La sintaxis es igual que para crear una columna pero cambiando ADD por MODIFY. Es importante que la columna exista en la tabla. Puede utilizarse la palabra **AFTER** también si se quiere mover la columna de posición. Si no se especifica nada, la columna sigue en su posición original.

Eliminar una columna

Para eliminar una columna se utiliza ALTER TABLE DROP COLUMN. Se eliminará la columna “categoría”:

```
ALTER TABLE noticia DROP COLUMN categoria;
```

Aquí simplemente se usa DROP COLUMN sin parámetros extra.

Renombrar una columna

Es raro tener que cambiarle el nombre a una columna, pero es posible. Puede ser útil si se produce una equivocación al crearla y quiere rectificarse rápidamente. Para eso se utiliza la palabra **CHANGE**. Es igual que MODIFY COLUMN, pero permite especificar un nuevo nombre tras el original. Se cambiará el nombre de la columna “precio” de la tabla “pieza” para que sea “precio_net”:

```
ALTER TABLE pieza CHANGE precio precio_neto DECIMAL(9, 2) NULL;
```



Consejo

A veces cuando quiere modificarse una columna, no puede hacerse de forma directa porque se perdería información. Si quiere modificarse la columna (A) pero sin hacerlo directamente para no perder información, hay que crear una columna (B) en la que se realizarían las operaciones necesarias para que los datos sean correctos. A continuación, podría borrarse la columna A y renombrar B de tal manera que tenga el nombre de A.

Se utiliza la palabra CHANGE seguida del nombre original. Tras un espacio se indicará el nuevo nombre seguido de la definición completa de la columna. Pueden introducirse cambios si se desea; podría hacerse que la columna cambiara de nombre y además, no admitiera nulos, por ejemplo.

Observaciones al modificar tablas

Modificar tablas es una tarea compleja a menos que las tablas estén vacías. Siempre hay que tener en mente que las modificaciones deben hacer que los datos existentes no se pierdan y que las aplicaciones de la base de datos, en la medida de lo posible, sigan funcionando sin cambios.

Por ejemplo, si se convierte el campo “contenido” a INT, todo el texto se va a perder.

Al crear una columna se debe asegurar que tiene un valor por defecto o de que admite nulos; de lo contrario las filas existentes tendrán datos inválidos.

Al **modificar una columna** debe asegurarse de que no se pierde información de ningún tipo.

Al **borrar una columna** se asegurará que las aplicaciones que emplean la base de datos no la están utilizando, de lo contrario, dejarán de funcionar.

Renombrado de tablas

Muy rara vez hará falta cambiar el nombre de una tabla. Puede ser útil si se produce una equivocación al crearla y se quiere rectificar justo a continuación o quizás si nos dan una base de datos diseñada pobremente y nos encargan la tarea de mejorar su estructura. Para hacerlo se utiliza ALTER TABLE RENAME. La tabla de noticias pasa a llamarse “noticias” en vez de “noticia”:

```
ALTER TABLE noticia RENAME noticias;
```

Hay que considerar siempre si la tabla está siendo referenciada desde alguna aplicación que explote la base de datos antes de renombrarla.



Importante

Si se renombra una tabla que esté siendo usada por alguna aplicación, esta dejará de funcionar a no ser que se modifique su código fuente.

Borrado de tablas

Para borrar tablas se utiliza DROP TABLE. Se va a borrar la tabla “pieza”, como ejemplo:

```
DROP TABLE pieza;
```

Hay que tener cuidado al borrar tablas porque no se pide confirmación y **accidentalmente podemos perder un montón de datos.**

3.2. Definición y creación de índices. Claves primarias y externas

Es necesario poder especificar las claves primarias, índices y atributos únicos mediante el lenguaje de definición de datos. En los anteriores ejemplos, en los cuales se creaban tablas, la columna “id” era la clave primaria. Si se quisiera crear una **clave primaria compuesta** por varias columnas esa sintaxis no habría funcionado.

Definición de la clave primaria

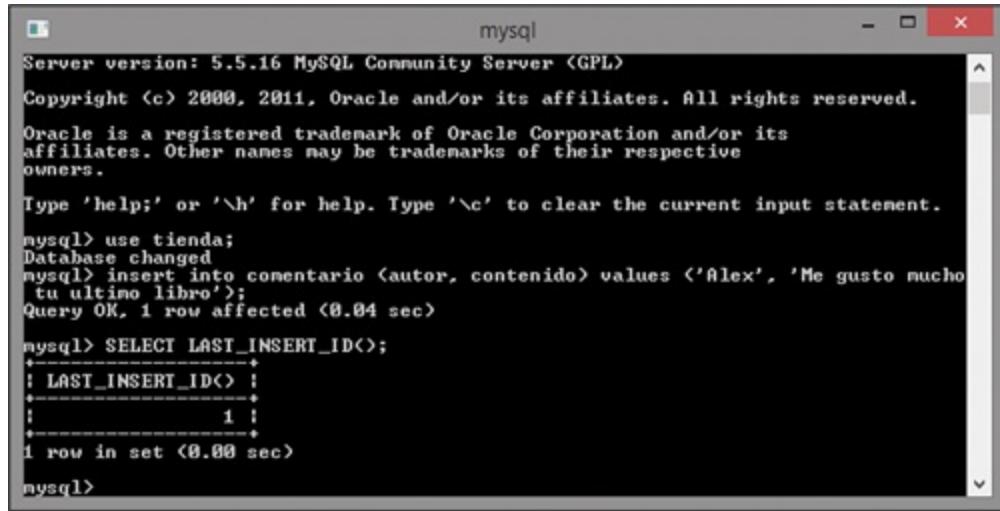
Cuando la clave primaria la compone un único campo lo más cómodo es añadir PRIMARY KEY al final de la definición de la columna. Es importante saber que los campos que componen la clave primaria deben ser **no nulos**. De lo contrario se recibirá un error.

En MySQL es común que la clave primaria sea un número entero auto numérico que empiece por el número 1 y vaya aumentando a medida que se crean nuevos registros. Para ello, debe utilizarse la palabra clave AUTO_INCREMENT, que hará que el propio SGBD cree el valor automáticamente. Esto solo se puede utilizar para claves primarias numéricas.

```
CREATE TABLE comentario (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    autor VARCHAR(50) NOT NULL DEFAULT '',
    contenido MEDIUMTEXT NOT NULL DEFAULT '',
    creacion TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

Esta palabra debe ir delante de PRIMARY KEY.

Si se necesita saber el valor que MySQL creó, nunca debería realizarse un conteo de la tabla o hacer una consulta extra para saber el mayor valor para la columna “id”. Debe utilizarse el sistema que proporciona el SGBD para obtenerlo. Esto es así porque entre que se realiza el INSERT es posible que otro usuario haya insertado un dato, por lo tanto, no sería un dato fiable. Sin embargo, si se utiliza la función de LAST_INSERT_ID() el dato será 100% fiable y devolverá el último ID generado por la conexión actual:



```
mysql
Server version: 5.5.16 MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use tienda;
Database changed
mysql> insert into comentario (autor, contenido) values ('Alex', 'Me gusto mucho
tu ultimo libro');
Query OK, 1 row affected (0.04 sec)

mysql> SELECT LAST_INSERT_ID();
+-----+
| LAST_INSERT_ID() |
+-----+
|           1      |
+-----+
1 row in set (0.00 sec)

mysql>
```

Claves primarias compuestas

Se han visto los casos especiales de campos numéricos que aumentan automáticamente y las claves primarias que solo están compuestas por una columna. Para crear una clave primaria compuesta por varios campos tendría que utilizarse la siguiente sintaxis:

```
CREATE TABLE pieza (
    departamento VARCHAR(50) NOT NULL,
    codigo VARCHAR(50) NOT NULL,
    precio DECIMAL(9,2) NULL,
    CONSTRAINT pieza_pk PRIMARY KEY (departamento, codigo)
);
```

Al final de la definición de columnas, se especifican las **cláusulas de integridad** especiales, cada una separada por comas (en este caso hay solo una).

Primero se añade CONSTRAINT, a continuación, el **nombre de la restricción** (sin espacios ni caracteres extraños y luego, el tipo de restricción (PRIMARY KEY en este caso. Después entre paréntesis se listan las columnas a las que la restricción es aplicable.

Modificar la clave primaria en una tabla

Es posible definir la clave primaria al crear una tabla, pero también es posible

cambiar la clave primaria de una tabla existente. Es una operación **realmente inusual**, porque si el modelo de datos es correcto, la clave primaria debería ser siempre la misma. Aun así, es posible hacerlo y se explicará cómo.



Consejo

Modificar la clave primaria es algo que prácticamente nunca necesitará hacerse a menos que se esté tratando de mejorar una base de datos mal diseñada realizada por otro.

Para eliminar la clave primaria de una tabla, se utilizará ALTER TABLE DROP PRIMARY KEY. Esto hará que la tabla, temporalmente, no cuente con ninguna clave primaria. Se borrará la clave primaria de la tabla pieza y se volverá a crear para ver cómo se hace:

```
ALTER TABLE pieza DROP PRIMARY KEY;  
ALTER TABLE pieza ADD CONSTRAINT pieza_pk PRIMARY KEY (departamento, codigo);
```

Especificar la clave primaria es sencillo, se utilizará ALTER TABLE ADD CONSTRAINT y lo que va después es idéntico a cuando se define una tabla con CREATE TABLE.

Para borrar la clave primaria debe tenerse en cuenta que **no es posible tener una columna con atributo AUTO_INCREMENT que no sea clave primaria** (restricción de MySQL). Primero debería eliminarse el AUTO_INCREMENT para poder borrar la clave primaria. Ahora se hará lo mismo con la tabla “comentario”, que tiene un campo AUTO_INCREMENT:

```
/* Eliminar clave primaria */
ALTER TABLE comentario MODIFY COLUMN id INT UNSIGNED;

ALTER TABLE comentario DROP PRIMARY KEY;

/* Ponerla de nuevo */
ALTER TABLE comentario ADD CONSTRAINT id PRIMARY KEY (id);

ALTER TABLE comentario MODIFY COLUMN id INT UNSIGNED AUTO_INCREMENT;
```

Creación de índices

Los índices son fundamentales para aumentar el rendimiento de la base de datos.

Hay índices normales e índices únicos. Un índice normal se utiliza simplemente para **aumentar el rendimiento** de las consultas. El índice único sirve para eso, pero además no permite que haya valores repetidos. En caso de intentar añadir un registro con un valor que ya existe se recibirá un error del SGBD.

Para índices únicos puede actuarse como con la clave primaria: añadir la palabra clave **UNIQUE** en la definición de la columna si está compuesto solo por una columna. Para índices normales esto no es posible. Vea un ejemplo:

```
CREATE TABLE cliente (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    dni CHAR(9) NOT NULL UNIQUE,
    ciudad VARCHAR(50) NOT NULL DEFAULT '',
    fecha_nacimiento DATE NULL,
    INDEX ciudad_index (ciudad)
);
```

Para el campo único, se añade la palabra clave. Para el índice debe irse al final de la definición de columnas y añadir una nueva línea. Empieza con la palabra **INDEX**, a continuación el nombre del índice (sin espacios ni caracteres extraños) y entre paréntesis la lista de campos que lo componen.

A continuación, un ejemplo con índice único compuesto por varias columnas:

```
CREATE TABLE pieza (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,
    departamento VARCHAR(50) NOT NULL,
    codigo VARCHAR(50) NOT NULL,
    precio DECIMAL(9,2) NULL,
    CONSTRAINT codigo_uq UNIQUE (departamento, codigo)
);
```

Es idéntico a crear la clave primaria múltiple pero en vez de PRIMARY KEY, se escribe UNIQUE.

Cabe destacar que los campos UNIQUE **admiten nulos** pero la PRIMARY KEY no. Si el campo UNIQUE admite nulos, estos no son considerados a la hora de comprobar si una combinación de valores está repetida. Es decir, podemos tener un campo “DNI” como UNIQUE que admita nulos. En ese caso podrá haber muchos clientes con el DNI vacío, pero ninguno que lo tenga podrá tenerlo repetido. Un índice único que no admita nulos es una clave candidata.

Modificar índices en tablas existentes

También es posible añadir, modificar y eliminar índices de tablas existentes utilizando ALTER TABLE. Elimine el índice único de la tabla “pieza” y añádalo de nuevo:

```
/* Quitar indice unico */
ALTER TABLE pieza DROP INDEX codigo_uq;

/* Añadirlo de nuevo */
ALTER TABLE pieza ADD CONSTRAINT codigo_uq UNIQUE (departamento, codigo);
```



Consejo

Normalmente cuando el modelo de datos está bien diseñado y los requisitos del sistema permanecen estables, la única modificación que se precisa sobre tablas existentes es la creación de índices para acelerar algunas consultas.

Para borrar el UNIQUE se utiliza DROP INDEX. No existe DROP CONSTRAINT en MySQL; hay que borrar índices. El UNIQUE es un tipo de índice, simplemente se utiliza su nombre y se borrará.

Para añadir el índice único sí se emplea ADD CONSTRAINT, con la sintaxis igual que al crear una tabla.

Para borrar un índice se utiliza exactamente lo mismo. Pruebe eliminando y creando el índice ciudad_index de la tabla índice:

```
/* Quitar indice */
ALTER TABLE cliente DROP INDEX ciudad_index;

/* Añadirlo de nuevo */
ALTER TABLE cliente ADD INDEX ciudad_index (ciudad);
```

Para eliminar el índice es igual que con un índice único. Para añadirlo se utiliza ADD INDEX seguido del nombre del índice junto a la lista de columnas entre paréntesis y separadas por comas.

Definición de claves externas

Ahora se verá cómo especificar las **relaciones entre tablas**, cómo declarar que una serie de columnas son clave foránea de otra tabla.

Al usar MySQL lo primero que debe saber es que para que las claves foráneas funcionen, debe utilizarse el **motor de almacenamiento InnoDB**. En muchas instalaciones el motor por defecto es MyISAM. Para forzar que las tablas sean en InnoDB, tras la definición de columnas se especificará el motor a utilizar.

Para explicar cómo crear las relaciones va a utilizarse la misma tabla mostrada en el primer ejemplo, la de productos. Se enseñará cómo forzar la utilización de InnoDB primero:

```
CREATE TABLE producto (
    id INT UNSIGNED PRIMARY KEY,
    titulo VARCHAR(100) NOT NULL DEFAULT '',
    precio DECIMAL(12, 2) NOT NULL DEFAULT 0,
    fecha_alta TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB;
```

Atención, porque esta especificación va detrás de los paréntesis y es una **sintaxis especial**. Si ya se había creado la tabla en el paso anterior se recibirá un error diciendo que la tabla ya existe; bórrela ejecutando DROP TABLE producto.

Ahora se va a crear una tabla de imágenes de un producto. Cada imagen pertenece a un producto, por lo que deberá tener una clave foránea apuntando al id del producto. Se efectuará así:

```
CREATE TABLE producto_imagen (
    id INT UNSIGNED PRIMARY KEY,
    producto_id INT UNSIGNED,
    ruta_imagen VARCHAR(255) NOT NULL,
    CONSTRAINT fk_producto FOREIGN KEY (producto_id) REFERENCES producto(id)
) ENGINE=InnoDB;
```

Aquí se aprecia una sintaxis más complicada que a la hora de crear claves primarias e índices únicos.

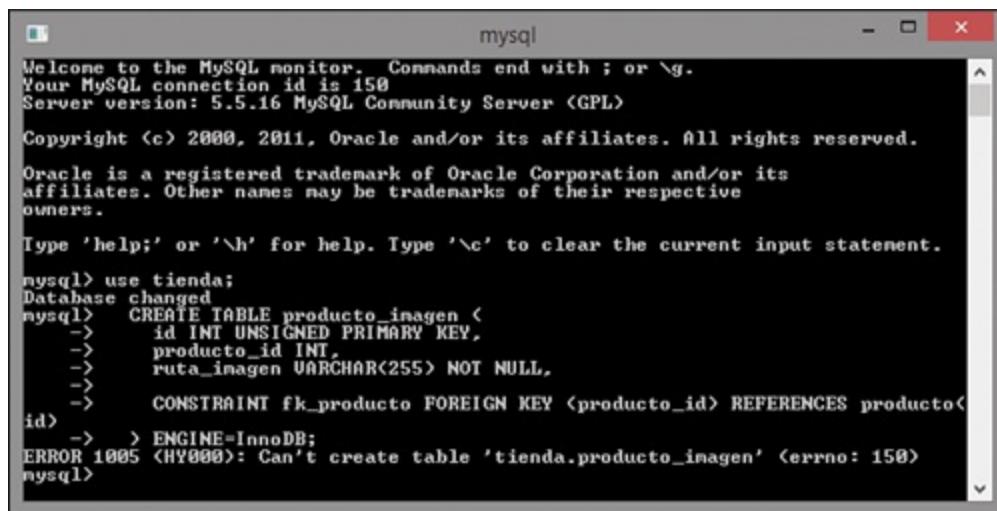
Primero se escribe **CONSTRAINT**, luego el nombre de la restricción (sin caracteres raros ni espacios. A continuación, **FOREIGN KEY** y luego, entre paréntesis la lista de columnas que la componen. Después, la palabra **REFERENCES** y luego el nombre de la tabla referenciada y entre paréntesis la lista de columnas a las que se refiere de esa tabla.

Es indispensable saber que al crear el campo de clave foránea (producto_id INT UNSIGNED el **nombre no importa**, pero el **tipo de dato ha de ser idéntico**. No puede ser parecido ni equivalente, sino exactamente igual. Si la clave primaria de “producto” es INT UNSIGNED, no es posible que “producto_id” en “producto_imagen” sea INT (sin UNSIGNED).

Sí es posible que el campo referenciado **admita nulos**. Esto sería útil en caso de relaciones de cero a muchos. Por ejemplo, una tabla de comentarios en el que algunos

comentarios hayan sido creados por usuarios registrados y otros por usuarios anónimos. En ese caso la posible clave foránea de “usuario_id” admitiría nulos, puesto que no todos los registros están enlazados a algún usuario.

Al principio es muy común encontrarse con multitud de errores al especificar claves foráneas, porque el SGBD es muy estricto en esto. Las especificaciones deben ser exactas y correctas.



The screenshot shows a terminal window titled "mysql" with the following text:

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 150
Server version: 5.5.16 MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> use tienda;
Database changed
mysql> CREATE TABLE producto_imagen (
    ->     id INT UNSIGNED PRIMARY KEY,
    ->     producto_id INT,
    ->     ruta_imagen VARCHAR(255) NOT NULL,
    ->     CONSTRAINT fk_producto FOREIGN KEY (producto_id) REFERENCES producto(
id)
    ->     ) ENGINE=InnoDB;
ERROR 1005 (HY000): Can't create table 'tienda.producto_imagen' (errno: 150)
mysql>
```

Ejemplo de crear una clave foránea con los datos incorrectos. En este caso el producto_id no es UNSIGNED.



Nota

Con la experiencia y realizando ejercicios se logrará la habilidad suficiente para utilizarlas con soltura.



Actividades

2. Asegúrese de que la tabla hotel que creó en el anterior ejercicio es de tipo InnoDB (use ALTER TABLE hotel ENGINE=InnoDB).
3. Cree la tabla “reserva”. Contiene un código de reserva (cadena de texto de 3 caracteres, clave primaria), un nombre completo de cliente, teléfono del cliente, e-mail del cliente, precio, fecha de inicio y fecha de fin. Coloque una clave foránea apuntando a la tabla “hotel”. De tal forma que cada

reserva se corresponda con un hotel.

Tenga en cuenta que se utilizarán estas tablas en todos los ejercicios posteriores en este capítulo, por lo que es conveniente realizarlos.

Eliminar y añadir relaciones a tablas existentes

Como siempre, es posible realizar tareas sobre tablas ya creadas. Pruebe a eliminar y volver a crear la clave foránea de fk_producto en la tabla producto_imagen:

```
/* Quitar clave foranea */
ALTER TABLE producto_imagen DROP FOREIGN KEY fk_producto;

/* Añadirla de nuevo */
ALTER TABLE producto_imagen ADD CONSTRAINT fk_producto
FOREIGN KEY (producto_id) REFERENCES producto(id);
```

Para borrar la clave foránea se utiliza `DROP FOREIGN KEY`. Para añadirla se utiliza `ADD CONSTRAINT` seguido de una sintaxis similar a la que se emplea al crear la tabla. Las mismas restricciones se aplican, no es posible crear una clave foránea que sea de un tipo de dato distinto.

3.3. Enlaces entre bases de datos

El concepto enlace entre bases de datos es utilizado mayoritariamente en *Oracle Database*. Consiste en compartir una tabla entre varias bases de datos.

Partiendo de una tabla que se encuentra en una base de datos (bbdd1, se desea que otra base de datos (bbdd2 pueda utilizarla también.

Para ello, en bbdd2 se crea un enlace a la bbdd1 para utilizar esa tabla. A partir de ese momento, en bbdd2 aparecerá esa tabla como si fuera suya. Cualquier modificación en ella se aplicará a ambas bases de datos.

Esto puede ser útil, por ejemplo, para una tabla de usuarios. Podría tener dos aplicaciones muy distintas entre sí pero se pretende que los usuarios registrados puedan utilizar ambas aplicaciones con la misma combinación de usuario y contraseña. Habría

una única tabla de usuarios que compartirían todas las bases de datos.

En *Oracle Database* se proporcionan métodos muy avanzados para esto. Por ejemplo, puede existir un enlace entre bases de datos que se encuentren en servidores distintos.

En *MySQL* **esto no es posible**. Tendría que indagarse una manera distinta de conseguir el mismo propósito de compartir tablas. Ambas bases de datos deberían estar en el mismo servidor.



Importante

El usuario que quiera utilizar el enlace de base de datos debe tener permiso de acceder a ambas bases de datos.

Para utilizar esa tabla, simplemente debería indicarse el nombre de la base de datos delante del nombre de la tabla seguido de un punto. En el apartado de consultas se tratará cómo seleccionar tablas que se encuentren en otra base de datos.

4. Gestión de registros en tablas

Ya se ha visto cómo definir las tablas de la base de datos. Ahora se pasará a conocer cómo **manipular sus datos**. Las operaciones posibles son la inserción, modificación y borrado de registros.

Para probar, se utilizará la tabla de “producto” y la de “producto_imagen” empleadas para explicar cómo crear tablas. Si no las creó en su momento, créelas ahora.

4.1. Inserción

La inserción de datos se utiliza para añadir registros a una tabla. La sintaxis es la siguiente:

```
INSERT INTO producto  
(id, título, precio)  
VALUES  
(1, 'Tabla de planchar básica', 12.99);
```

Se empieza con INSERT INTO seguido del nombre de la tabla en la que quieren añadir los registros. A continuación, entre paréntesis se **listan las columnas** de las cuales se quiere especificar su valor. Es posible que no se deseé especificar todas las columnas porque se use su valor por defecto. Por ejemplo, en este caso no se desea especificar la fecha de alta del producto porque se rellena automáticamente con el valor de CURRENT_TIMESTAMP.

Después se escribe la palabra VALUES y, entre paréntesis y separado por comas, se listan los valores que se le va a dar al registro. Los valores van en orden, si la primera columna especificada es “id”, el primer valor que se ponga irá a parar al id.



Importante

Si se especifica una columna pero luego no se le da valor, se recibirá un error.

Los valores de texto y de fecha deben aparecer entre comillas simples; los demás aparecen sin comillas (números, NULL, etc..). Si un valor de texto incluye comillas, deben escaparse con la barra invertida (\). Los valores de fecha deben estar escritos utilizando el formato que utilice el SGBD. En MySQL es algo así como 2012-01-01 01:02:02 (año, mes, día, hora, minutos y segundos).

Al hacer una inserción es posible utilizar funciones del sistema para asignar los valores. Más adelante se verán algunas de esas funciones al explicar cómo realizar consultas de datos.

Hay multitud de posibles errores y problemas al insertar datos. Por ejemplo, añadir NULL a una columna que no acepta nulos; fechas inválidas; cadenas de texto más largas de lo que permite el campo, etc. Ante esos casos cada SGBD actuará de una forma diferente. **MySQL** es bastante “relajado” al respecto y permite muchas incorrecciones; en vez de devolver un error es posible que deje introducir el registro pero genere una advertencia interna (la cual no afecta al cliente de la base de datos).

Hay una sintaxis extra que permite añadir varios registros de una vez, aumentando significativamente el rendimiento. Es la misma sintaxis pero con conjuntos de valores separados por coma. En el propio ejemplo se añadirá una cadena de caracteres con comillas, para ver cómo tratarlas:

```
INSERT INTO producto
(id, titulo, precio)
VALUES
(50, 'Libro de Christian O\Donnell', 6.99),
(77, 'Máquina de coser', 100),
(99, 'Funda para el móvil', NULL);
```



Consejo

Siempre que vaya a insertar más de un registro a la vez, utilice esta sintaxis, porque aumenta el rendimiento enormemente.

Si el precio es NULL, debe especificarse, no es posible dejar ese valor vacío. Si se especifican tres columnas, todos los registros deben tener tres valores.



Actividades

4. En la tabla “hotel”, inserte cuatro hoteles con los datos que desee.
5. En la tabla “reserva”, cree 10 reservas para esos hoteles. Que tres de los hoteles tengan reservas pero que uno de ellos no tenga ninguna reserva.
Se utilizarán estos datos de prueba en los siguientes ejercicios de consultas de SQL, por lo que debe asegurarse de que los datos son más o menos coherentes.

Inserción desde datos de otra tabla

Es posible que para insertar datos tengan que utilizarse datos de otras tablas. Esto es muy útil cuando se realizan tareas de mantenimiento o se quiere normalizar una tabla

que tenía una estructura no ideal. Para ello, se insertarán datos a través de una sentencia SELECT que será la que obtenga los datos. De momento se explicará de forma muy básica. En los siguientes apartados se estudiará mejor la manera de utilizar las sentencias SELECT.

La sintaxis es la siguiente:

```
INSERT INTO tabla  
(campo1, campo2, campo3)  
SELECT elemento1, elemento2, elemento3  
FROM otra_tabla;
```

En vez de utilizar VALUES, lo que se hace es añadir una consulta debajo. Primero se especifican los campos y la consulta debe tener los campos exactamente en el mismo orden. Los tipos de datos deben ser compatibles.

De esta manera insertarán en la tabla “tabla” los resultados de la consulta realizada.



Importante

No es necesario que sean idénticos, pero deben ofrecer la posibilidad de ser utilizados.

Esto es muy útil cuando se normalizan tablas o se realizan operaciones complejas. Es mucho más rápido insertar filas utilizando una SELECT que leer los datos y volverlos a enviar al SGBD con inserciones normales.

4.2. Modificación

La modificación de datos se utiliza para cambiar datos en registros existentes. Para ello se usa la sentencia UPDATE. La sintaxis es la siguiente:

```
UPDATE producto  
SET titulo='Máquina de coser SINGER', precio=120  
WHERE id=77;
```

Primero, se utilizará la palabra **UPDATE** seguida del nombre de la tabla que va a modificarse. A continuación, la palabra **SET** y la lista de modificaciones que se van a realizar, separadas por comas.

En este caso se modificará el valor de “**titulo**” y de “**precio**”. Para especificar los valores se deben seguir las mismas reglas que para la inserción de datos.

Por último, se utiliza la palabra **WHERE** para filtrar el o los registros que quieran modificarse. En este caso se modificará solo el producto con id 77. En el apartado de selección de datos se ampliará el uso del WHERE.

Es posible utilizar el propio valor actual del campo para calcular su nuevo valor. También es posible no utilizar cláusula WHERE, aunque siempre hay que tener cuidado de lo que se modifica. Por ejemplo, vea la siguiente consulta que se utiliza para subir un 10% el precio de todos los productos:

```
UPDATE producto  
SET precio=precio*1.10;
```



Nota

Es posible reutilizar el valor original para realizar operaciones sobre el mismo. Lo que no es posible es utilizar operadores comunes en otros lenguajes como **+=**, **-=**, **++** para aplicar cambios directamente. Siempre debe haber un signo igual seguido del nuevo valor.

4.3. Borrado

Para borrar filas se empleará una sintaxis muy parecida a la que se usa para modificarlas, pero sin utilizar la sección SET.

```
DELETE FROM producto WHERE id=99;
```

Primero, se escribe **DELETE FROM** seguido del nombre de la tabla y luego detrás del WHERE se listan las filtraciones que se precisen.

Si quiere vaciarse una tabla es posible, simplemente, dejar la consulta sin WHERE.

Borrar filas es una tarea delicada y siempre hay que tener cuidado de lo que se coloca en el WHERE. Conviene probar primero con una consulta SELECT que lo que se está borrando es correcto.

Un error típico es lanzar sentencias DELETE FROM, olvidando el WHERE. Si se hace sobre una tabla con datos reales, se habrá vaciado la tabla y seguramente no haya vuelta atrás.

Nunca hay que tener la **más mínima prisa** al escribir una sentencia SQL de borrado de filas.



Aplicación práctica

Imagine que se encuentra en una reunión con un cliente, el cual le está explicando cómo debe ser la aplicación que quiere que usted desarrolle. Le ha explicado los siguientes puntos:

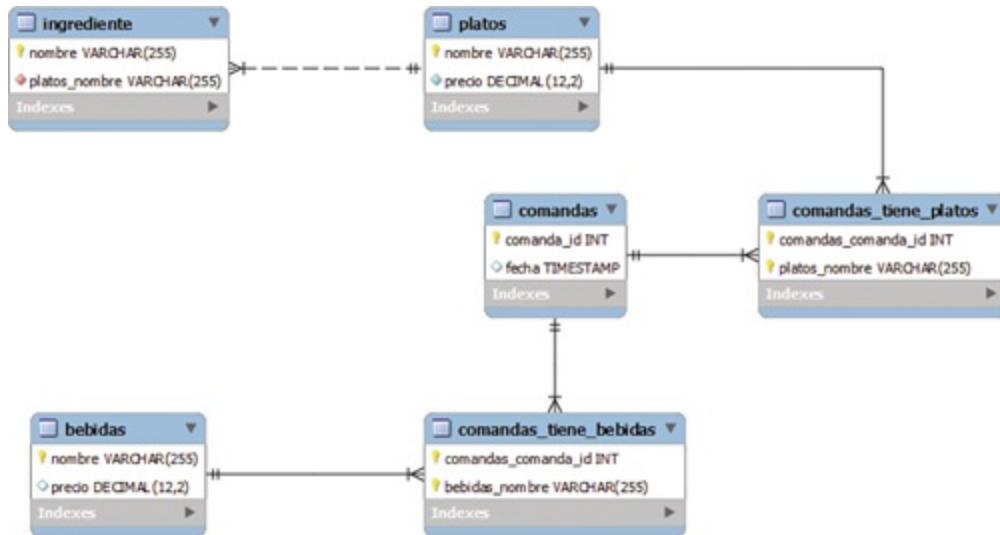
Existen platos (de comida), los cuales tienen un precio, un nombre y una serie de ingredientes (cada ingrediente tiene solo un nombre). También se desea guardar la fecha en la que el plato se introdujo en el sistema.

Existen bebidas, las cuales tienen un nombre y un precio.

Existen comandas, las cuales pueden tener varios platos y varias bebidas. Se debe guardar la fecha en que se produjo la comanda.

Cree el modelo de datos mediante el modelo entidad/relación, posteriormente normalice las tablas (si fuera necesario) y cree la base de datos en MySQL.

SOLUCIÓN



Todo el modelo de datos se encuentra normalizado. Las consultas para crear las tablas serían:

```

CREATE TABLE IF NOT EXISTS platos (
    nombre VARCHAR(255) NOT NULL ,
    precio DECIMAL(12,2) NOT NULL ,
    PRIMARY KEY (nombre)
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS ingrediente (
    nombre VARCHAR(255) NOT NULL ,
    platos_nombre VARCHAR(255) NOT NULL ,
    PRIMARY KEY (nombre) ,
    CONSTRAINT fk_ingredientes_platos1 FOREIGN KEY (platos_nombre) REFERENCES platos (nombre )
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS bebidas (
    nombre VARCHAR(255) NOT NULL ,
    precio DECIMAL(12,2) NULL ,
    PRIMARY KEY (nombre)
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS comandas (
    comanda_id INT NOT NULL ,
    fecha TIMESTAMP NULL ,
    PRIMARY KEY (comanda_id)
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS comandas_tiene_bebidas (
    comandas_comanda_id INT NOT NULL ,
    bebidas_nombre VARCHAR(255) NOT NULL ,
    PRIMARY KEY (comandas_comanda_id,bebidas_nombre) ,
    CONSTRAINT fk_comandas_has_bebidas_comandas1 FOREIGN KEY (comandas_comanda_id) REFERENCES comandas (comanda_id) ,
    CONSTRAINT fk_comandas_has_bebidas_bebidas1 FOREIGN KEY (bebidas_nombre) REFERENCES bebidas (nombre )
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS comandas_tiene_platos (
    comandas_comanda_id INT NOT NULL ,
    platos_nombre VARCHAR(255) NOT NULL ,
    PRIMARY KEY (comandas_comanda_id, platos_nombre) ,
    CONSTRAINT fk_comandas_has_platos_comandas1 FOREIGN KEY (comandas_comanda_id) REFERENCES comandas (comanda_id) ,
    CONSTRAINT fk_comandas_has_platos_platos1 FOREIGN KEY (platos_nombre) REFERENCES platos (nombre )
) ENGINE = InnoDB;
  
```

5. Consultas

Ahora comienza el apartado más amplio del uso de SQL: las consultas de datos. Hasta ahora se ha tratado cómo crear tablas y manipular sus datos y, a continuación, se estudiará la parte más interesante que es la de consultar esos datos de tal manera que puedan ser utilizados por la aplicación o informe que se esté desarrollando.

La idea general es que se lanza una consulta (que tiene una estructura parecida a los INSERT, UPDATE y DELETE) y el SGBD **devuelve una tabla con las columnas deseadas** y las filtraciones que se hayan especificado.



Importante

En caso de no devolver resultados, el SGBD o bien devuelve una tabla vacía o devuelve un mensaje diciendo que no hay resultados (dependiendo de la situación).



Actividades

6. Para aprender a realizar consultas se proponen dos tablas de marcas y modelos de coche. Están estudiadas para permitir realizar multitud de diferentes consultas, por lo que se puede aprender mucho con ellas:

```

CREATE TABLE marca (
    marca_id INT UNSIGNED PRIMARY KEY,
    marca_nombre VARCHAR(255) NOT NULL UNIQUE,
    marca_pais VARCHAR(50) NULL
) ENGINE=InnoDB;

CREATE TABLE modelo (
    modelo_id INT UNSIGNED PRIMARY KEY,
    marca_id INT UNSIGNED,
    modelo_nombre VARCHAR(255) NOT NULL,
    modelo_padre INT UNSIGNED NULL,
    modelo_precio DECIMAL(12, 2) NULL,
    modelo_fecha_lanzamiento DATE,
    CONSTRAINT fk_marca FOREIGN KEY (marca_id) REFERENCES marca(marca_id),
    CONSTRAINT fk_modelo_padre FOREIGN KEY (modelo_padre) REFERENCES modelo(modelo_id)
) ENGINE=InnoDB;

INSERT INTO marca
(marca_id, marca_nombre, marca_pais)
VALUES
(10, 'SEAT', 'España'),
(20, 'Renault', 'Francia'),
(30, 'Audi', 'Alemania'),
(31, 'Volkswagen', 'Alemania'),
(40, 'Ford', 'EEUU');

INSERT INTO modelo
(modelo_id, marca_id, modelo_nombre, modelo_padre, modelo_precio, modelo_fecha_lanzamiento)
VALUES
(1, 10, 'Ibiza', NULL, 15600, '2002-02-20'),
(2, 10, 'Cordoba', 1, 16000, '2002-04-01'),
(5, 20, 'Megane', NULL, 17500, '2008-09-10'),
(6, 20, 'Clio', NULL, 14000, '2012-05-23'),
(7, 20, 'Captur', 6, 15300, '2012-08-10'),
(30, 30, 'A3', NULL, 18000, '2013-07-01'),
(33, 30, 'R8', NULL, NULL, '2010-01-01');

```

Hay una tabla de marcas y otra de modelos. Las marcas tienen un ID (manual), un nombre y un país de origen.

Los modelos son de una marca, tienen un ID y un nombre. Además, pueden tener un modelo que sea el originario, por ejemplo, el Seat Córdoba procede del Seat Ibiza y el Renault Captur procede del Renault Clío. Se tiene su precio (que puede ser desconocido) y una fecha de lanzamiento.

Créalas a mano en su base de datos para poder seguir los ejemplos y ejercicios de esta sección.

5.1. Estructura general de una consulta

A continuación, se mostrará una consulta con bastantes elementos para después analizarla:

```
SELECT marca.marca_id, marca_nombre, COUNT(*) AS total_modelos
FROM marca
INNER JOIN modelo ON modelo.marca_id = marca.marca_id
WHERE marca_pais IS NOT NULL
GROUP BY marca.marca_id, marca_nombre
HAVING COUNT(*) >= 1
ORDER BY total_modelos DESC;
```



Nota

Además de los elementos aquí mostrados, *MySQL* cuenta con la sección LIMIT que sirve para limitar el número de filas devueltas. No es una palabra estándar. Se explicará en detalle más adelante.

Y estos son los resultados que se obtendrían:

The screenshot shows a Windows command-line window titled "mysql". The user has run the command "use tienda" to select a database. Then, they have run the previously shown SQL query to select data from the "marca" and "modelo" tables. The output shows three rows of data:

marca_id	marca_nombre	total_modelos
20	Renault	3
30	Audi	2
10	SEAT	2

Below the table, it says "3 rows in set <0.00 sec>".

Lo que se busca es una lista de marcas junto al número de modelos que tienen. Solo interesan modelos cuyo país tenga un valor y las marcas que tengan más de un modelo.

Simplemente se explicará la estructura de una consulta, entrando en detalle dentro de cada sección a través de los siguientes apartados.

Lo primero es la palabra **SELECT**, a continuación se listan las filas que se quiere

que el SGBD devuelva. Después hay que especificar **dónde** se encuentran esos datos. Para ello, se utiliza la palabra **FROM** y los **JOINS**. A continuación, en el WHERE se especifican las condiciones que deben cumplir las filas para ser procesadas o no. Después, se utiliza una agrupación de datos (GROUP BY) en la cual pueden exigirse condiciones extra con el HAVING. Por último, con ORDER BY se ordenan los resultados de la forma que se deseé.

Las consultas **siempre terminan con punto y coma**. La sección del SELECT es la única que es obligatoria (en *Oracle Database* el FROM también, pero en *MySQL* no).

Las consultas siempre se dividen en:

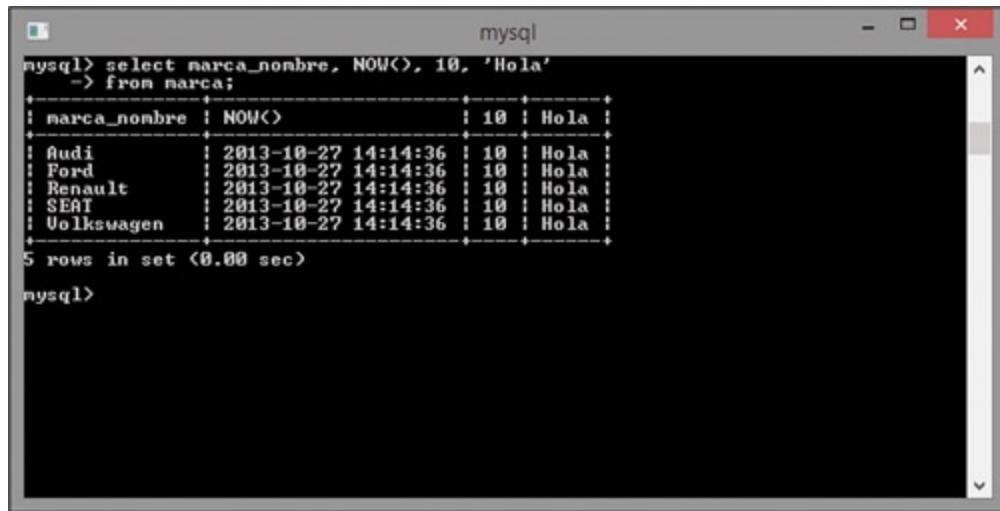
- Selección de las columnas
- Lista de tablas donde se encuentran esos datos
- Filtraciones sobre los registros
- Posible agrupación con posible condición
- Ordenación de los resultados

Como se aprecia en la imagen con los resultados, *MySQL* devuelve una tabla que contiene las filas deseadas. Cada cliente de la base de datos tratará esa “tabla virtual” de una forma. El cliente de *MySQL* por línea de comandos la muestra de una forma visual al usuario. Un cliente que se conecte mediante Java recibirá los datos en otro formato, por ejemplo.

El SGBD devuelve un objeto compuesto de filas y de columnas y cada cliente se encarga de hacer lo que quiera con él.

5.2. Selección de columnas. Obtención de valores únicos

Este apartado se centra en la parte que va desde el SELECT hasta el FROM. Aquí es donde se **seleccionan las columnas que se desea** que el SGBD devuelva. Para ello, pueden utilizarse campos de tablas, valores simples, funciones o subconsultas (se verán más adelante).



The screenshot shows a Windows command-line window titled "mysql". The command entered is:

```
mysql> select marca_nombre, NOW(), 10, 'Hola'  
      -> from marca;
```

The resulting table output is:

marca_nombre	NOW()	10	'Hola'
Audi	2013-10-27 14:14:36	10	'Hola'
Ford	2013-10-27 14:14:36	10	'Hola'
Renault	2013-10-27 14:14:36	10	'Hola'
SEAT	2013-10-27 14:14:36	10	'Hola'
Volkswagen	2013-10-27 14:14:36	10	'Hola'

5 rows in set <0.00 sec>

mysql>

Lo primero es ocuparse de un campo de la tabla marca y después, de una función del sistema (que devuelve la fecha y hora actuales). Los dos últimos campos son valores especificados manualmente.

Cada columna debe ir **separada por comas**. Si un campo seleccionado se encuentra en varias tablas (por ejemplo, varias tablas con un campo que se llame “id”) deberá añadirse el nombre de la tabla delante seguido de un punto (por ejemplo, marca.marca_id).

Lo habitual es seleccionar campos de tablas y funciones que realicen cambios sobre esos valores. Utilizar valores simples es útil a la hora de hacer un INSERT SELECT únicamente.



Nota

El INSERT SELECT es el tipo de inserción en el cual se utilizan los datos devueltos por una consulta SELECT.

Si se quieren devolver todas las filas disponibles, puede utilizarse un asterisco (*) en vez de listar las columnas. Para listar todos los campos de una tabla puede utilizarse nombre_tabla.*. Esto es útil en caso de utilizar uniones entre tablas, cuando se muestren datos de una sola tabla, pero es necesario unir varias tablas para filtrar los datos correctos.

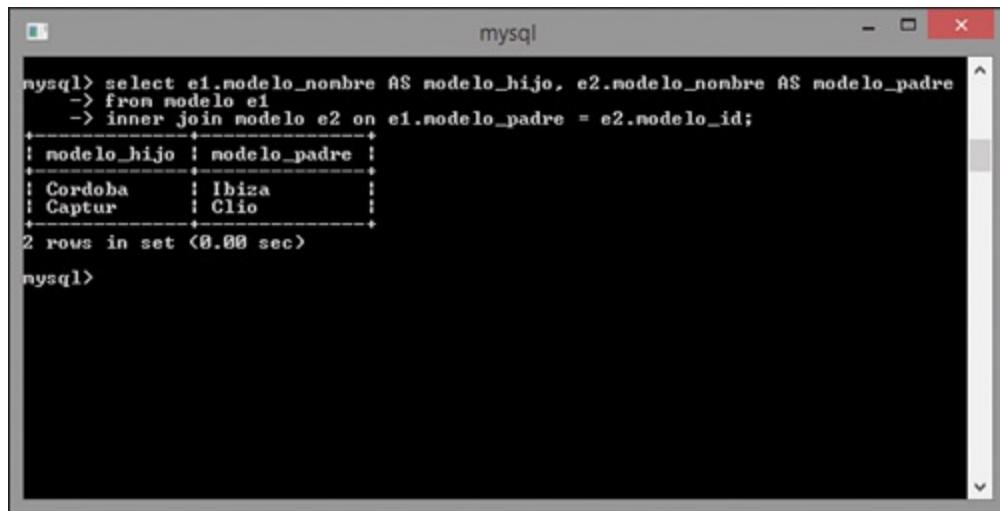


Consejo

Cuando vaya a escribir una consulta utilizada por una aplicación informática, evite utilizar los asteriscos. En caso de añadir más campos a las tablas involucradas, las aplicaciones estarían pidiendo más información de la que necesitan, disminuyendo su rendimiento.

Introducir alias a los nombres de las columnas

Es posible que lo que se desee es dar nombres personalizados a una columna en particular. Puede ser útil si el nombre es muy complejo (porque se utilicen muchas funciones, por ejemplo) o porque hay varias columnas con el mismo nombre y quieren diferenciarse. Para ello se utilizan los “alias”, que consisten en dar nombres personalizados. Por ejemplo, vea la siguiente consulta:



The screenshot shows a terminal window titled "mysql" displaying a SQL query. The query selects names from two tables, e1 and e2, using an inner join on their IDs. The results are displayed with aliases: "modelo_hijo" and "modelo_padre". The output shows two rows: one where "Cordoba" is the child of "Ibiza" and another where "Captur" is the child of "Clio".

```
mysql> select e1.modelo_nombre AS modelo_hijo, e2.modelo_nombre AS modelo_padre
-> from modelo e1
-> inner join modelo e2 on e1.modelo_padre = e2.modelo_id;
+ modelo_hijo : modelo_padre +
| Cordoba      : Ibiza          |
| Captur       : Clio           |
+-----+
2 rows in set (0.00 sec)

mysql>
```

Aquí se realiza una consulta bastante compleja porque es reflexiva, pero fíjese únicamente en la parte del SELECT. Aquí se dan nombres personalizados utilizando la palabra AS seguido del nombre personalizado.

Si se desea que el nombre personalizado tenga espacios o caracteres extraños habrá que especificar su nombre entre comillas simples. No es muy recomendable dar nombres extraños porque normalmente lo que se busca con los alias es simplificar los nombres.

Todos los campos seleccionados pueden tener alias. No importa que sean funciones, campos reales, subconsultas o datos fijos.



Actividades

7. De la tabla “hotel”, liste todos los nombres. El nombre de la columna devuelta debe ser “nombre”.

Obtención de valores únicos

SQL también permite filtrar registros únicos. Esto posibilita, por ejemplo, obtener una lista de países de la tabla “marca” en la cual “Alemania” no debería aparecer repetida. Para ello, se introducirá la palabra distinct justo después de SELECT:

mysql> select distinct marca_pais from marca;

marca_pais
España
Francia
Alemania
EEUU

4 rows in set <0.00 sec>

mysql>

Hay que tener en cuenta una serie de observaciones que causan confusión entre muchos usuarios inexpertos de SQL:

- Distinct actúa **después** de ejecutarse la consulta. El SGBD primero calcula los resultados y luego va uno a uno detectando duplicados y eliminándolos. Hay que tener en cuenta esto para el rendimiento de la consulta y para saber en qué momento se ejecuta el Distinct.
- Distinct elimina **tuplas duplicadas**, es decir, elimina registros que sean idénticos en todos los campos. En este caso solo había una columna. Si se hiciera SELECT DISTINCT marca_nombre, marca_pais, se obtendría toda la tabla porque no hay ninguna marca con el mismo nombre y mismo país repetido.
- Normalmente se utiliza de manera poco frecuente y para realizar informes. Si la aplicación requiere un uso intensivo del distinct es porque las tablas seguramente no estén normalizadas u optimizadas.

- Existe una variante llamada COUNT DISTINCT, es muy diferente a lo que se está viendo ahora y será tratada en profundidad en la sección de funciones de agrupación.

Limitar filas y paginación de resultados

En estos ejemplos se trabaja con tablas pequeñas, pero es habitual tener que tratar con tablas con cientos de registros o más. En ese caso es habitual contar con paginación, de tal manera que solo se vean los primeros 10 registros, por ejemplo, y se vaya poco a poco navegando por los demás.

Cada SGBD implementa su sistema para permitir esto. En *MySQL* esto se consigue mediante el LIMIT. LIMIT no es una palabra estándar SQL, hay que tenerlo en cuenta. Siempre se coloca al final de la consulta y puede recibir dos parámetros o uno:

```
SELECT *
FROM modelo
LIMIT 2;

SELECT *
FROM modelo
LIMIT 2, 5;
```

En la primera consulta, *MySQL* devuelve los dos primeros registros únicamente. En la segunda lo que hace es colocarse en la fila número 3 y devolver 5 registros. Hay que tener en cuenta que el número de fila empieza en el número cero.

La primera versión es útil cuando simplemente se quiere obtener una lista limitada de registros, para evitar ver cientos de ellos. La segunda es útil cuando se prefiere navegar a través de los resultados.



Consejo

La utilización de LIMIT es bastante eficiente en *MySQL*. Tan pronto como se devuelven los resultados pedidos, el SGBD para de procesar información, con el consiguiente aumento en el rendimiento.



Actividades

8. Suponiendo que el sistema de reservas muestra páginas de 5 resultados por cada página, escriba la consulta que habría que utilizar para listar la segunda página de resultados.

5.3. Selección de tablas. Enlaces entre tablas

Ahora es preciso ocuparse de la parte comprendida desde el FROM hasta el WHERE. Ahí es donde se especifica **dónde se encuentran los datos** que se van a utilizar en la consulta.

Después de haber pensado qué campos se necesitan, en esta sección hay que plantearse dónde están esos campos. Para ello se listarán en el FROM las tablas en que se encuentran.

En caso de haber varias tablas involucradas, deben utilizarse uniones de tablas (JOINS). En esta sección se verán uniones simples entre tablas, donde se especifican sus condiciones de manera sencilla.

He aquí un ejemplo de utilización del FROM:

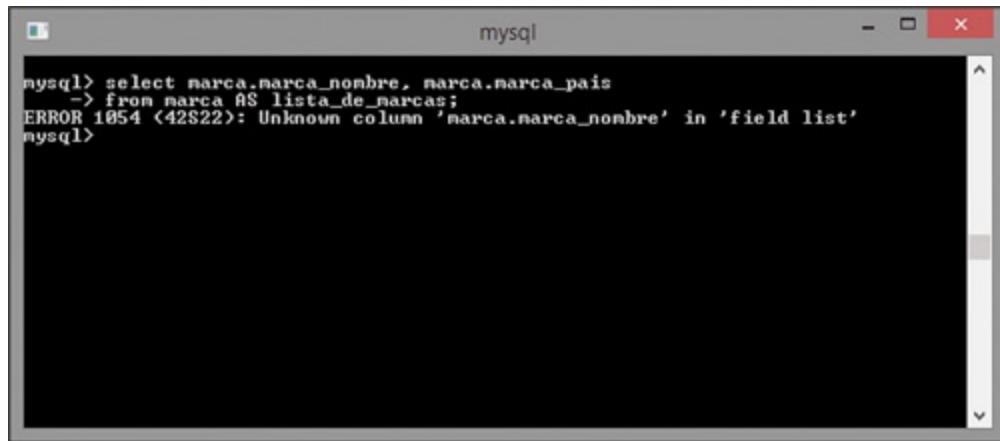
```
mysql> SELECT marca.marca_nombre, modelo.modelo_nombre
    -> FROM marca, modelo
    -> WHERE marca.marca_id = modelo.marca_id;
+-----+-----+
| marca_nombre | modelo_nombre |
+-----+-----+
| SEAT         | Ibiza          |
| SEAT         | Cordoba        |
| Renault      | Megane         |
| Renault      | Clio           |
| Renault      | Captur         |
| Audi          | A3              |
| Audi          | R8              |
+-----+-----+
7 rows in set (0.00 sec)

mysql>
```

Aquí se utilizan dos tablas: la de marcas y la de modelos. Es necesario utilizar un

WHERE para especificar la relación entre las tablas, de lo contrario se obtendría un resultado que no tendría sentido. En la sección de uniones entre tablas se ahondará en este aspecto.

En el FROM deben listarse las tablas involucradas separadas por comas. Pueden añadirse alias exactamente igual que en el SELECT. Serán útiles si los nombres de las tablas son largos o si se necesita realizar una consulta reflexiva en la que la misma tabla se utilice dos veces para poder diferenciarlas.



A screenshot of a Windows-style terminal window titled "mysql". The window contains the following text:
mysql> select marca.marca_nombre, marca.marca_pais
-> from marca AS lista_de_marcas;
ERROR 1054 <42S22>: Unknown column 'marca.marca_nombre' in 'field list'
mysql>



Importante

Si se le da un alias a una tabla ya no será posible referirse a esa tabla por su nombre original.

Aquí se da a “marca” el alias de “lista_de_marcas”, ya no resulta posible referirse a esa tabla como “marca” porque no está en la lista de tablas utilizadas.

En el FROM es posible, además de tablas, especificar subconsultas que contengan los datos a utilizar, esto se verá al llegar a la sección sobre las subconsultas.



Nota

Técnicamente no es necesario utilizar la palabra AS. Simplemente con especificar un nombre tras un espacio ya se aplica el alias. Aunque por cuestiones de legibilidad, es mucho mejor utilizar la palabra AS.



Actividades

-
9. Escriba una consulta que muestre el nombre del hotel, el nombre del cliente, la fecha de inicio de la reserva y la fecha de fin en una tabla.
-

Seleccionar tablas de otra base de datos

Es posible seleccionar tablas que se encuentren en una base de datos que no sea la actual. Para ello, delante del nombre de la tabla hay que indicar el nombre de la base de datos seguido de un punto. De esta manera, se indica la ruta a seguir para saber dónde se encuentra la tabla.

Por ejemplo, si estuviéramos en una base de datos distinta y se quisieran listar todas las marcas, se haría lo siguiente:

```
SELECT * FROM tienda.marca;
```

Simplemente se indica el nombre de la base de datos delante. Así es posible realizar **enlaces entre bases de datos** que se encuentren en el mismo servidor. Para que esto funcione, el usuario debe tener **permisos para acceder a ambas bases de datos**.

5.4. Condiciones. Funciones útiles en la definición de condiciones

Ahora es el turno de la parte del WHERE, posiblemente la más amplia en contenido. Tras la palabra WHERE se listará la **lista de condiciones** que deben cumplir los datos para ser devueltos por el SGBD.

Aquí puede utilizarse toda la lógica de condiciones habitual en cualquier lenguaje de programación, aunque siguiendo la sintaxis de SQL. Es posible utilizar condiciones AND y OR, además de utilizar paréntesis para agrupar esas condiciones.

Para mostrarlo, se enseña una consulta que liste los modelos que sean de España o Francia y además valgan entre 14.000 € y 16.000 € (ambos inclusive):

```
SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND (
    marca.marca_pais = 'España'
    OR marca.marca_pais = 'Francia'
)
AND modelo_precio >= 14000
AND modelo_precio <= 16000;
```



Consejo

Al mezclar cláusulas de filtración AND y OR, utilice paréntesis, de lo contrario la comprobación puede devolver resultados incorrectos. Puede utilizar todos los paréntesis que desee en una consulta.

Como se aprecia, pueden agruparse entre paréntesis las distintas condiciones. Utiliza el mismo método que cualquier lenguaje de programación, por lo que con un poco de experiencia se dominará sin problema.

Criterios de igualdad

Para determinar si un valor cumple una condición se utilizarán comparadores de igual ($=$, distinto (\neq), mayor que ($>$), menor que ($<$), mayor o igual que (\geq) y menor o igual que (\leq). Cabe destacar que el igual utiliza un único signo igual, a diferencia de muchos lenguajes de programación que utilizan un doble signo igual. Esto es así aquí porque no hay lugar a ambigüedades y un único signo es válido. El símbolo de distinto estándar es \neq , aunque la mayoría de SGBD también aceptan el común exclamación igual ($!=$).

Se debe utilizar AND y OR. No es posible utilizar `(&&` o `(||` como en otros lenguajes.

Utilidad BETWEEN

En ocasiones resulta útil comprobar si un valor se encuentra dentro de un **rango de valores**. Estos valores pueden ser numéricos, de fecha o de texto. Siguiendo un ejemplo parecido al anterior, podrían seleccionarse los modelos que valgan entre 14.000 € y 16.000 € de una manera más simplificada:

```
SELECT marca.marca_nombre, modelo.modelo_nombre  
FROM marca, modelo  
WHERE marca.marca_id = modelo.marca_id  
AND modelo_precio BETWEEN 14000 AND 16000;
```

Los valores que se incluyan en el BETWEEN son inclusivos. Es decir, un modelo con precio 14.000 € o un modelo con precio 16.000 € pasarán la condición.

Además de valores escritos manualmente puede utilizarse el valor de otras columnas ahí; incluso pueden realizarse subconsultas.



Actividades

-
10. Escriba una consulta que liste las reservas cuyo precio sea mayor que 40.
 11. Escriba una consulta que liste las reservas cuyo precio se encuentre entre 30 y 100.
-

Palabra NOT

Delante de una condición la palabra **NOT** invierte el resultado de esa condición de tal manera que si devolvía verdadero ahora devolverá falso y viceversa.



Ejemplo

Un NOT BETWEEN obligaría a que el valor no se encuentre dentro de ese rango de valores.

A continuación, se muestran ejemplos de uso de esta palabra con distintas funciones.

Utilidad IN

Esta es una simplificación cuando se quiere comprobar que un valor se encuentra dentro de una lista ya conocida de valores. Por ejemplo, obtener la lista de modelos que sean de España o Francia sería más simple de la siguiente manera:

```
SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND marca_pais IN ('España', 'Francia');
```

Dentro del IN se listan los posibles valores separados por comas. Si son cadenas de texto o fechas, deben estar entre comillas simples.

Es posible utilizar NOT IN() para devolver las filas que no cumplen esa condición. Por ejemplo, devolver todos los modelos que no sean de Francia ni de España.



Consejo

Las utilidades BETWEEN e IN simplifican mucho las consultas y las hacen mucho más legibles. Además, pueden llegar a ser más rápidas que escribir las condiciones a mano utilizando AND y OR.

Utilidad IS

A continuación, se estudiará más en profundidad el tratamiento de valores nulos, pero la función IS se utiliza para verificar si un valor es nulo o no.

La mayor peculiaridad es que la palabra **NOT** se suele incluir delante de la condición a evaluar excepto con la función IS, en la cual se añade después para que sea **más natural su lectura**.

Vea un ejemplo que muestra los modelos en los que el precio no sea nulo:

```
SELECT marca.marca_nombre, modelo.modelo_nombre  
FROM marca, modelo  
WHERE marca.marca_id = modelo.marca_id  
AND modelo_precio IS NOT NULL;
```

Utilizar `modelo_precio = NULL` sería incorrecto. No se obtendría ningún resultado. La evaluación daría falso siempre para todos los registros.

Utilidad LIKE

Para búsquedas no exactas de datos puede recurrirse a la utilidad LIKE. Sirve solamente para cadenas de texto, aunque *MySQL* las soporta para fechas también —lo cual se sale del estándar—.

Lo que hace LIKE es filtrar filas cuyos valores **contengan** lo que se quiere buscar. Su sintaxis es la siguiente:

```

SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND modelo_nombre LIKE '%a%';

SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND modelo_nombre LIKE '%a';

SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND modelo_nombre LIKE 'a%';

SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND modelo_nombre LIKE 'c__';

SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND modelo_nombre LIKE '%i_';

```

Entre comillas debe especificarse la condición a buscar. Si la búsqueda incluye una comilla habrá que escaparla.

El operador % significa “lo que sea en esa posición”. Desde cero a muchos caracteres. El operador _ significa “un carácter cualquiera en esa posición”. Debe haber al menos un carácter.



Consejo

Trate de dominar las búsquedas del tipo ‘%a%’ y ‘a%’ puesto que son las más comunes en el mundo real.

La explicación de cada consulta del ejemplo es la siguiente:

- Todos los modelos cuyo nombre contenga una “a”. Esto es así porque delante y detrás pueden tener lo que sea. La única condición es que haya una a, lo demás da igual.

- Todos los modelos cuyo nombre acabe en “a”. Esto es así porque delante puede haber lo que sea, pero la última letra debe ser una a, porque no hay nada después.
- Todos los modelos que empiecen por “a”. Es un caso similar al anterior.
- Todos los modelos cuyo nombre empiece por “c” y tengan cuatro letras. Esto es así porque la primera letra debe ser “c” y luego debe haber exactamente tres caracteres más, los que sean.
- Modelos que acaben con la letra “í” y cualquier otra letra. Solo el Renault Clío cumple esa condición.

Cabe destacar que *MySQL* normalmente **no distingue entre mayúsculas y minúsculas** en este tipo de comprobaciones a menos que se deseé explícitamente.

Es posible añadir la palabra **NOT** delante de **LIKE** para filtrar las filas que no cumplan esa condición.

Si la búsqueda incluye un guion bajo o un signo de porcentaje habrá que escaparlo. Esto se hace añadiendo un carácter de escape delante (la barra invertida).



Actividades

12. Quiere validarse que en el campo de correo electrónico de las reservas, el valor sea correcto. Para ello, escriba una consulta que liste las direcciones de e-mail que no contengan el carácter de arroba (@).
-

Utilización de funciones del sistema

Al igual que en el **SELECT**, en el **WHERE** es común utilizar **funciones del sistema** para realizar las comprobaciones. Las funciones disponibles las veremos en más detalle en la sección de funciones, simplemente mencionamos aquí que es posible utilizarlas sin problema.

Utilidades para filtrar resultados de subconsultas

Al utilizar subconsultas se pueden emplear más filtraciones como **EXISTS**, **ALL**, **ANY** y **SOME**, las cuales se tratarán en el apartado de subconsultas.

5.5. Significado y uso del valor null

El SGBD debe proporcionar un tratamiento específico y eficiente para los valores nulos.

Un valor nulo representa la ausencia de valor. Por ejemplo, no es lo mismo tener 0 puntos en el carnet de conducir que directamente no tener carnet de conducir. Uno contiene un valor vacío (0) y otro es una **ausencia de valor** (no tiene 0 puntos, simplemente no tiene carnet).

No es posible hacer una comprobación simple para ver si un campo contiene NULL. Por ejemplo, de estas dos consultas solo la segunda devuelve resultados:

```
SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND modelo_precio = NULL;

SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND modelo_precio IS NULL;
```

Una comprobación en la que uno de los dos extremos tenga el valor NULL siempre dará como resultado NULL, lo cual se evalúa como falso en el WHERE.

Para comprobar que un valor sea nulo es preciso usar la utilidad IS y IS NOT.

Existe una función útil para el manejo de valores nulos llamada IFNULL (campo, valor). Lo que hace esta función es comprobar si el campo contiene un valor nulo. En caso positivo devolverá el valor que se haya especificado como segundo parámetro. Si no es nulo utilizará su valor original:

```
SELECT marca.marca_nombre, modelo.modelo_nombre, IFNULL(modelo_precio, 'N/A') AS precio
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id;
```

Los valores nulos siempre se tratan de forma especial. Si se quisieran filtrar modelos cuyo precio fuera inferior a 16.000 € el registro con precio nulo no sería devuelto a menos que se pida específicamente.



Nota

Cualquier operación con un valor nulo devolverá NULL.

NULL + 30 => NULL.

NULL = NULL => NULL

NULL < 0 => NULL

Para poder tratar con los valores de una forma controlada hay que utilizar las funciones aquí mencionadas.

5.6. Ordenación del resultado de una consulta

La última parte de una consulta simple es la ordenación de los datos. El modelo relacional no se preocupa de en qué orden se guardan los datos en la base de datos. El SGBD no mantiene un orden específico y devuelve las filas conforme las va encontrando.

Esto es útil y válido desde el punto de vista del modelo relacional, pero para los seres humanos no lo es tanto, porque queremos ver los datos de una manera legible. Para ello, se utiliza la cláusula ORDER BY, la cual permite establecer el orden en que se quiere que los resultados sean devueltos.

El ORDER BY es lo último que debe especificarse en una consulta. Puede ordenarse por más de una columna, por ejemplo, ordenar primero por apellido 1, luego por apellido 2 y luego por nombre. Para ello, se especifica cada columna seguida del tipo de orden (ascendente o descendente):

```
SELECT marca.marca_nombre, modelo.modelo_nombre, modelo.precio
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
ORDER BY marca_nombre ASC, modelo.precio DESC;
```

ASC significa ascendente y DESC descendente. El orden por defecto es ASC. Se añade aquí para aumentar la legibilidad, pero la palabra **ASC** es en sí redundante.



Importante

Cabe destacar el tratamiento de los valores nulos. Al ordenar por una columna con nulos, los valores nulos siempre irán al final.



Aplicación práctica

Se encuentra desarrollando una aplicación informática que utiliza la siguiente tabla de nombre “ciudades” (mostramos sólo un ejemplo de los primeros registros de la misma):

ID	Nombre	CodigoPais	Distrito	Poblacion
1	Kabul	AFG	Kabol	1780000
2	Qandahar	AFG	Qandahar	237500
3	Herat	AFG	Herat	186800
4	Mazar-e-Sharif	AFG	Balkh	127800
5	Amsterdam	NLD	Noord-Holland	731200
6	Rotterdam	NLD	Zuid-Holland	593321
7	Haag	NLD	Zuid-Holland	440900
8	Utrecht	NLD	Utrecht	234323
9	Eindhoven	NLD	Noord-Brabant	201843

10	Tilburg	NLD	Noord-Brabant	193238
11	Groningen	NLD	Groningen	172701
12	Breda	NLD	Noord-Brabant	160398
13	Apeldoorn	NLD	Gelderland	153491
14	Nijmegen	NLD	Gelderland	152463
15	Enschede	NLD	Overijssel	149544
16	Haarlem	NLD	Noord-Holland	148772
17	Almere	NLD	Flevoland	142465
18	Arnhem	NLD	Gelderland	138020
19	Zaanstad	NLD	Noord-Holland	135621
20	's-Hertogenbosch	NLD	Noord-Brabant	129170
21	Amersfoort	NLD	Utrecht	126270
22	Maastricht	NLD	Limburg	122087
23	Dordrecht	NLD	Zuid-Holland	119811
24	Leiden	NLD	Zuid-Holland	117196
25	Haarlemmermeer	NLD	Noord-Holland	110722
26	Zoetermeer	NLD	Zuid-Holland	110214
27	Emmen	NLD	Drenthe	105853
28	Zwolle	NLD	Overijssel	105819
29	Ede	NLD	Gelderland	101574
30	Delft	NLD	Zuid-Holland	95268

Por necesidades del sistema, debe sacar listados de ciudades con los siguientes parámetros:

- Ciudades con una población entre 500.000 y un millón.
- Ciudades cuyo código de país empiece por con la letra “A”.
- Mostrar las 10 ciudades con más población.
- Mostrar las ciudades cuyo código de país sea “ESP” y tengan menos de 100.000 habitantes.

SOLUCIÓN

```
SELECT *
FROM ciudades
WHERE Poblacion BETWEEN 500000 AND 1000000;

SELECT *
FROM ciudades
WHERECodigoPais LIKE 'A%';

SELECT *
FROM ciudades
ORDER BY Poblacion DESC
LIMIT 10;

SELECT *
FROM ciudades
WHERECodigoPais = 'ESP'
AND Poblacion < 100000;
```



Actividades

-
13. Escriba una consulta que muestre las reservas por orden de precio: el más caro primero y el más barato después. En caso de tener igual precio, se ordenará alfabéticamente por el nombre del cliente.
-

6. Conversión, generación y manipulación de datos

Todos los SGBD incorporan una serie de funciones que facilitan la manipulación y consulta de los datos de la base de datos.



Nota

La lista de funciones que trae el SQL estándar es muy limitada y cada SGBD las interpreta de una manera a veces distinta.

Aquí se verán las funciones del SQL estándar más habituales y además, se tratarán algunas funciones exclusivas de *MySQL*; siempre especificando cuáles son estándar y cuáles no.

Como en cualquier lenguaje de programación, las funciones son componentes informáticos que **reciben** una cantidad de **parámetros** X y que **devuelven un resultado**. Aquí la sintaxis es la siguiente: SUBSTR (modelo_pais, 1, 2). Substr es el nombre de la función, entre paréntesis y separados por comas van los parámetros. Es posible que una función no reciba parámetros, en ese caso se abren y cierran paréntesis, por ejemplo: NOW () .

6.1. Funciones para la manipulación de cadenas de caracteres

A continuación, se explicarán las funciones más habituales utilizadas para manipular cadenas de caracteres (texto).

Concatenación

El SQL estándar especifica que debe ser posible contar con una manera de concatenar cadenas de texto (unir varias cadenas en una sola). No obstante, cada SGBD utiliza un método diferente para conseguir el mismo resultado.

En *MySQL* se cuenta con la función CONCAT(), la cual recibe un número arbitrario de parámetros (desde dos hasta el infinito) y devuelve una cadena de texto única formada por todos sus parámetros.

```
SELECT CONCAT(marca.marca_nombre, ' ', modelo.modelo_nombre)
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id;
```

En esta consulta se obtiene en una sola columna la marca y el modelo, separados por un espacio.

Función LOWER y UPPER

Estas funciones son muy sencillas y comunes a todos los SGBD. Lo que hacen es convertir una cadena a todo minúsculas (LOWER) o todo mayúsculas (UPPER).

Función SUBSTRING

Sirve para cortar cadenas grandes de texto, algo útil cuando solo interesa quedarse con una parte de la misma.

La sintaxis estándar utiliza FROM y FOR. MySQL, además, soporta otra sintaxis más sencilla en la que se suprimen esas palabras.

La idea es seleccionar el campo a utilizar, donde empezar la nueva cadena y el número de caracteres a devolver (todos si no se especifica). El lugar donde empezar a cortar es un valor numérico que **comienza en 1** y va aumentando a medida que se avanza a través de la cadena.

A continuación, se muestran las dos sintaxis. Las dos primeras consultas hacen lo mismo y las dos últimas, también.

```
SELECT SUBSTRING(marca_nombre FROM 2)
FROM marca;

SELECT SUBSTRING(marca_nombre, 2)
FROM marca;

SELECT SUBSTRING(marca_nombre FROM 1 FOR 3)
FROM marca;

SELECT SUBSTRING(marca_nombre, 1, 3)
FROM marca;
```

En las dos primeras se devuelve el nombre de la marca quitando el primer carácter. El 2 es donde se empieza, que es en el segundo carácter (porque el primero no hace falta).

En las dos segundas se obtienen las primeras tres letras del nombre de la marca. Para ello, se empieza por el primer carácter (1 y luego se obtienen 3 caracteres).

Función TRIM

Cuando los datos provienen de los usuarios es muy común que contengan espacios o

saltos de línea al principio o al final de las cadenas de texto. Esto es así bien porque el propio usuario no se da cuenta (no es visible para ellos), bien porque el propio sistema añade un salto de línea (en línea de comandos es algo habitual). También es muy típico que al copiar y pegar texto de un lugar a otro, se cuele el espacio de la palabra anterior o siguiente.

Para ello, se recurre a la función TRIM, que recibe una cadena de texto y elimina espacios y saltos de línea al principio y final de la misma.



Importante

MySQL proporciona otras funciones llamadas LTRIM y RTRIM que eliminan solamente los espacios a la izquierda o derecha respectivamente.

Función LENGTH

La función LENGTH recibe una cadena de texto como parámetro y devuelve un número entero, indicando el número de caracteres que la componen.

Función REPLACE

En cada SGBD tiene una sintaxis distinta y en algunos ni siquiera está disponible. En MySQL se trata de una función que recibe una cadena de texto, un valor a buscar y un valor de reemplazo:

```
SELECT REPLACE(modelo_nombre, 'o', 'a')  
FROM modelo;
```

Aquí se reemplazan todas a las “oes” por “aes”.

Cabe destacar que en esta función que el valor esté en mayúsculas o minúsculas sí afecta, algo que no ocurre al hacer una comprobación en un LIKE o un igual.



Consejo

Todas las funciones de cadenas de texto que se han visto son muy utilizadas y es importante dominarlas. Son realmente útiles.



Aplicación práctica

Partiendo de la siguiente tabla de nombre “clientes”, ejecute las siguientes consultas:

Nombre	Apellidos
JOSÉ MARÍA	PÉREZ SALAS
ALBERTO	Gómez Pérez
Maria	Sánchez Menéndez

Consulta que muestre todos los nombres y apellidos en mayúscula.

- 1. Consulta que muestre al cliente cuyo nombre ocupe más caracteres.**
- 2. Los nombres y los apellidos vienen directamente desde un formulario web. Por lo que pueden contener espacios donde no debieran. Muestre los nombres y apellidos sin espacios extra.**

SOLUCIÓN

```
SELECT UPPER(Nombre) AS Nombre, UPPER(Apellidos) AS Apellidos  
FROM clientes;  
  
SELECT * FROM clientes ORDER BY LENGTH(Nombre) DESC LIMIT 1;  
  
SELECT TRIM(Nombre) AS Nombre, TRIM(Apellidos) AS Apellidos  
FROM clientes;
```



Actividades

14. Escriba una consulta que muestre una lista de reservas. Se quiere devolver una tabla compuesta por el id de la reserva y otra columna con el nombre del cliente, pero en mayúsculas y solo los primeros 8 caracteres.
15. Escriba una consulta que devuelva la lista de nombres de hoteles. Los nombres deben estar en minúscula y no debe haber espacios. Estos deben ser reemplazados por guiones bajos.

6.2. Funciones para la manipulación de números

Existen muchas funciones matemáticas para trabajar con números. Hay tantas que aquí se tratarán las más comunes.

Si fuera necesario realizar operaciones matemáticas más complicadas siempre puede consultarse el manual del SGBD para conocerlas todas. La mayoría reciben un número o dos y devuelven otro número.



Consejo

ABS es muy útil para ordenar por proximidad de un valor numérico. Por ejemplo, para ver coches con el precio más cercano a 14.000 €, se haría ORDER BY ABS(14000 – modelo_precio) ASC; así se obtendrían los modelos ordenados por diferencia de precio, tanto por arriba como por abajo.

Función	Ejemplo	Resultado ejemplo	Explicación
ABS (numero)	ABS(12-13)	1	Devuelve el valor absoluto de un número. Convierte negativo en positivo.
			Realiza un

CEIL/CEILING (numero_decimal)	CEIL(12.01)	13	redondeo hacia arriba. Devuelve el número entero más cercano sin ser inferior.
FLOOR (numero_decimal)	FLOOR(12.99)	12	Redondea hacia abajo. Devuelve el número entero más cercano sin ser mayor.
GREATEST (num1, num2)	GREATEST(10,20)	20	Devuelve el valor mayor recibido por parámetros.
LEAST (num1, num2)	LEAST(10, 20)	10	Devuelve el menor número recibido por parámetros.
MOD (num1, num2)	MOD(3, 2)	1	Devuelve el resto al dividir num1 entre num2.
ROUND (num, precisión)	ROUND(1.30) ROUND(1.298, 1) ROUND(24.93, -1)	1 1.3 20	Redondea al número más cercano. Si se da un segundo parámetro redondea hasta X decimales. Si el segundo parámetro es negativo redondea X dígitos hacia la izquierda.



Actividades

16. Escriba una consulta que muestre las reservas cuyo precio se aproxime más a 40. Muestre las 3 reservas que más se acerquen.
 17. Escriba una consulta que aplique un 10 % de descuento sobre el precio pagado. No se desean decimales. Si los hay, que se redondee hacia abajo.
-



Aplicación práctica

Partiendo de la siguiente tabla de nombre “productos”, escriba las siguientes consultas SQL:

Producto	Precio
Cortacésped	202.50
Sierra eléctrica	159.99
Motosierra	252.40

1. Consulta que devuelva el producto cuyo precio se acerque más a 210.
2. Consulta que devuelva los productos con su precio redondeado a la cifra entera más cercana.
3. Consulta que devuelva los productos cuyo precio sea divisible entre dos.

SOLUCIÓN

```
SELECT * FROM productos ORDER BY ABS(Precio - 210) ASC LIMIT 1;  
SELECT Producto, ROUND(Precio) FROM productos;  
SELECT * FROM productos WHERE MOD(Precio, 2) = 0;
```

6.3. Funciones de fecha y hora

Tratar con fechas siempre es algo engorroso. Afortunadamente, SQL proporciona algunas funciones estándar que facilitan mucho la vida.

Las fechas siempre se devuelven y reciben en el formato de fechas por defecto del SGBD. En MySQL es YYYY-MM-DD HH:MM:SS (año, mes, día hora, minuto y segundo) por defecto (aunque se puede cambiar).

Obtener fecha y hora actuales

Hay muchas funciones para esto y muchas de ellas están duplicadas (tienen el mismo comportamiento pero distinto nombre), por lo que se mostrará un ejemplo de cada tipo:

- CURDATE() devuelve la fecha actual (sin la hora).
- CURTIME() devuelve la hora actual (sin fecha).
- NOW() devuelve la fecha y hora actuales.

Funciones que devuelven los componentes de una fecha/hora

Se cuenta con una función diferente para cada componente de una fecha (SECOND, MINUTE, HOUR, DAY, MONTH, YEAR). Reciben una fecha como parámetro y devuelven un número entero con su valor. Resulta útil, por ejemplo, para filtrar por registros introducidos durante este año.

Es posible anidar llamadas a funciones, por lo que se podría obtener la lista de modelos lanzados este mismo año de la siguiente manera:

```
SELECT marca.marca_nombre, modelo.modelo_nombre, modelo.modelo_fecha_lanzamiento
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id
AND YEAR(modelo_year_lanzamiento) = YEAR(NOW());
```

Están fuera del estándar, pero *MySQL* proporciona, además, las funciones WEEK y WEEKDAY. Dependiendo de la configuración del sistema, WEEKDAY devolverá un número del 1 al 7 o del 0 al 6. Dependiendo también de la configuración, WEEK devolverá un número entre el 0 y el 53 o del 1 al 53.



Importante

WEEKDAY retorna el número de día dentro de la semana de una fecha. WEEK retorna el número de la semana en que se encuentra una fecha.

Funciones para calcular fechas

Es muy común la generación de fechas dinámicamente. Por ejemplo, para realizar una aplicación que muestre una gráfica con la ocupación de un hotel para la semana pasada. Cada día que pase, la fecha cambiará, pero los requisitos de la aplicación serán siempre los mismos: mostrar datos de la semana pasada.

Para eso existen funciones que realizan cálculos de fecha. Casi ninguna está en el estándar de SQL, así que se explicarán las aplicables a *MySQL* directamente.

DATEDIFF()

Devuelve el número de días que hay entre dos fechas. Recibe dos fechas por parámetros y devuelve un número entero.

DATE_ADD()

Dada una fecha, añade un intervalo de tiempo a la misma. Puede sumar o restar tiempo, según las necesidades. Vea un caso de uso:

```
SELECT * FROM productos ORDER BY ABS(Precio - 210) ASC LIMIT 1;  
SELECT Producto, ROUND(Precio) FROM productos;  
SELECT * FROM productos WHERE MOD(Precio, 2) = 0;
```

Aquí interesa ver los modelos que se lanzaron en los últimos 4 años —sin importar en qué año nos encontramos actualmente—. Para eso se suman 4 años a la fecha de lanzamiento. Si sigue siendo superior a la actual, querrá decir que es menor a 4 años.

La sintaxis es la siguiente: el primer parámetro es la fecha que quiere manipularse. El segundo parámetro empieza con la palabra INTERVAL seguida de un número (positivo o negativo) seguido del nombre de intervalo a añadir (las opciones son MICROSECOND, SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR). Esta palabra debe ir en singular siempre.

FROM_UNIXTIME()

Con esto se utiliza un *timestamp* de UNIX como valor de fecha. Los *timestamps* de UNIX son números enteros que representan el número de segundos

transcurridos desde el 1 de enero de 1970.



Importante

Es muy común su utilización en el desarrollo de aplicaciones informáticas. Esta función recibe un número entero y devuelve una fecha válida.

UNIX_TIMESTAMP()

Es la función contraria a la anterior. Recibe una fecha válida y devuelve un número entero con el *timestamp* de UNIX.

Funciones de formateo de fechas

MySQL utiliza su formato por defecto para mostrar fechas. Este formato por defecto es muy bueno porque permite **ordenar alfabéticamente las fechas** de forma que queden ordenadas también en el tiempo. Es bueno para un programador o para una máquina, pero a ningún usuario le gusta ver una fecha así: por ejemplo, 2013-02-03 23:49:44.

Para ello se cuenta con las funciones que permiten formatear las fechas cómodamente.

DATE_FORMAT()

Recibe una fecha y un parámetro de formato y devuelve una cadena de texto con la fecha formateada. El formato es una cadena de texto con una serie de códigos que son reemplazados por su valor. Vea la lista de variables disponibles:

Variable	Descripción
%a	Día de semana abreviado (Sun..Sat)
%b	Mes abreviado (Jan..Dec)
%c	Mes, numérico (0..12)
%D	Día del mes con sufijo inglés (0th, 1st, 2nd, 3rd, ...)
%d	Día del mes numérico (00..31)

%e	Día del mes numérico (0..31)
%f	Microsegundos (000000..999999)
%H	Hora (00..23)
%h	Hora (01..12)
%I	Hora (01..12)
%i	Minutos, numérico (00..59)
%j	Día del año (001..366)
%k	Hora (0..23)
%l	Hora (1..12)
%M	Nombre mes (January..December)
%m	Mes, numérico (00..12)
%p	AM o PM
%r	Hora, 12 horas (hh:mm:ss seguido de AM o PM)
%S	Segundos (00..59)
%s	Segundos (00..59)
%T	Hora, 24 horas (hh:mm:ss)
%U	Semana (00..53), donde domingo es el primer día de la semana
%u	Semana (00..53), donde lunes es el primer día de la semana
%V	Semana (01..53), donde domingo es el primer día de la semana; usado con %X
%v	Semana (01..53), donde lunes es el primer día de la semana; usado con %x
%W	Nombre día semana (Sunday..Saturday)
%w	Día de la semana (0=Sunday..6=Saturday)
%X	Año para la semana donde domingo es el primer día de la semana, numérico, cuatro dígitos; usado con %V
%x	Año para la semana, donde lunes es el primer día de la semana, numérico, cuatro dígitos; usado con %v
%Y	Año, numérico, cuatro dígitos
%y	Año, numérico (dos dígitos)
%%	Carácter '%' literal



Consejo

Esta función es bastante útil y potente, aunque lo habitual es tratar con las fechas en su formato original en cada aplicación de la base de datos. Sobre todo si se trata de una aplicación multi-idioma, porque *MySQL* siempre devolverá las fechas en inglés. Lo normal es hacer una consulta SQL con el formato estándar y aplicar los cambios en el cliente.

Los nombres de día y mes están en inglés. Si es necesaria una aplicación multi-idioma tal vez convenga más procesar la fecha en el cliente de la base de datos.

Ejemplo de cómo mostrar la fecha 2012-10-03 20:04:59 como 3/10/2012:

```
mysql> SELECT DATE_FORMAT('2012-10-03 20:04:59', '%e/%c/%Y');
+ DATE_FORMAT('2012-10-03 20:04:59', '%e/%c/%Y') +
| 3/10/2012
+-----+
| 1 row in set (0.00 sec)

mysql>
```

Es simplemente cuestión de jugar con los posibles formatos para ver todo lo que puede hacerse con esta función.

STR_TO_DATE

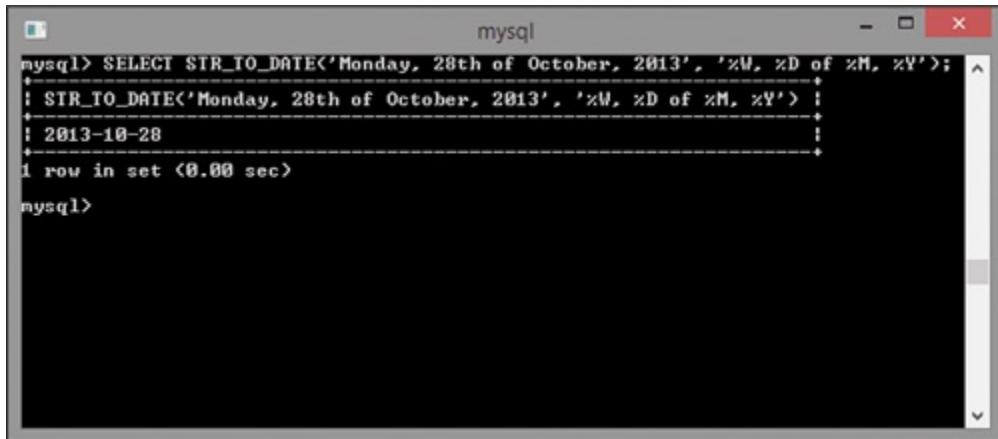
Esta función es completamente inversa a DATE_FORMAT(). Recibe lo que DATE_FORMAT() devuelve y devuelve lo que DATE_FORMAT() recibe.



Importante

Su función es recibir una fecha formateada de forma especial y convertirla en una fecha *MySQL* válida.

Conversión de “Monday, 28th of October, 2013” a MySQL:



```
mysql> SELECT STR_TO_DATE('Monday, 28th of October, 2013', '%W, %D of %M, %Y');
+-----+
| 2013-10-28 |
+-----+
1 row in set (0.00 sec)

mysql>
```

STR_TO_DATE devuelve una fecha/hora si el formato especifica horas, minutos o segundos; de lo contrario, devuelve una fecha simple.

Tanto para esta función como para DATE_FORMAT resulta muy útil tener la tabla de formatos delante, así simplemente hay que reemplazar los valores con variables.



Actividades

18. Escriba una consulta que muestre las reservas que empiezan mañana.
 19. Escriba una consulta que devuelva la lista de reservas pero cuyo formato de fechas sea día/mes/año.
-



Aplicación práctica

Partiendo de la siguiente tabla de nombre empleados:

Empleado	Fecha_alta
José	2013-01-01 17:40:30
Maria	2010-03-10 16:00:01

Rubén	2008-10-30 13:59:59
Lucía	2004-05-15 10:05:39

Escriba las consultas SQL que permitirían hacer las siguientes operaciones:

1. Listar empleados que se dieron de alta hace menos de tres años.
2. Listar el nombre del empleado junto al número de años que lleva en la empresa.
3. Listar empleados junto a la fecha de alta en el siguiente formato: 3 de octubre de 2010, suponiendo que MySQL devuelve el número de meses en español.

SOLUCIÓN

```
SELECT * FROM empleados WHERE Fecha_alta > DATE_ADD(NOW(), INTERVAL -3 YEAR);
SELECT Empleado, FLOOR(DATEDIFF(NOW(), Fecha_alta) / 365) FROM empleados;
SELECT Empleado, DATE_FORMAT(Fecha_alta, '%e de %M de %Y') FROM empleados;
```

6.4. Funciones de conversión de datos

Para convertir un dato de un tipo a otro se cuenta con la función CAST(). Esto es útil, por ejemplo, para extraer la parte de hora de un dato DATETIME convirtiéndolo en tipo TIME. O convertir un número UNSIGNED en un número con signo si hace falta realizar operaciones matemáticas con él que podrían dar como resultado un número negativo. Vea un par de ejemplos:

```
mysql> SELECT CAST('2010-01-01 20:00:00' AS TIME);
+-----+
| 20:00:00 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT CAST(marca_id AS SIGNED) - 20 FROM marca;
+-----+
| 10   |
| 20   |
| 0    |
| -10  |
| 11   |
+-----+
5 rows in set (0.00 sec)

mysql> SELECT marca_id - 20 FROM marca;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '<'tienda'.`marca`.`marca_id` - 20'
mysql>
```

En la primera consulta se convierte un DATETIME a TIME. En la segunda, un UNSIGNED a SIGNED. Así puede operarse con números sin signo. En la tercera, como marca_id contiene números sin signo y la resta da un valor negativo, se obtiene un error, porque no es posible realizar la operación.

7. Consultas múltiples. Uniones (joins)

La principal característica del modelo relacional es que diferentes **entidades** se **relacionan entre sí**. Gracias al SQL pueden realizarse uniones entre tablas para obtener, **con una sola consulta**, datos que se encuentran en **varias tablas**.

En los ejemplos anteriores ya se han utilizado bastantes uniones de tablas. Se unía la tabla de marcas con la de modelos para poder obtener el nombre de la marca en los resultados.

Siempre, en la medida de lo posible, hay que tratar de obtener la mayor cantidad de información posible con el **menor número de consultas** de SQL posible.

Un ejemplo de unión de tablas es el siguiente:

```
SELECT marca_nombre, modelo_nombre
FROM marca, modelo
WHERE marca.marca_id = modelo.marca_id;
```

Como se aprecia, se seleccionan datos de las tablas de marcas y modelos. En el WHERE se especifica la cláusula de unión para explicar mediante SQL a qué marca corresponde cada modelo.

Sin utilizar uniones podría haberse seleccionado marca_id y modelo_nombre de la tabla de modelos y, mediante el cliente de la base de datos, realizar un bucle que obtuviera el nombre para cada marca.



Importante

Es mejor utilizar una unión porque el SGBD lo hará de una manera mucho más eficiente.

7.1. Definición de producto cartesiano aplicado a tablas

Un producto cartesiano es un **concepto algebraico**. El producto cartesiano de dos relaciones consiste en una nueva relación con todas las **combinaciones de valores** posibles. Se representa mediante el símbolo \times (no es exactamente una equis).

Por ejemplo, el producto cartesiano de:

$$A = \{1, 2\}$$

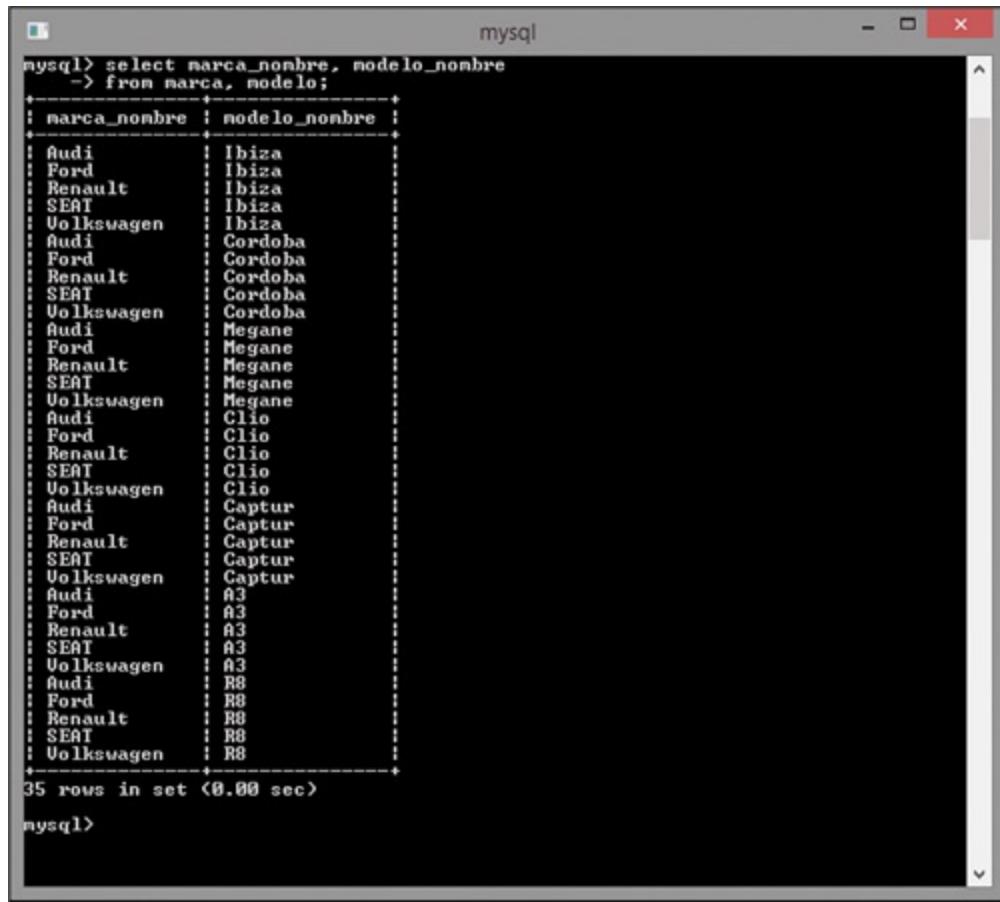
$$B = \{3, 4\}$$

$$A \times B = \{(1, 3), (1, 4), (2, 3), (2, 4)\}$$

Como ve, simplemente consiste en una nueva relación compuesta por todas las combinaciones entre A y B posibles.

El modelo relacional mantiene que los datos se almacenan en relaciones. Las relaciones son tablas en el SGBD. Por lo tanto, es posible realizar el producto cartesiano entre tablas.

Todo el motor de uniones entre tablas se basa en este concepto. A continuación, se muestra un ejemplo práctico de producto cartesiano entre tablas:



The screenshot shows a MySQL command-line interface window. The command entered was:

```
mysql> select marca_nombre, modelo_nombre  
-> from marca, modelo;
```

The resulting table has two columns: `marca_nombre` and `modelo_nombre`. The data consists of 35 rows, representing all combinations of car brands and models. The data is as follows:

marca_nombre	modelo_nombre
Audi	Ibiza
Ford	Ibiza
Renault	Ibiza
SEAT	Ibiza
Volkswagen	Ibiza
Audi	Cordoba
Ford	Cordoba
Renault	Cordoba
SEAT	Cordoba
Volkswagen	Cordoba
Audi	Megane
Ford	Megane
Renault	Megane
SEAT	Megane
Volkswagen	Megane
Audi	Clio
Ford	Clio
Renault	Clio
SEAT	Clio
Volkswagen	Clio
Audi	Captur
Ford	Captur
Renault	Captur
SEAT	Captur
Volkswagen	Captur
Audi	A3
Ford	A3
Renault	A3
SEAT	A3
Volkswagen	A3
Audi	R8
Ford	R8
Renault	R8
SEAT	R8
Volkswagen	R8

At the bottom of the screen, the message "35 rows in set <0.00 sec>" is displayed, followed by the MySQL prompt "mysql>".

Aquí se obtienen todas las combinaciones posibles entre marcas y modelos. El resultado no tiene sentido real. Sin embargo, si en el WHERE se especifica que `marca_id` de la tabla “marca” sea igual a `marca_id` de la tabla “modelo” sí se obtendrán los resultados correctos. **Primero se realizaría el producto cartesiano y luego se filtraría para hacer que los resultados concuerden.**

Por lo general, se aplica el producto cartesiano junto a una condición en el WHERE que aporte sentido a la relación entre las tablas. No obstante, a veces puede llegar a ser útil realizar el producto cartesiano puro y duro.



Nota

Puede ser útil cuando se quieran obtener las combinaciones de valores entre dos tablas.

7.2. Uniones de tablas (joins). Tipos: inner, outer, self, equi, etc.

Ya se ha visto en qué consiste el producto cartesiano. Sirve para obtener combinaciones de valores. Junto a unas cláusulas en el WHERE pueden filtrarse los resultados de forma que tengan más sentido. Esta es una manera de realizar uniones entre tablas muy común y más eficiente a veces. No obstante, SQL proporciona una sintaxis específica y ampliada para realizar uniones.



Consejo

Casi siempre que se le pida que en una sola consulta se obtengan datos de varias tablas, necesitará hacer un JOIN. Primero debe identificar las tablas que harán falta y, a continuación, cómo relacionar los registros de todas esas tablas de forma que los resultados sean coherentes.

Los tipos de uniones por separado se mostrarán a continuación.

INNER JOIN

Este es el tipo de unión básica. Su sintaxis es la siguiente:

```
SELECT marca_nombre, modelo_nombre  
FROM marca  
INNER JOIN modelo ON marca.marca_id = modelo.marca_id;
```

Detrás del FROM se añaden las palabras **INNER JOIN** (sin comas, a continuación el nombre de la tabla a unir. En el ON (obligatorio se especifica la **condición de unión**.

Como se puede ver, es muy parecido al ejemplo utilizado al realizar el producto cartesiano, pero moviendo la condición al ON en vez de al WHERE.

Es posible unir más de dos tablas. El número de tablas a unir es infinito. Cree la siguiente tabla para así poder practicar uniones entre más de dos tablas:

```

CREATE TABLE coche (
    coche_id INT UNSIGNED PRIMARY KEY,
    modelo_id INT UNSIGNED,
    coche_matricula VARCHAR(10) NOT NULL UNIQUE,
    coche_fecha_matriculacion DATE,
    CONSTRAINT fk_modelo FOREIGN KEY (modelo_id) REFERENCES modelo(modelo_id)
) ENGINE=InnoDB;

INSERT INTO coche
(coche_id, modelo_id, coche_matricula, coche_fecha_matriculacion)
VALUES
(1, 1, '505050FRA', '2004-04-13'),
(2, 6, '523698FGW', '2004-04-13'),
(3, 5, '874126WEP', '2004-04-13'),
(4, 2, '963214PQM', '2004-04-13'),
(5, 6, '852147PFD', '2004-04-13'),
(6, 7, '932855YNF', '2004-04-13'),
(7, 5, '820144MDO', '2004-04-13');

```

Ahora se va a realizar una consulta que devuelva marca, modelo y matrícula de cada coche:

```

SELECT marca_nombre, modelo_nombre, coche_matricula
FROM coche
INNER JOIN modelo ON modelo.modelo_id = coche.modelo_id
INNER JOIN marca ON marca.marca_id = modelo.marca_id;

```

El orden en el que se realizan las uniones afecta dependiendo de la situación. No importa si se coloca modelo en el FROM y coche en el INNER JOIN, pero la marca sí debe ir después de la tabla de modelo y coche, porque de lo contrario los campos aún no han sido declarados. Se llamaría al campo modelo.marca_id antes de tiempo.

OUTER JOIN

El concepto de los OUTER JOIN, junto al concepto de agrupaciones son los dos temas que más dificultad de entendimiento ofrecen a los no iniciados en el SQL.

Su idea es unir los datos de la tabla A con los datos de la tabla B, teniendo **la tabla A una prioridad mayor**. Devuelve **todos los registros de la tabla A** pero de la tabla B solo los que **cumplan la condición**. Si no cumplen la condición sus columnas devolverán un valor NULL.

En este ejemplo se ve su utilidad, donde se quieren listar todos los modelos junto a su modelo padre. Es posible que un modelo no tenga un modelo padre:

```
SELECT hijo.modelo_nombre modelo_hijo, padre.modelo_nombre AS modelo_padre
FROM modelo AS hijo
LEFT JOIN modelo AS padre ON padre.modelo_id = hijo.modelo_padre;
```

La sintaxis consiste en reemplazar la palabra INNER por LEFT o RIGHT. Cuando se utiliza LEFT quiere decir que la tabla de la izquierda (hijo) es la principal. Si se utiliza RIGHT, la tabla principal sería “padre” (y se obtendrían resultados incorrectos).

Por lo general, lo óptimo es formular las consultas de tal manera que solo se utilice LEFT JOIN, puesto que es **mucho más legible** (se lee de izquierda a derecha). Lo mejor es saber cómo funcionan los RIGHT JOIN por si aparecen en una aplicación existente, pero se debe **evitar su uso**.

Los OUTER JOIN son útiles para relaciones de cero a muchos. De especial utilidad son los OUTER JOIN junto a agrupaciones. Para ver un ejemplo de esto, se van a listar marcas junto al número de coches que tienen (coches de la tabla coche, no modelos):

```
SELECT marca.marca_nombre, COUNT(coche.coche_id) AS num_coches
FROM marca
LEFT JOIN modelo ON modelo.marca_id = marca.marca_id
LEFT JOIN coche ON coche.modelo_id = modelo.modelo_id
GROUP BY marca.marca_nombre
ORDER BY num_coches;
```



Nota

Los OUTER JOIN Permiten obtener datos de la entidad fuerte junto a los de la entidad débil (opcionalmente).

No se fije demasiado en la agrupación, pero fíjese en que también puede utilizar varios LEFT JOIN.

El orden de los LEFT JOIN es importante. Si hace un OUTER JOIN, después no podrá hacer un INNER JOIN, porque de lo contrario se invalidarían los OUTER JOIN anteriores. En este caso interesan marcas junto al número de coches. Si se quisieran marcas que tuvieran modelos (desaparecería Ford), la unión con la tabla “modelo” debería ser de tipo INNER.

Una característica de los OUTER JOIN es que permiten invertir la situación. Es decir, la idea principal es listar todos los registros de la tabla A y algunos de la tabla B. Pero mediante una filtración en el WHERE podrían listarse todos los registros de la tabla A **que no se encuentran** en B. Para ello, simplemente hay que comprobar que algún campo no nulo de B tenga un valor nulo.

Por ejemplo, listar todos los modelos que no tienen coches:

```
SELECT marca.marca_nombre, modelo.modelo_nombre
FROM marca
INNER JOIN modelo ON modelo.marca_id = marca.marca_id
LEFT JOIN coche ON coche.modelo_id = modelo.modelo_id
WHERE coche.coche_id IS NULL;
```

Si el coche_id está nulo quiere decir que el modelo no tiene correspondencia en esa tabla. Cuando veamos el apartado sobre subconsultas veremos una manera distinta de hacer esto, aunque el OUTER JOIN de este tipo es más eficiente por lo general.



Nota

Para identificar consultas que necesitan un OUTER JOIN, puede hacerlo si ve alguno de los siguientes requisitos:

- Que aparezcan ceros en el resultado de un COUNT() en un GROUP BY.
- Cuando se necesitan datos de dos tablas pero los de una tabla son opcionales.
- Cuando se piden los datos de una tabla que no tienen relación con otra tabla.

SELF JOINS

Ya se vio antes un ejemplo de SELF JOIN aunque no se haya mencionado. Un SELF JOIN es simplemente **una unión sobre la misma tabla**; una unión reflexiva. Para referenciar una tabla más de una vez, hay que añadirles alias porque, de lo contrario, no sería posible identificar cada referencia a la tabla. Puede realizarse INNER y OUTER JOINS; el primer ejemplo de OUTER JOIN era un SELF JOIN.

Otro ejemplo puede ser más sencillo: listar modelos que son hijos de otro modelo:

```
SELECT marca.marca_nombre, hijo.modelo_nombre
FROM modelo AS hijo
INNER JOIN modelo AS padre ON padre.modelo_id = hijo.modelo_padre
INNER JOIN marca ON marca.marca_id = hijo.marca_id;
```

La sintaxis es la misma que con cualquier JOIN, simplemente debe asegurarse de utilizar alias, porque si no el SGBD devolverá un error diciendo que la tabla está duplicada.

EQUI JOIN

Todos los ejemplos aquí expuestos correspondían a uniones de tipo EQUI JOIN. Este tipo de uniones simplemente quieren decir que **la condición en el ON es de igualdad**.

Es más fácil entender en qué consisten si se explican las uniones que no son EQUI. Para ello tendría que realizarse una unión en la que la condición no sea de igualdad simple, sino que haya signos de mayor o igual o condiciones especiales. Por ejemplo, se realizará una consulta en la que muestren marcas junto al número de modelos que tienen presentados desde el 2005. Si no tienen ningún modelo, que muestre un cero:

```
SELECT marca.marca_nombre, COUNT(modelo.modelo_id) AS modelos_desde_2005
FROM marca
LEFT JOIN modelo ON modelo.marca_id = marca.marca_id
AND modelo.modelo_fecha_lanzamiento > '2005-01-01'
GROUP BY marca.marca_nombre
ORDER BY modelos_desde_2005 DESC;
```

Simplemente en la condición del JOIN se especificarán condiciones que no son de igualdad.

NATURAL JOIN

El NATURAL JOIN es una simplificación del INNER JOIN. La idea es omitir la cláusula ON del JOIN y SQL interpretará que los campos que se llamen igual en ambas tablas deberán ser utilizados para determinar las correspondencias.

```
SELECT marca.marca_nombre, modelo.modelo_nombre  
FROM marca  
NATURAL JOIN modelo;
```

Ambas tablas tienen un campo llamado marca_id. SQL automáticamente interpreta que modelo.marca_id debe ser igual a marca.marca_id.

Esta es una simplificación que puede ser útil para ahorrar tiempo al hacer consultas rápidas puntuales, pero no debería ser utilizada en aplicaciones permanentes, porque si se cambia la estructura de las tablas dejaría de funcionar correctamente.



Nota

Si se añadiera un campo llamado fecha_creacion a ambas tablas, la consulta dejaría de dar resultados coherentes.



Actividades

20. Realice una consulta que devuelva una tabla que muestre el nombre del hotel, el nombre de cliente, su teléfono, su correo y las fechas de inicio y fin de la reserva.

7.3. Subconsultas

En una consulta de SQL es posible anidar otras consultas de SQL dentro. Esto es útil

cuando para filtrar las filas, se **necesitan los datos de otras filas** para determinar si debe ser utilizada o no.

Las subconsultas son consultas SELECT que se encuentran dentro de otra consulta SELECT entre paréntesis.

Con este ejemplo se entiende mejor:

```
SELECT marca_nombre, modelo_nombre, modelo_precio
FROM marca
INNER JOIN modelo ON modelo.marca_id = marca.marca_id
WHERE modelo_precio > (
    SELECT AVG(modelo_precio)
    FROM modelo
);
```

Aquí se están listando los modelos cuyo precio esté por encima de la media. Se colocará entre paréntesis una nueva consulta dentro del WHERE.

Hay que entender el orden en que se ejecuta esta consulta. Primero se ejecuta la subconsulta para conocer la media de precios. A continuación, el SGBD coloca ese valor como si se hubiera puesto el valor a mano. Puede decirse que la media son 16.000. Es como si se hubiera escrito a mano que el modelo_precio sea mayor que 16.000.

Comprobaciones en el WHERE

Con las subconsultas se realiza una serie de comprobaciones en el WHERE; estas se explican a continuación.

Comparaciones de valor simple

Comparaciones normales ($=$, $>$, $<$, \neq). Es posible comprobar que el valor de una columna sea igual, mayor, menor, etc. que el valor devuelto por una subconsulta. El ejemplo anterior utiliza una comparación de este tipo.

Comparador de pertenencia

Es posible comprobar que el valor de una columna se encuentre dentro los valores devueltos por una subconsulta. Para ello, se pasa una subconsulta a la función IN().

Por ejemplo, listar las marcas que tengan modelos de más de 16.000 €:

```
SELECT *
FROM marca
WHERE marca_id IN (
    SELECT marca_id
    FROM modelo
    WHERE modelo_precio > 16000
);
```

En este caso, el SGBD primero obtiene la lista de valores de marca_id que cumplen esa condición. A continuación, los coloca en el IN().

Es posible utilizar NOT IN() si se quiere invertir la comprobación.

Comparador de existencia

Mediante EXISTS y NOT EXISTS se comprueba que la subconsulta devuelve alguna fila. Si devuelve una fila, EXISTS devuelve TRUE (pasa la comprobación).

El EXISTS tiene sentido cuando se utilizan subconsultas correlacionadas. Cuando una subconsulta es correlacionada, normalmente lo mismo que se realiza con EXISTS se puede hacer con un IN(), pero en esos casos EXISTS es más eficiente porque solamente comprueba que existan filas, no devuelve todos los resultados.

```
SELECT *
FROM marca
WHERE EXISTS (
    SELECT modelo_id
    FROM modelo
    WHERE modelo_precio > 16000
    AND modelo.marca_id = marca.marca_id
);
```



Consejo

Normalmente, cuando en el enunciado de un ejercicio se pide que el valor de alguna columna debe tener un valor que depende de otro registro, se necesitará una subconsulta.

El EXISTS es una comprobación por sí sola. No hay que poner ningún nombre de columna delante.

Comparación cuantificada

Para ello, se utiliza el ANY y el ALL. Sirve para que una comprobación se cumpla con todos los valores de una subconsulta o con alguno de ellos.

Por ejemplo, listar modelos que valgan más que cualquier modelo de SEAT:

```
SELECT marca_nombre, modelo_nombre
FROM marca
INNER JOIN modelo ON modelo.marca_id = marca.marca_id
WHERE modelo_precio > ALL (
    SELECT modelo_precio
    FROM modelo
    INNER JOIN marca ON modelo.marca_id = marca.marca_id
    WHERE marca_nombre = 'SEAT'
);
```

Ahora, modelos que tengan un precio mayor que algún modelo de Renault:

```
SELECT marca_nombre, modelo_nombre
FROM marca
INNER JOIN modelo ON modelo.marca_id = marca.marca_id
WHERE modelo_precio > ANY (
    SELECT modelo_precio
    FROM modelo
    INNER JOIN marca ON modelo.marca_id = marca.marca_id
    WHERE marca_nombre = 'Renault'
);
```

Tipos de subconsulta

Dentro de las subconsultas puede distinguirse entre dos tipos, según el número de filas; estas se describen a continuación.

Subconsultas que devuelven una fila

Dentro de este tipo, se encuentran las que devuelven una fila con una sola columna o las que devuelven una tupla (fila con varias columnas).

Cuando se devuelve una sola columna, pueden hacerse comparaciones directas como si se tratara de un valor escrito a mano por el usuario; comparaciones de mayor, igual, menor, distinto, etc.

Si una subconsulta devuelve más de una fila y se realizan comparaciones de este tipo, MySQL devolverá un error, porque es incorrecto.

Cuando se devuelve una tupla, se puede comprobar que una tupla de la SELECT principal coincida con la tupla de la subconsulta. Vea un ejemplo de cómo funcionaría:

```
SELECT *
FROM tabla1
WHERE (col1, col2) = (
    SELECT tabla2.col1, tabla2.col2
    FROM tabla2
);
```



Nota

Con este tipo de subconsultas no se pueden hacer comparaciones de mayor o menor, porque no tendría sentido.

Es simplemente para ver la sintaxis. Esto no se utiliza muy a menudo. Suele ser útil cuando las tablas tienen claves primarias compuestas por varias columnas y quiere comprobarse que el valor de la clave primaria es igual a otro.

Subconsultas que devuelven más de una fila

Estas son las subconsultas que devuelven varios resultados. Pueden no devolver ninguna fila, devolver una sola o devolver muchas.

Dentro de este tipo también se diferencia entre las que devuelven una sola columna o varias.

Cuando se devuelve una sola columna, la subconsulta se trata como una lista de valores.

Se pueden hacer comparaciones con IN(), ANY(), EXISTS() o ALL(). No es posible utilizar operadores de comparación normales como el signo igual, mayor que, etc., porque se devuelven varios resultados.

Cuando la subconsulta devuelve varias columnas, el resultado devuelto es como una tabla. Pueden hacerse comparaciones para ver si una tupla se encuentra entre los resultados. Es una sintaxis igual que la de las subconsultas que devuelven una fila con varias columnas, pero con otro tipo de comparaciones:

```
SELECT *
FROM tabla1
WHERE (col1, col2) IN (
    SELECT tabla2.col1, tabla2.col2
    FROM tabla2
);
```

Es importante diferenciar entre una subconsulta de una sola fila y una subconsulta de varias filas pero que solo devuelva una fila. Cuando una subconsulta de varias filas devuelve una sola fila, quiere decir que es una casualidad, pero es posible que en el futuro devuelva más de una.

Una subconsulta de una fila devolverá siempre una fila (o ninguna si no hay datos). Una subconsulta de varias devolverá un número de filas que dependerá de los datos que haya en las tablas.

Subconsultas correlacionadas

Una subconsulta es correlacionada cuando dentro de la subconsulta se hace **referencia a columnas y tablas utilizadas en la consulta principal**.

Es un sistema muy potente que permite hacer filtraciones verdaderamente complejas que de otra manera necesitarían de más de una consulta para ejecutarse.

Son útiles cuando se desean obtener datos que están relacionados con el registro actual de alguna manera. En los ejemplos anteriores, por ejemplo, el de modelos que valgan más de la media, no había ninguna relación entre el registro actual y la subconsulta. La subconsulta siempre devolvía los mismos resultados para cada registro.

En las subconsultas no correlacionadas, la subconsulta se ejecuta primero y los resultados que se obtienen de ella son colocados en la consulta principal, la cual se ejecutará a continuación.

En una subconsulta correlacionada este proceso cambia. Ahora la consulta principal se ejecuta y **por cada registro** evaluado se ejecuta la subconsulta.

En este ejemplo se listan modelos que tengan un precio por encima de la media de su marca:

```
SELECT marca_nombre, modelo_nombre  
FROM marca  
INNER JOIN modelo ON modelo.marca_id = marca.marca_id  
WHERE modelo_precio > (  
    SELECT AVG(e2.modelo_precio)  
    FROM modelo e2  
    WHERE e2.marca_id = modelo.marca_id  
);
```

En una subconsulta correlacionada es importante **utilizar alias** en las tablas de dentro de la subconsulta, de lo contrario no es posible referirse a la tabla de fuera y a la de dentro. Aquí, simplemente, en el WHERE de la subconsulta se añade la condición de que la marca del modelo que se está evaluando sea igual que la marca de la media que se está calculando.



Nota

En un enunciado de un problema, normalmente cuando aparezca la palabra “su”, quiere decir que se necesita una subconsulta correlacionada. Esto es así porque suele hacer referencia a que para cada registro debe hacerse una operación concreta.

El rendimiento de una subconsulta correlacionada es muy inferior que el de una subconsulta convencional, pero es mucho más útil.

Subconsultas en el SELECT

Además de en el WHERE, pueden utilizarse subconsultas dentro de la selección de columnas, entre el SELECT y el FROM. Esto es útil cuando, en una sola consulta, se quieren obtener columnas que no están muy relacionadas con la consulta principal. Para utilizar una subconsulta en el SELECT, se coloca entre paréntesis y se le asigna un alias.

A continuación, se listarán modelos junto a la media de su marca:

```
SELECT marca_nombre, modelo_nombre, modelo_precio, (
    SELECT AVG(e2.modelo_precio)
    FROM modelo e2
    WHERE e2.marca_id = modelo.marca_id
) AS media_modelo
FROM marca
INNER JOIN modelo ON modelo.marca_id = marca.marca_id;
```

Normalmente, cuando se utiliza una subconsulta en el SELECT quiere decir que se está saltando algún paso del modelo relacional. Esto se hace para aumentar el rendimiento en una aplicación. Con una sola consulta se obtienen los datos deseados.



Nota

Cuando haya que realizar una consulta que devuelva ciertas columnas y alguna de ellas traiga verdaderos quebraderos de cabeza, normalmente se podrá obtener mediante una subconsulta.

Subconsultas en el FROM

Las subconsultas en el FROM son posiblemente el apartado más complejo del SQL. No por la dificultad de implementar, sino porque permiten resolver enunciados realmente enrevesados. Si se encuentra un enunciado que parece imposible, seguramente se pueda resolver de esta manera.

Consiste simplemente en colocar, entre paréntesis y con un alias, una subconsulta dentro del FROM como si de una tabla más se tratara. Esta tabla será calculada por el SGBD (seguramente, creando una tabla temporal con los datos y utilizada en el resto de la consulta. Es posible colocar incluso una tabla correlacionada en el FROM.

Es muy raro encontrarse con problemas de este tipo. Normalmente, si una aplicación necesita de una consulta que contenga otra subconsulta en el FROM quiere decir que el modelo no se encuentra normalizado correctamente.

Ejemplo de una consulta que devuelve los modelos de la marca que tiene el modelo más caro:

```
SELECT marcas_mas_cara.marca_nombre, modelo_nombre
FROM (
    SELECT marca.marca_id, marca.marca_nombre
    FROM marca
    INNER JOIN modelo ON modelo.marca_id = marca.marca_id
    ORDER BY modelo_precio DESC
    LIMIT 1
) AS marcas_mas_cara
INNER JOIN modelo ON modelo.marca_id = marcas_mas_cara.marca_id;
```

7.4. Consejos y consideraciones al utilizar subconsultas

Para escribir una consulta que tenga una subconsulta, puede ser útil escribir primero la subconsulta por separado y ejecutarla, para ver si da los resultados esperados.

Si la subconsulta es correlacionada, lo que puede hacerse es, en vez de utilizar la condición de igualdad de la correlación, colocar un valor manualmente para probar que da los resultados deseados. Por ejemplo, para la consulta que devolvía modelos cuyo precio estuviera por encima de la media de su marca, podría reemplazarse modelo.marca_id por el id de una marca, como el número 20:

```
SELECT AVG(modelo_precio)
FROM modelo e2
WHERE e2.marca_id = 20;
```

Este ejemplo es muy simple, pero en una correlacionada con muchas filtraciones, puede ser útil.

Al utilizar subconsultas debe considerarse el rendimiento. Por cada fila evaluada en la tabla principal, es posible que haya que evaluar otras tantas filas para cada subconsulta, haciendo que haya una progresión geométrica en cuanto a número de filas a evaluar.



Actividades

21. Escriba una consulta que muestre las reservas más baratas de su hotel.
22. Necesita una consulta que muestre las reservas cuyo precio estén por encima

de la media total del año pasado.

8. Agrupaciones

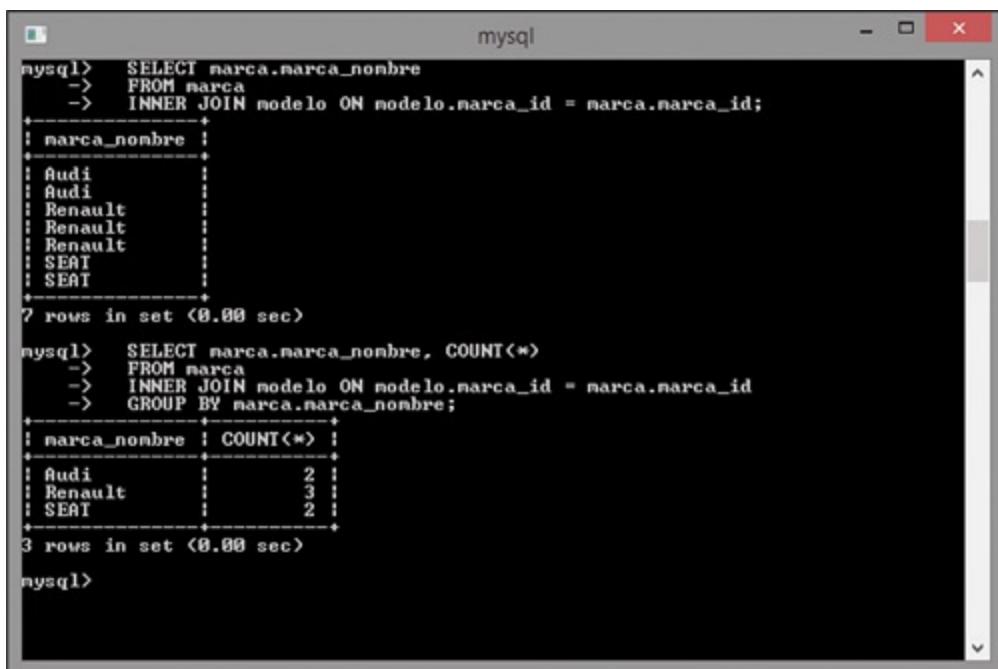
Las agrupaciones son una herramienta muy potente que el SQL proporciona para simplificar enormemente el proceso de obtener datos que, de lo contrario, requerirían mucho trabajo y muchas consultas para hacerlo.

Agrupar consiste en, una vez se han procesado los registros que cumplían con todas las condiciones de la consulta, realizar una operación extra sobre esos datos para contar duplicados, calcular medias, mínimos, máximos, etc.

8.1. Conceptos de agrupación de datos

Como ya se ha visto, agrupar consiste en hacer operaciones sobre los registros que cumplen las condiciones especificadas.

De forma gráfica sería así:



The screenshot shows a MySQL command-line interface window. It displays two SQL queries and their results. The first query is:

```
mysql> SELECT marca.marca_nombre
-> FROM marca
-> INNER JOIN modelo ON modelo.marca_id = marca.marca_id;
```

The result of the first query is:

marca_nombre
Audi
Audi
Renault
Renault
Renault
SEAT
SEAT

The second query is:

```
mysql> SELECT marca.marca_nombre, COUNT(*)
-> FROM marca
-> INNER JOIN modelo ON modelo.marca_id = marca.marca_id
-> GROUP BY marca.marca_nombre;
```

The result of the second query is:

marca_nombre	COUNT(*)
Audi	2
Renault	3
SEAT	2

En la primera consulta se unen marcas con modelos. Como es lógico, se obtienen datos duplicados sin mucho sentido. En la segunda se ha utilizado una agrupación: marca_nombre. SQL procesará la consulta original e irá fila a fila; por cada valor diferente de marca_nombre devolverá una sola fila, además, podrá realizar cálculos extra utilizando las funciones de agregación (las cuales se verán a continuación).

Podrían haberse eliminado duplicados utilizando DISTINCT simplemente, pero entonces no hubiera sido posible contar cuántos modelos tiene cada marca, porque DISTINCT simplemente elimina duplicados. GROUP BY va fila a fila comprobando los valores y realizando todas las operaciones que se le pidan.

El GROUP BY se coloca después de todas las condiciones del WHERE. Se añade la palabra GROUP BY seguida de la columna o columnas (separadas por comas) por las que quiera agruparse.

En el SELECT solamente pueden listarse funciones de agrupación, subconsultas o columnas que se encuentren dentro del GROUP BY. Si se agrupa por marca_nombre no es posible seleccionar marca_nombre y marca_pais porque la agrupación perdería su sentido.

En MySQL esto es técnicamente posible, en otros SGBD se obtendría un error. En MySQL es posible pero el valor devuelto no tiene por qué tener sentido. En este ejemplo una marca va a ser siempre del mismo país, pero obsérvelo con atención:

Cliente_nombre	Cliente_ciudad	Cliente_direccion
Alfredo	Málaga	Alameda Principal
José	Málaga	Plaza Uncibay
Marcos	Barcelona	Vía Laietana
María	Barcelona	Ronda de Sant Pere

Si se agrupa por Cliente_ciudad y se selecciona SELECT Cliente_ciudad, Cliente_Direccion, COUNT(*), la dirección sería ilógica y no seguiría el modelo relacional en absoluto. Para Málaga podría obtenerse Alameda Principal o Plaza Uncibay, ni siquiera prever cuál de los dos aparecería. En casos en los que esto no ocurra puede ser útil, aunque no es correcto hacerlo.

Requiere bastante práctica dominar las agrupaciones porque realizan tareas

complejas; es difícil a veces predecir qué haría una consulta concreta.

Cabe destacar que una consulta que utilice funciones de agregación y no tenga una agrupación **siempre devolverá una fila** de resultados como mínimo. Si se realiza COUNT(*) y no hay filas, se devolverá un cero, en vez del mensaje de que no hay resultados.



Nota

Por norma general, cuando en el enunciado de una consulta aparezcan las palabras “por cada”, está dando la pista de que es posible que la manera de resolverla sea mediante una agrupación.

8.2. Funciones de agrupación

Prácticamente todo el sentido de las agrupaciones es el uso de las llamadas funciones de agregación. De lo contrario, solo servirían para poco más que eliminar duplicados.

Las funciones de agrupación tienen una sintaxis casi idéntica a las funciones de SQL normales. Solamente cambia que la función COUNT() admite un par de parámetros especiales.

Cuando una función de agrupación es utilizada, SQL **la ejecutará solo una vez por cada grupo**. Si no se especifica ningún grupo, solo se ejecutará una vez. Por ejemplo, la consulta del ejemplo anterior sin GROUP BY:

A screenshot of a Windows command-line window titled "mysql". The window displays a SQL query and its results. The query is: "SELECT marca.marca_nombre, COUNT(*) FROM marca INNER JOIN modelo ON modelo.marca_id = marca.marca_id;". The result shows one row: "Audi" with a count of 7. The output ends with "1 row in set (0.00 sec)".

```
mysql> SELECT marca.marca_nombre, COUNT(*)
-> FROM marca
-> INNER JOIN modelo ON modelo.marca_id = marca.marca_id;
+ marca_nombre + COUNT(*) +
+ Audi           +      7 +
1 row in set (0.00 sec)

mysql>
```

La función COUNT(*) solo se ejecuta una vez por cada grupo. Solo hay un grupo en esta consulta.



Importante

SQL extrae el primer valor de marca_nombre que encuentra y hace un COUNT(*) de todo.

Esta consulta es incorrecta, pero las funciones de agrupación globales son muy útiles también. Por ejemplo, vea las siguientes consultas:

A screenshot of a Windows command-line window titled "mysql". It contains three separate SQL queries. The first query selects the count of all rows from the "marca" table and stores it in a variable named "total_marcas", which is then displayed as the value 5. The second query selects the average price from the "modelo" table and stores it in a variable named "media_precio", which is then displayed as the value 16066.666667. The third query selects the maximum and minimum prices from the "modelo" table and stores them in variables named "precio_maximo" and "precio_minimo" respectively, which are then displayed as 18000.00 and 14000.00.

```
mysql>     SELECT COUNT(*) AS total_marcas
->     FROM marca;
+-----+
| total_marcas |
+-----+
|      5      |
+-----+
1 row in set <0.00 sec>

mysql>
mysql>     SELECT AVG(modelo_precio) AS media_precio
->     FROM modelo;
+-----+
| media_precio |
+-----+
| 16066.666667 |
+-----+
1 row in set <0.00 sec>

mysql>
mysql>     SELECT MAX(modelo_precio) AS precio_maximo, MIN(modelo_precio) AS precio_minimo
->     FROM modelo;
+-----+-----+
| precio_maximo | precio_minimo |
+-----+-----+
|    18000.00   |     14000.00  |
+-----+-----+
1 row in set <0.00 sec>

mysql>
```

Pueden obtenerse datos globales acerca de todos los datos procesados. Lo que no tiene sentido es añadir campos de tablas como se había hecho antes. Es posible añadir condiciones al WHERE, con el objetivo de hacer las operaciones solo sobre un conjunto de datos.

Las funciones de agrupación disponibles son las que se describen a continuación.

COUNT()

Esta es una función especial porque **admite parámetros especiales**.

Cuando se quiere hacer un cuenteo general, se añade un **asterisco** como parámetro para especificar que hay que contar **todas las filas**.

Al ver el apartado sobre los OUTER JOIN se vio que eran útiles cuando eran utilizadas junto a agrupaciones. Esto es así porque puede especificarse qué columna contar y solamente contará el **número de filas** en las que esa columna tenga un **valor no nulo**. En el ejemplo del apartado de OUTER JOIN de nuevo fíjese esta vez en la agrupación:

```
SELECT marca.marca_nombre, COUNT(coche.coche_id) AS num_coches
FROM marca
LEFT JOIN modelo ON modelo.marca_id = marca.marca_id
LEFT JOIN coche ON coche.modelo_id = modelo.modelo_id
GROUP BY marca.marca_nombre
ORDER BY num_coches;
```

Si en el COUNT() se añadiera un asterisco en vez de coche.coche_id, el resultado no sería útil. Nunca se obtendría un cero, porque siempre habría una fila para cada grupo. Si se especifica una columna dentro del COUNT (solo puede especificarse una), se contarán las veces que esa columna no contiene nulos dentro del grupo.

COUNT permite otro parámetro especial más: contar valores distintos de una determinada columna. Por ejemplo, contar distintos años de presentación de modelos:

```
SELECT COUNT(DISTINCT YEAR(modelo.fecha_lanzamiento)) AS 'distintos_años'
FROM modelo;
```

En este caso solamente cuentan valores distintos del año de fecha de lanzamiento, se obtendrían 5 filas.



Nota

En COUNT DISTINCT pueden especificarse más de una columna o expresión

separada por comas.

Demás funciones de agrupación

Las demás funciones de agrupación son mucho más simples, se listan a continuación:

- **SUM()**: recibe una columna o expresión y devuelve la suma de sus valores. Solo es útil al trabajar con valores numéricos.
- **AVG()**: igual que SUM pero devuelve la media de los valores.
- **MIN()**: recibe una columna o expresión y devuelve el menor de sus valores dentro del grupo. Es útil para números y caracteres. Si el parámetro es una cadena de caracteres, devolverá el registro que sea menor dentro del orden alfabético.
- **MAX()**: igual que MIN pero devolviendo el valor máximo.

Existen otras funciones matemáticas muy poco comunes como **DEV()** y **VARIANCE()**. Cada SGBD incorpora sus funciones de agregación extra propias. No se suelen utilizar.



Actividades

23. Liste los nombres de hotel junto al número de reservas que tienen para el presente año. Si no tienen ninguna reserva, que muestre un cero.
 24. Liste los nombres de hotel junto a la media de recaudación de este año.
 25. Liste los nombres de hotel junto a la reserva más cara y la más barata.
-

8.3. Agrupación multicolumna

A la hora de agrupar, el aspecto clave es escoger **las columnas por las que agrupar**. En los ejemplos más sencillos lo habitual es agrupar por un solo campo, pero es posible **agrupar por más de uno**; cuando esto sucede, se **agrupa de izquierda a derecha**. Primero se agrupa por el primer campo, luego por el segundo y así sucesivamente.

No es posible seleccionar campos que no se encuentren dentro de la agrupación. A veces es conveniente añadir campos a la agrupación simplemente para poderlos

seleccionar. Por ejemplo, se quiere elaborar una lista de marcas junto a su número de modelos y al país de la marca:

```
SELECT marca_nombre, marca_pais, COUNT(modelo.modelo_id) AS total_modelos
FROM marca
LEFT JOIN modelo ON modelo.marca_id = marca.marca_id
GROUP BY marca_nombre, marca_pais;
```

Es importante el orden porque si se agrupa primero por país no se obtendrían los resultados correctos.

Otro ejemplo puede ser una consulta que liste modelos (junto al nombre de su marca) y el número de coches matriculados:

```
SELECT marca_nombre, modelo_nombre, COUNT(coche.coche_id) AS total_coches
FROM marca
INNER JOIN modelo ON modelo.marca_id = marca.marca_id
LEFT JOIN coche ON coche.modelo_id = modelo.modelo_id
GROUP BY marca_nombre, modelo_nombre;
```

En este caso no se agrupa por más campos para poder obtener los valores; se agrupa por varios campos porque es necesario.

Como todas las agrupaciones, las múltiples requieren de práctica para ser dominadas.



Ejemplo

Puede ser útil realizar un informe que muestre el número de visitas por cada mes. No sería suficiente agrupar solo por mes. Haría falta agrupar primero por año y luego por mes, de lo contrario se obtendrían cifras no muy útiles (normalmente).

8.4. Agrupación vía expresiones

En una agrupación lo común es agrupar por columnas de alguna tabla. No es imprescindible, también puede agruparse por el resultado de una función, por el resultado de una operación matemática, etc.

Por ejemplo, número de coches presentados por año:

```
SELECT YEAR(modelo_fecha_lanzamiento) AS 'año_lanzamiento', COUNT(*)  
FROM modelo  
GROUP BY YEAR(modelo_fecha_lanzamiento);
```

8.5. Condiciones de filtrado de grupos

Pueden especificar condiciones extra a la hora de agrupar gracias al potentísimo sistema llamado GROUP BY HAVING. Gracias a él puede realizarse un filtrado posterior a la agrupación para descartar filas que no interesen. Primero se hace el filtrado del WHERE, a continuación se ejecuta el GROUP BY. **El HAVING va al final**, hace comprobaciones sobre los resultados del GROUP BY.

Por ejemplo, en una consulta que liste marcas junto a su número de modelos, donde solo interesan las marcas que tengan más de un modelo:

```
SELECT marca_nombre, COUNT(*)  
FROM marca  
INNER JOIN modelo ON modelo.marca_id = marca.marca_id  
GROUP BY marca_nombre  
HAVING COUNT(*) > 1;
```

Aquí no se hace un LEFT JOIN ni se cuentan solo las ocurrencias de modelo_id porque interesan marcas que tengan más de un modelo, no hace falta considerar las marcas con cero resultados.

Tras el GROUP BY, se añade la palabra HAVING seguida de las condiciones que se desean aplicar. Aquí pueden utilizarse funciones de agregación e incluso subconsultas para las comprobaciones. En el HAVING solo pueden añadirse comprobaciones sobre campos que son utilizados en el GROUP BY o sobre funciones de agregación. Aquí en

esta consulta se comprueba que el contador es mayor que 1.



Importante

Podrían hacerse comprobaciones sobre el campo marca_nombre, pero no sobre marca_pais porque no se encuentra dentro de la agrupación.

Es posible probar con una consulta más compleja que liste las marcas cuyo precio esté por debajo de la media global de precio:

```
SELECT marca_nombre
FROM marca
INNER JOIN modelo ON modelo.marca_id = marca.marca_id
GROUP BY marca_nombre
HAVING AVG(modelo_precio) < (
    SELECT AVG(e2.modelo_precio)
    FROM modelo e2
);
```



Actividades

26. Liste los hoteles junto al número de reservas. Solo interesan los hoteles con más de 2 reservas. Ordene los resultados por número de reservas, de mayor a menor.
-

9. Vistas

Las vistas permiten simplificar el realizar **consultas frecuentes** y el manejo de **permisos** entre los usuarios de la base de datos. Consisten en **consultas de SQL almacenadas** en el SGBD que pueden ser utilizadas como si fueran tablas.

Es posible que un usuario tenga permiso para ver solamente modelos de marcas de

Alemania, por ejemplo. La manera de gestionar permisos no permitiría definir esa regla por sí sola, sin embargo, podría crearse una vista que contuviera únicamente modelos de Alemania. A ese usuario podría quitársele el acceso a la tabla de marcas pero dárselo a la vista de modelos de Alemania.

9.1. Concepto de vista (view)

Una vista es una consulta de SQL almacenada en el SGBD. Cualquier consulta puede ser utilizada para una vista, en MySQL la única restricción es que en esa consulta no debe haber ninguna subselect en el FROM. En este ejemplo en el que se crea una vista con los modelos de Alemania se puede ver su sintaxis:

```
CREATE VIEW modelos_de_alemania
AS
SELECT modelo.*
FROM modelo
INNER JOIN marca ON marca.marca_id = modelo.marca_id
WHERE marca_pais = 'Alemania';
```

La sintaxis es parecida a un INSERT SELECT. Primero se empieza con CREATE VIEW seguido del nombre de la vista (sin espacios o caracteres extraños. A continuación, se utiliza la palabra AS seguida de la consulta que se va a utilizar.

La vista se creará y el nombre de los campos será **el mismo que el de los campos seleccionados** en el SELECT. En este caso solo interesan los campos de la tabla modelo, todos. Es posible que quisiéramos solo unos pocos, o cambiarles el nombre. También es posible aplicar cambios sobre los datos originales.

Al crear una vista, el **tipo de datos es idéntico al original** de la SELECT. El SGBD sabe detectar qué tipo de dato contiene cada columna y lo aplicará a la vista.

Para probar la vista, ejecute SELECT * FROM modelos_de_alemania;. Verá que la estructura es igual que la de la tabla “modelo” pero con los datos ya filtrados.

Se probará con una vista que liste marca, modelo y año de lanzamiento para ver cómo personalizar nombres de columna:

```
CREATE VIEW marca_modelo_ano
AS
SELECT marca.marca_nombre, modelo.modelo_nombre,
YEAR(modelo.fecha_lanzamiento) AS 'ano_lanzamiento'
FROM modelo
INNER JOIN marca ON marca.marca_id = modelo.marca_id;
```

Se utiliza un alias para personalizar el nombre de la columna. Si no lo hiciera, acceder a esa columna desde la VIEW sería engorroso. El tipo de dato se detecta automáticamente. MySQL considera al campo `ano_lanzamiento` como un entero de cuatro dígitos.

En la vista puede utilizarse cualquier tipo de consulta. La única restricción de MySQL es que no deben tener subconsultas en el FROM.

Utilizar una vista en una consulta SELECT es idéntico a utilizar una tabla normal. En el FROM de una SELECT es posible utilizar tanto tablas como vistas indistintamente.

Si se modifica un dato en la tabla original, el cambio se ve reflejado en la vista. Si en la tabla “modelo” cambia el precio del Audi R8 a 200.000 €, el cambio se verá al consultar la vista de `modelos_de_alemania`.



Importante

No es posible tener tablas y vistas con el mismo nombre; se obtendría un error al crearlas.



Importante

En determinadas circunstancias y SGBD, la utilización de vistas aumenta el rendimiento de la base de datos, puesto que es más fácil consultar una vista que consultar las tablas originales con un montón de filtraciones. En MySQL esto no es apenas relevante porque no hay un aumento en el rendimiento significativo.

9.2. Criterios para el uso de vistas

Las vistas son útiles para la **gestión de permisos**, para dar acceso a un determinado conjunto de filas concreto a una serie de usuarios.

También son útiles cuando una aplicación realiza una **consulta compleja muy frecuentemente**. De esta manera el propio SGBD almacenaría esa consulta compleja y los clientes de la base de datos accederían a esos datos de una manera mucho más sencilla.

Otro caso de uso es el que se produce cuando varias aplicaciones se conecten a la base de datos. Es posible que todas ellas tengan que hacer una consulta con gran cantidad de filtraciones complicadas. Si se crea una vista que encapsule todas esas filtraciones, además de simplificar la consulta, se asegura de que **todas las aplicaciones aplican las filtraciones correctas**.

9.3. Creación, modificación y borrado de vistas

Ya se ha estudiado la sintaxis básica para crear vistas. A continuación, se probará con una vista más que liste marcas junto a su número de modelos:

```
CREATE VIEW marcas_modelos
AS
SELECT marca_nombre, COUNT(modelo.modelo_id) AS total_modelos
FROM marca
LEFT JOIN modelo ON marca.marca_id = modelo.marca_id
GROUP BY marca_nombre;
```

Para modificar una vista hay que utilizar la sentencia ALTER VIEW. Con ella puede sustituirse la consulta utilizada. Vamos a modificar la vista marcas_modelos para no incluir ceros, solamente se quieren marcas que tengan modelos:

```
ALTER VIEW marcas_modelos  
AS  
SELECT marca_nombre, COUNT(*) AS total_modelos  
FROM marca  
INNER JOIN modelo ON marca.marca_id = modelo.marca_id  
GROUP BY marca_nombre;
```

La sintaxis es idéntica a CREATE VIEW. Solamente se cambia CREATE por ALTER. La vista debe existir ya con ese nombre, de lo contrario se recibirá un error diciendo que la vista no existe.

Para borrar una vista se utiliza DROP VIEW seguido del nombre de la misma:

```
DROP VIEW marcas_modelos;
```

El nombre de la vista debe existir, de lo contrario se obtendrá un error.

También es posible ver qué consulta se utilizó para generar una vista. De esa manera pueden verse las filtraciones y las tablas que utiliza. Para ello, se utiliza SHOW CREATE VIEW seguido del nombre de la vista.



Actividades

-
27. Cree una vista que muestre las reservas para el presente año. Llámela “reservas_actuales”.
-

9.4. Vistas actualizables

Ya se ha visto que las vistas se pueden **consultar de manera idéntica a tablas normales**. Pueden ser utilizadas en subconsultas o en el FROM de una SELECT sin ninguna diferencia con respecto a una tabla física.

La complicación llega cuando se quiere añadir, eliminar o modificar registros de una

vista. **Hay vistas que son actualizables y vistas que no lo son.**

En las vistas actualizables es posible realizar operaciones INSERT, UPDATE y DELETE. Para que una vista sea actualizable se deben cumplir las siguientes condiciones (en cada SGBD pueden haber variaciones):

- No puede haber funciones de agregación (SUM, MAX, COUNT, etc.), ni GROUP BY, ni DISTINCT, subconsultas en el SELECT, etc.
- No puede haber uniones entre tablas.
- No se puede utilizar una vista no actualizable en el FROM de la vista (una vista que consulte datos de otra vista, la cual no sea actualizable).

Esas son las condiciones generales, para que se puedan realizar operaciones INSERT hay un par de restricciones más:

- No puede haber nombres de columna duplicados.
- Si la tabla original tiene un campo no nulo sin valor por defecto que no esté presente en la vista, no se podrán realizar inserciones. De lo contrario se estarían realizando inserciones incompletas.
- Las columnas de la vista deben ser referencias directas a columnas de la tabla principal. Si se aplican expresiones o funciones en la columna, esta no será actualizable.

Básicamente, la idea es que una vista puede ser actualizable cuando sea posible realizar esa misma modificación sobre la tabla original sin recibir errores. Por ejemplo, en la tabla de marcas_modelos, no tendría sentido modificar el campo “total_modelos” porque es un campo calculado.



Importante

En la vista de modelos_de_alemania tendría sentido modificar el precio de un modelo, por ejemplo, pero no sería posible, porque hay una unión con la tabla de marcas.

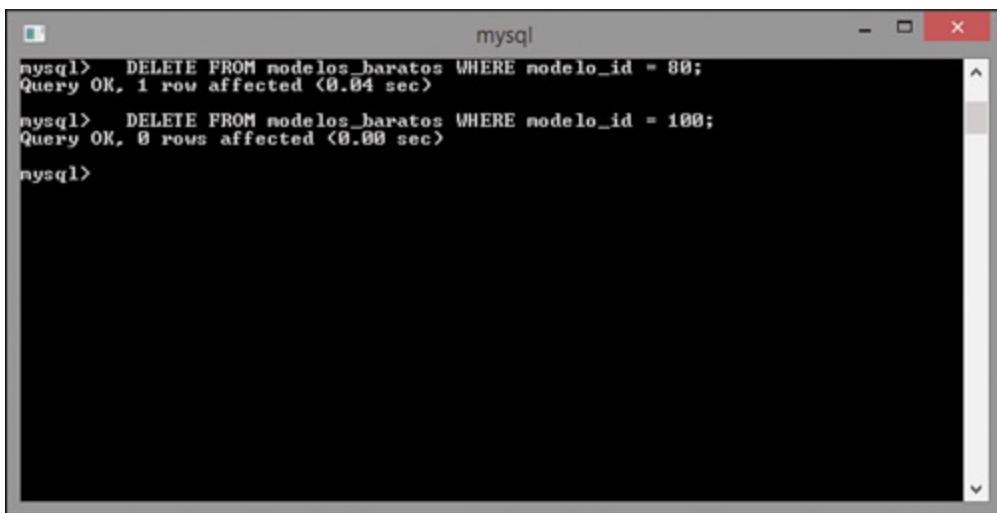
Ahora va a crearse una vista que liste modelos cuyo precio sea inferior a 18.000 €. Esta vista será actualizable porque cumple con todos los requisitos:

```
CREATE VIEW modelos_baratos
AS
SELECT *
FROM modelo
WHERE modelo_precio < 18000;
```

Pueden insertarse nuevos registros sobre modelos_baratos porque hay una **correspondencia directa con la tabla de modelos**. También se pueden insertar incluso modelos que tengan un precio mayor a 18.000 €. En ese caso el registro se insertaría en la tabla de modelos, pero no sería visible en la vista de modelos baratos.

```
INSERT modelos_baratos
(modelo_id, marca_id, modelo_nombre, modelo_precio, modelo_fecha_lanzamiento)
VALUES
(80, 31, 'Up!', 12000, '2012-07-08'),
(90, 31, 'Polo', 15400, '2008-11-25'),
(100, 31, 'Golf', 20000, '2013-02-01');
```

Pueden borrarse registros utilizando DELETE FROM. Es posible insertar filas que luego no se encuentren en la vista, pero no es posible borrar filas que no estén presentes:



The screenshot shows a Windows-style terminal window titled "mysql". It contains the following MySQL session:

```
mysql> DELETE FROM modelos_baratos WHERE modelo_id = 80;
Query OK, 1 row affected (0.04 sec)

mysql> DELETE FROM modelos_baratos WHERE modelo_id = 100;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

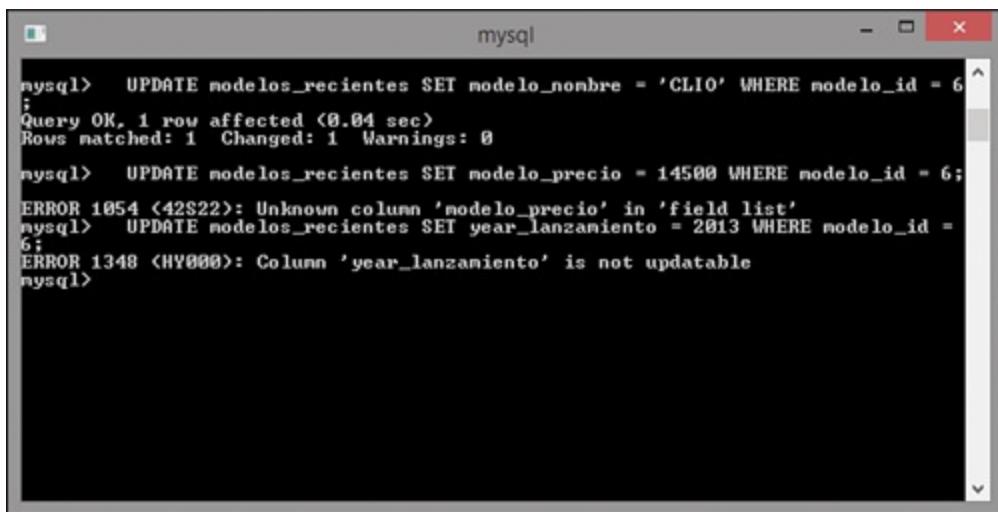
Como se aprecia, el primer registro es eliminado. El registro con id 100 (el

Volkswagen Golf) no es eliminado.

Modificar filas significa moverse en un terreno un poco más amplio, porque es posible que **algunos campos sean actualizables y otros no**. Se creará una vista para mostrar un ejemplo:

```
CREATE VIEW modelos_recientes
AS
SELECT modelo_id, marca_id, modelo_nombre, modelo_padre,
YEAR(modelo_fecha_lanzamiento) AS year_lanzamiento
FROM modelo
WHERE modelo_fecha_lanzamiento >= DATE_ADD(CURDATE(), INTERVAL -3 YEAR);
```

En esta vista se muestran modelos presentados en los últimos tres años. El campo modelo_precio no está disponible, por lo tanto, no es posible modificar ese campo desde la vista. El campo year_lanzamiento es un campo calculado, por lo tanto, no es posible modificarlo tampoco.



The screenshot shows a terminal window titled "mysql" running on Windows. It displays the following SQL session:

```
mysql> UPDATE modelos_recientes SET modelo_nombre = 'CLIO' WHERE modelo_id = 6;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> UPDATE modelos_recientes SET modelo_precio = 14500 WHERE modelo_id = 6;
ERROR 1054 (42S22): Unknown column 'modelo_precio' in 'field list'
mysql> UPDATE modelos_recientes SET year_lanzamiento = 2013 WHERE modelo_id = 6;
ERROR 1348 (HY000): Column 'year_lanzamiento' is not updatable
mysql>
```

Como se ve, el campo modelo_nombre es plenamente modificable sin problemas. El campo modelo_precio no se encuentra en la vista, por lo que no es modificable. El campo year_lanzamiento no es un campo modificable.



Actividades

28. Pruebe a crear, modificar y eliminar registros en la tabla

“reservas_actuales”. Pruebe a insertar un registro con una fecha que no sea del presente año, para ver si funciona.

10. Funciones avanzadas

En esta sección se verán algunas funciones avanzadas que proporcionan los SGBD para su **administración**. Generalmente, cada SGBD proporciona un conjunto diferente de funciones avanzadas para cubrir necesidades específicas. *MySQL* soporta la mayoría de ellas, aunque no es tan destacable en este apartado como otras soluciones, como *Oracle Database*.

10.1. Restricciones. Integridad de bases de datos

El punto débil de *MySQL* posiblemente sea el manejo de restricciones y de las cláusulas de integridad. No obstante, soporta las más comunes y permite un **uso normal** de cualquier base de datos. A continuación, se listarán las restricciones posibles.

Restricciones de clave primaria y claves únicas

Cuando un campo es PRIMARY KEY o UNIQUE, no se permitirá **bajo ningún concepto** insertar una fila que tenga un valor ya repetido en un registro anterior. Si se trata de insertar una fila con un valor duplicado, el SGBD devolverá un error y rechazará la transacción. En los campos UNIQUE sí se permiten nulos; si hay varios registros con un campo UNIQUE con valor nulo no pasaría nada. Lo que no puede haber es duplicados no nulos.



Consejo

Cuando una columna no debe tener duplicados, utilice siempre la restricción UNIQUE, así se asegurará de que no haya datos inválidos en la base de datos.

Restricciones de clave foránea

Este tipo de restricciones solo las proporciona el motor InnoDB de MySQL. En un campo clave foránea no es posible introducir valores que no se **correspondan con un registro real** en la tabla referenciada. Por ejemplo, en la tabla “modelo” no sería posible hacer que el campo marca_id del modelo “Ibiza” tuviera de valor el número 90, porque no existe ninguna marca con ese id.

Al borrar tablas de la tabla principal también puede haber problemas. No podría borrarse la marca SEAT de la tabla “marca” sin antes borrar sus registros en la tabla de modelos porque, de lo contrario, esos registros quedarían **huérfanos**.

Para estas situaciones existen las cláusulas especiales llamadas ON UPDATE y ON DELETE, las cuales permiten que cuando un registro se modifique, las claves foráneas sean modificadas también; o que cuando un registro sea eliminado, se apliquen cambios en las claves foráneas.

ON UPDATE

Piense qué pasaría si cambiara la marca_id de SEAT desde 10 a 19. Los registros de la tabla “modelo” dejarían de estar enlazados correctamente con su marca.

Para esto, el SGBD cuenta con **tres opciones**: rechazar la modificación (comportamiento estándar), modificar automáticamente los registros de la tabla “modelo” de tal manera que estén actualizados (modificación en cascada) o dejar el campo marca_id de la tabla modelo como NULL (porque ya no tiene un valor fiable).

Para ello, a la hora de crear una tabla o de modificarla, tiene que especificarse el tipo de restricción a aplicar. Se hará que las modificaciones se realicen en cascada en la tabla de modelos:

```
ALTER TABLE modelo DROP FOREIGN KEY fk_marca;  
ALTER TABLE modelo ADD CONSTRAINT fk_marca FOREIGN KEY (marca_id)  
REFERENCES marca(marca_id) ON UPDATE CASCADE;
```

Ahora, si se modifica la clave primaria de la tabla marca, los registros de la tabla de modelos cambian:

```
mysql> UPDATE marca SET marca_id = 18 WHERE marca_id = 18;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT marca_id, modelo_nombre FROM modelo;
+-----+-----+
| marca_id | modelo_nombre |
+-----+-----+
|     18    | Ibiza        |
|     18    | Cordoba      |
|     20    | Megane       |
|     20    | CLIO         |
|     20    | Captur       |
|     30    | A3           |
|     30    | R8           |
|     31    | Polo          |
|     31    | Golf          |
+-----+-----+
9 rows in set (0.00 sec)

mysql>
```

Los valores posibles son CASCADE, NO ACTION y SET NULL. NO ACTION es el valor por defecto, es como no añadir nada. SET NULL quiere decir que cuando se modifica marca_id de la tabla “marca”, los registros de la tabla “modelo” pasan a tener ese campo como NULL.



Nota

Obviamente, esto solo es posible cuando el campo admite nulos, de lo contrario no sería posible.

ON DELETE

Piense qué pasaría si borrara la marca SEAT. Los registros de la tabla “modelo” pasarían a estar huérfanos. Aquí se cuenta con las mismas opciones que en el ON UPDATE: CASCADE, NO ACTION y SET NULL.

CASCADE borraría los registros de las tablas que referencien a ese registro borrado. NO ACTION es la acción por defecto: rechazar el borrado. SET NULL haría que ese campo pasara a estar nulo.

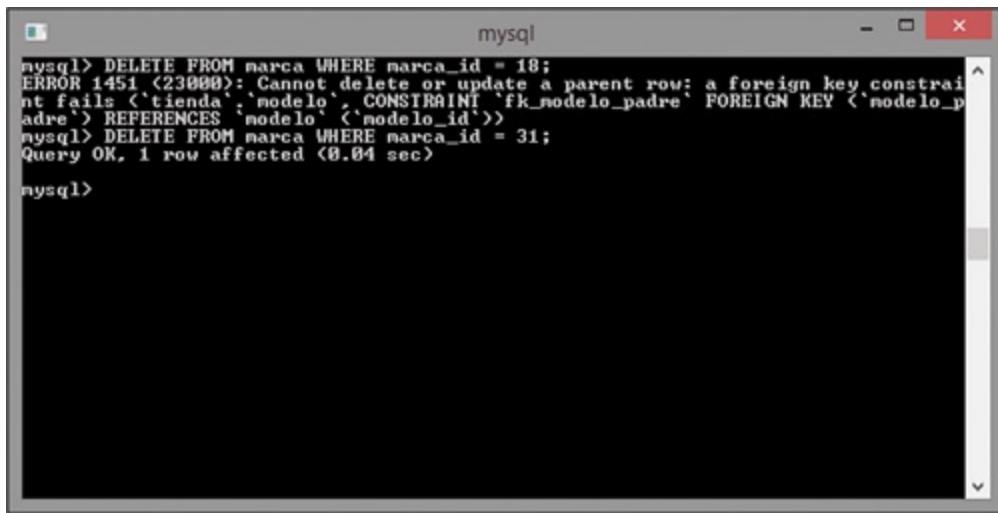
Lo habitual es dejar la opción como NO ACTION o como SET NULL. **ON DELETE CASCADE puede ser peligroso** si no se hace bajo control.

Para especificar esta opción se utiliza una sintaxis similar a la de las modificaciones. En caso de tener que especificar un ON UPDATE y un ON DELETE, el DELETE va primero. Ahora se mostrará un ejemplo haciendo que si

se borra una marca, los modelos se borren también:

```
ALTER TABLE modelo DROP FOREIGN KEY fk_marca;
ALTER TABLE modelo ADD CONSTRAINT fk_marca FOREIGN KEY (marca_id) REFERENCES marca(marca_id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

Aquí hay que tener cuidado. Hay una clave foránea en la tabla “coche” enlazando a la tabla de modelo. Esa clave foránea debería ser modificada también para tener un borrado en cascada. De lo contrario, solo pueden borrarse marcas que no tengan coches en esa tabla.



```
mysql> DELETE FROM marca WHERE marca_id = 18;
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key constraint fails ('tienda'.'modelo', CONSTRAINT 'fk_modelo_padre' FOREIGN KEY ('modelo_p
adre') REFERENCES 'marca' ('marca_id'))
mysql> DELETE FROM marca WHERE marca_id = 31;
Query OK, 1 row affected (0.04 sec)

mysql>
```

No podría borrarse SEAT porque tiene coches. Podría borrarse Volkswagen porque no tiene coches matriculados.



Actividades

29. No podemos permitir que se borren las reservas porque se necesitan informes de ingresos. Sin embargo, que los hoteles dejen de existir y los tengamos que borrar. Para ello, haga que la clave foránea que hay en la tabla “reserva” admita nulos.
30. Una vez la clave foránea admite nulos, haga que cuando se elimine un hotel, sus referencias pasen a tener valor NULL.
31. Pruebe a borrar un hotel y lea los resultados de la tabla “reserva”.

Restricciones de tipo de dato

Otra de las restricciones que se incluyen son las de no permitir introducir **datos inválidos** en una columna. Si una columna permite números enteros no debe ser posible añadir cadenas de texto en ellas.

MySQL es especialmente permisivo en este aspecto. Permite hacer cosas que parecen muy incorrectas pero que no dan error.



Nota

En *MySQL* se puede insertar prácticamente cualquier cosa en cualquier columna. Es el propio SGBD el que hace conversiones internas para hacer que se pueda. Otros SGBD son mucho más restrictivos y no permitirían realizar la inserción.

Las restricciones básicas son estas:

- Insertar un valor NULL en un campo NOT NULL dará error y no se permite insertar el registro.
- Si una cadena de caracteres es más larga de lo que permite el campo, se cortará.
- Si un número es más grande de lo que admite el campo se almacenará el valor más grande posible.
- Si se almacena una cadena de texto que empieza por un número en un campo numérico, se introduce el número correctamente. De lo contrario se añade un cero.
- Es posible introducir fechas incorrectas en campos DATE y DATETIME. *MySQL* acepta muchos valores que no son correctos pero que puede llegar a entender. En caso de ser totalmente incorrecta, almacenaría 0000-00-00 00:00:00.
- Si no se añade valor para una columna, se intenta utilizar el valor por defecto del campo. En caso de no existir, introducirá el valor por defecto implícito del tipo de dato (0 para números, cadena vacía para texto, etc.).

MySQL se puede configurar en **modo estricto** para que no se realicen estas conversiones y solo se permitan valores 100% correctos, aunque no es lo habitual hacer eso.

Restricciones de campos ENUM y SET

En los campos de tipo ENUM todos los registros deben tener un valor comprendido entre los valores del ENUM o NULL si se permiten nulos. Si no se especifica ningún valor, el primer valor posible de ENUM será utilizado.

Si se especifica un valor incorrecto se utiliza un valor interno que se corresponde a la cadena vacía.

No es posible bajo ningún concepto contar con un valor que no esté dentro de la enumeración.

En los campos de tipo SET simplemente se ignoran los valores inválidos y solo se consideran los valores válidos.

10.2. Disparadores

Un disparador (*TRIGGER*, en inglés) es un **objeto asociado a una tabla** de base de datos que se ejecuta cuando ocurre un **evento sobre esa tabla**.



Ejemplo

Puede haber un disparador que se ejecute antes de insertar una fila para validar los datos.

Los disparadores se pueden ejecutar **antes o después** de una operación de inserción, modificación o borrado. Algunos casos en los que pueden ser útiles son los siguientes:

- Crear un histórico de operaciones de modificación de datos con fines de auditoría. Antes de modificar las filas, pueden guardarse en un histórico los datos anteriores, fecha de modificación, etc.
- Los disparadores pueden cambiar los valores antes de insertarlos o modificarlos en la tabla. Por ello, se pueden filtrar y validar los datos de la manera que se quiera.
- Como se pueden ejecutar antes de cambiar los valores, es posible asignar valores por defecto complejos a algunas columnas. En MySQL el valor por defecto de una columna debe ser un valor normal a excepción de las columnas con DEFAULT CURRENT_TIMESTAMP. Gracias a los disparadores podría hacerse que el valor por defecto sea calculado mediante una función o mediante operaciones matemáticas. Por ejemplo, hacer que el valor por defecto de una

columna sea una cadena de caracteres aleatoria.

Va a mostrarse un ejemplo creando un disparador sobre la tabla marca que haga que los valores de marca_nombre siempre estén en mayúsculas, sin importar lo que el usuario inserte:

```
DELIMITER //

CREATE TRIGGER update_nombre_mayusculas BEFORE UPDATE ON marca
FOR EACH ROW BEGIN
    SET NEW.marca_nombre = UPPER(NEW.marca_nombre);
END
//

DELIMITER ;
```

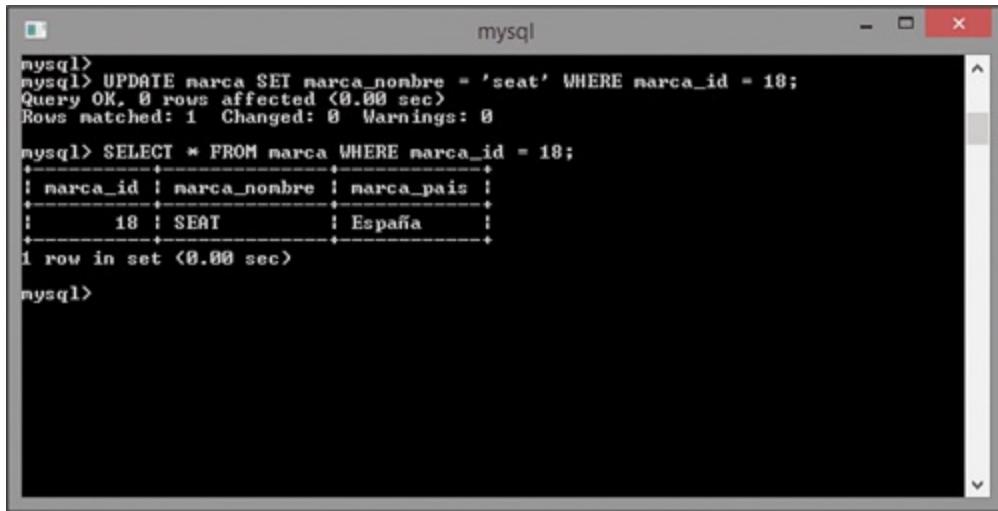
A continuación, se explicará paso a paso la sintaxis y el funcionamiento de este código:

- Cada trigger debe tener un nombre único en la base de datos.
- Hay que explicar el concepto **delimitador**. Por defecto, cuando una sentencia acaba, termina en punto y coma. En un trigger es posible tener muchas sentencias dentro. Debe cambiarse el delimitador por defecto temporalmente para que puedan utilizarse puntos y comas dentro del código. Para ello, primero se cambia el delimitador a “//”. Se utiliza un delimitador que no sea usado en absoluto en SQL. Podría ser “|”, “@” o cualquier símbolo poco común. Cuando se termina de crear el trigger, se cambia al delimitador por defecto, de lo contrario las consultas siguientes tendrían que ser finalizadas con “//” en vez de con punto y coma. Esto solo es válido para la sesión de usuario actual.
- Para crear un trigger se empieza con CREATE TRIGGER seguido del nombre (único y sin caracteres especiales). A continuación, se dice cuándo y qué evento quiere controlarse. En este caso se quiere controlar el evento que ocurre **antes** de modificar filas en la tabla “marca”.
- A continuación, se añaden las palabras **FOR EACH ROW BEGIN**. Esto quiere decir que la operación se ejecuta sobre cada fila. Si se hace un UPDATE que modifique muchas filas, el trigger se ejecuta una vez por cada fila. Debe escribirse porque es el estándar. Otros SGBD soportan otros sistemas además del FOR EACH ROW.
- Entre BEGIN y END es donde se añade el código personalizado.
- Pueden utilizarse los valores anteriores y nuevos de cada columna: NEW.

columna o OLD.columna. En inserciones solo se cuenta con NEW, en borrados, solo con OLD. En modificaciones, con ambos. NEW se refiere al nuevo valor y OLD al existente.

Dentro del código del trigger se introduce **código de procedimiento almacenado**. Cada SGBD soporta un conjunto de operaciones. Esto es un lenguaje de programación en sí mismo con una sintaxis inspirada en el SQL. En este libro no puede profundizarse mucho en los procedimientos almacenados. Además, en MySQL son muy diferentes a los de Oracle o SQL Server. Simplemente, se mostrarán unos cuantos ejemplos, pero para un uso completo haría falta un mayor estudio sobre el tema.

En este caso lo que se hace es convertir a mayúscula el valor de marca_nombre. Para ello, se utiliza SET seguido del valor a modificar. En este caso quiere modificarse el nuevo valor de marca_nombre. Se probará a modificar el nombre de SEAT para que esté en minúsculas, se comprobará que no es posible:



```
mysql> UPDATE marca SET marca_nombre = 'seat' WHERE marca_id = 18;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 1  Changed: 0  Warnings: 0

mysql> SELECT * FROM marca WHERE marca_id = 18;
+-----+-----+-----+
| marca_id | marca_nombre | marca_pais |
+-----+-----+-----+
|     18   | SEAT        | España      |
+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

Los triggers cuentan con una serie de restricciones y condiciones que pueden desaconsejar su uso. Esos inconvenientes se indican a continuación (sobre todo relacionados con MySQL):

- En MySQL, cuando se realizan modificaciones en cascada fruto de una clave foránea (ON UPDATE CASCADE), los triggers de modificación sobre la tabla **no son lanzados**. Es una restricción que está tratando de ser resuelta pero que de momento no está arreglada.
- No es posible tener más de un trigger que se ejecute en el mismo momento en la misma tabla. No es posible tener dos que se ejecuten BEFORE INSERT sobre la misma tabla. Sí es posible tener más de uno pero que se ejecuten en

momentos distintos.

- Los disparadores pueden crear confusión si no están muy bien documentados. Se realizan conversiones sobre los datos, las cuales pueden no ser conocidas por los usuarios de la base de datos. Es importante explicar qué triggers hay y qué hace cada uno a cada usuario de la base de datos.
- Pueden ser fruto de problemas de rendimiento en la base de datos. Si hay que realizar muchas comprobaciones cada vez que una tabla recibe modificaciones, el rendimiento cae en picado. En vez de ejecutar una sentencia, se ejecutan por detrás un montón de ellas.
- Si están mal desarrollados pueden introducir errores en cadena que dejen inutilizada la base de datos temporalmente. Hay que ser extremadamente cuidadosos al crear un disparador y probarlo a conciencia antes de aplicarlo sobre tablas reales. Cualquier error, por pequeño que sea, afectará a los datos de una u otra manera; aspecto que siempre se debe evitar.
- Los triggers, como todos los procedimientos almacenados, utilizan una extensión de SQL que es muy poco estándar. Mientras que más del 90% de las consultas de ejemplo que se han visto a lo largo del capítulo son estándar y funcionarían en cualquier SGBD, es prácticamente imposible que un trigger funcione en cualquier SGBD simultáneamente a menos que sea extremadamente sencillo.



Consejo

Cuando cree un disparador, escriba la documentación del mismo, explicando su nombre, su comportamiento, cuándo se ejecuta y sobre qué tabla. Todos los usuarios de la base de datos deben tener acceso a esa documentación. Si desconocen su funcionamiento, es posible que hagan algo que no le devuelva los resultados esperados.

Antes se ha visto cómo hacer que sea imposible cambiar el nombre de tal manera que no esté en mayúscula. Esto solo funcionaría al modificar filas, porque solo se ha hecho para eso. Para hacer que se ejecute antes de insertar filas se tiene que crear un disparador nuevo con otro nombre pero con idéntico código dentro del BEGIN END:

```
DELIMITER //  
  
CREATE TRIGGER inserta_nombre_mayusculas BEFORE INSERT ON marca  
FOR EACH ROW BEGIN  
    SET NEW.marca_nombre = UPPER(NEW.marca_nombre);  
END  
//  
  
DELIMITER ;
```

Ahora es imposible crear marcas en minúsculas.

Borrar disparadores

No es posible **desactivar triggers** temporalmente en *MySQL*, solamente pueden borrarse completamente y volverse a crear más adelante. Para borrar uno se utiliza **DROP TRIGGER** seguido del nombre del mismo.

Validar datos con un trigger

En caso de querer validar que sea imposible introducir una marca de coche de tres caracteres o menos, no sería posible añadirla mediante un **CREATE TABLE**, por lo que se necesitaría un trigger que lo realizara.

Lo que se hará es borrar el trigger de “*inserta_nombre_mayusculas*” para darle otro nombre y que incluya esta modificación. Recuerde que no es posible tener más de un disparador que se ejecute en el mismo evento:

```
DROP TRIGGER inserta_nombre_mayusculas;

DELIMITER //

CREATE TRIGGER pre_inserta_marca BEFORE INSERT ON marca
FOR EACH ROW BEGIN
    SET NEW.marca_nombre = UPPER(NEW.marca_nombre);

    IF LENGTH(NEW.marca_nombre) <= 3 THEN
        SIGNAL SQLSTATE '45000' SET message_text =
            'No es posible crear una marca de menos de 3 caracteres';
    END IF;
END
//

DELIMITER ;
```

Para validar puede utilizarse el IF, el cual comienza con IF, seguido de la comprobación (con mismo formato que en un WHERE de una SELECT) seguido de la palabra THEN. Entre el THEN y el END IF es donde se añade el código que debe ejecutarse si se cumple la condición.



The screenshot shows a terminal window titled "mysql" running on a Windows operating system. The command entered is:

```
mysql> INSERT INTO marca (marca_id, marca_nombre, marca_pais) VALUES (5, 'kia', 'Corea');
```

An error message is displayed:

```
ERROR 1644 <45000>: No es posible crear una marca de menos de 3 caracteres
```



Importante

Para provocar un error hay que añadir la línea del ejemplo. En message_text puede personalizarse el mensaje mostrado al usuario. El código 45000 es siempre el mismo.

No permite insertar una fila con tres letras.

Los disparadores tienen un lenguaje propio. Aquí solamente se han visto validaciones muy básicas y modificaciones sobre las filas. Si se quiere aprender más en profundidad habría que ir al manual de cada SGBD. Aquí se trata de explicar el SQL estándar orientado hacia *MySQL* y esta es una característica muy personalizada por cada SGBD.



Actividades

32. Cree un disparador que no permita que el teléfono y el correo electrónico de los clientes se deje vacío. Debe tener al menos 5 caracteres.
-

10.3. Gestión de permisos en tablas

Siempre se ha mencionado que una de las grandes bondades de los SGBD era la gestión de permisos, que permitía que determinados usuarios accedieran solo a **algunas áreas de la base de datos**. En este apartado se estudiará cómo asignar permisos a los usuarios.

Cada SGBD suele permitir controlar el acceso a unas u otras secciones, aunque todas siguen una sintaxis similar. La idea es asignar permisos a usuarios para **acceder a objetos** y realizar determinadas **operaciones**. Por ejemplo, permitir que el usuario “paco” acceda a la tabla marca para realizar lecturas de datos pero que no pueda hacer modificaciones.

Gestión de usuarios

El sistema de permisos se basa en usuarios. Son los usuarios los que tienen permisos a realizar determinadas acciones. Todo SGBD cuenta con un usuario *root* que tiene acceso a todas las secciones de la base de datos y que cuenta con la posibilidad de crear nuevos usuarios.

Es el DBA el que sabrá la contraseña del usuario *root*. Los clientes de la base de datos se conectarán mediante su usuario y contraseña. Es el DBA el que se encarga de que los usuarios tengan los permisos adecuados.

Para crear un usuario se utiliza la sentencia CREATE USER. Un usuario tiene un nombre y una contraseña. Pueden crearse usuarios sin contraseña pero está totalmente contraindicado. Así se crea un usuario de nombre “paco”:

```
CREATE USER 'paco'@'localhost' IDENTIFIED BY 'mipassword';
```

Primero, se indica el nombre del usuario seguido de arroba y luego el servidor desde el cual se conectará. En *MySQL*, un usuario se compone de **un nombre y un servidor**. Puede haber más de un usuario con el mismo nombre pero con diferente servidor. Para todas las pruebas se utilizará el servidor localhost, que se refiere a la máquina actual.

La contraseña se indica entre comillas simples. Puede contener caracteres especiales y espacios.



Importante

El uso de caracteres especiales y espacios es recomendable porque aumenta la complejidad de la contraseña.

Para borrar un usuario se usa DROP USER. Vea el ejemplo pero no lo ejecute, porque se seguirán haciendo demostraciones con el usuario “paco”:

```
DROP USER 'paco'@'localhost';
```

También puede ser útil cambiar de nombre a un usuario. Para ello, se utiliza RENAME USER (no ejecute la sentencia):

```
RENAME USER 'paco'@'localhost' TO 'paco2'@'localhost';
```

Primero, se especifica el nombre de usuario actual seguido de TO y después el nombre del usuario.

Es muy común también tener que reiniciar la contraseña de un usuario, bien porque la han olvidado, bien por motivos de seguridad que llevan a que se actualicen las contraseñas cada cierto tiempo. Para ello, se utiliza SET PASSWORD. Se hará que la contraseña de “paco” sea “entr4r”:

```
SET PASSWORD FOR 'paco'@'localhost' = PASSWORD('entr4r');
```

La sintaxis se sale un poco de lo habitual. Empieza con SET PASSWORD FOR seguido del nombre del usuario existente. Luego, un signo igual y la asignación del valor utilizando la función PASSWORD —la cual sirve para encriptar el valor—. SET PASSWORD espera una contraseña codificada de forma segura por defecto, por lo que hay que utilizar esta función para codificarla.



Nota

Por defecto, es posible que la cuenta *root* de *MySQL* tenga una contraseña vacía. Esto no es un gran problema en una máquina local a la que nadie se conecte, pero es inaceptable en un servidor real. Puede modificar la contraseña de *root* utilizando esta misma sintaxis. Se utiliza ‘*root*@’localhost’ como nombre de usuario.

Gestión de permisos

La gestión de permisos va englobada dentro del DCL (*Data Control Language*, la vertiente de SQL dedicada a la gestión de permisos y usuarios).

Una vez visto cómo crear, modificar y eliminar usuarios, llega el momento de dar utilidad a esos usuarios. Todo el sistema de permisos se basa en que los usuarios tienen permiso para acceder a objetos del SGBD. En *MySQL* estos objetos pueden ser bases de datos, tablas, columnas o procedimientos almacenados.

Los permisos se aplican en cascada. Es decir, si se dan permisos de lectura en una base de datos a un usuario, por defecto ese usuario podrá leer todas las tablas de esa base de datos.

Para dar permisos se utiliza la sentencia GRANT. Para eliminar permisos se recurrirá a REVOKE. Con GRANT hay que especificar el usuario, el objeto al que se hace referencia y el tipo de permiso que se le va a dar.



Ejemplo

Se puede dar acceso de solo lectura, permitir inserciones pero no borrados, etc.

A continuación, se describen los tipos de objeto a los que se les puede dar permisos:

- Nivel global: dar permisos a un usuario para acceder a todos los objetos del SGBD. En *MySQL* un usuario puede tener acceso a todas las bases de datos del sistema. No suele ser muy recomendable utilizar esta opción.
- Base de datos: los permisos se aplican a todos los objetos dentro de una base de datos concreta. Esta opción sí es muy habitual.
- Tablas: se aplican los permisos a tablas concretas y a todas sus columnas.
- Columnas: permisos aplicables a columnas de tablas concretas. Es útil cuando un usuario no debe poder acceder a columnas con información sensible.
- Procedimientos almacenados: permisos para crear y ejecutar procedimientos almacenados tales como disparadores, eventos, funciones, etc.

La lista de permisos que es posible dar es la siguiente:

Permiso	Significado
ALL [PRIVILEGES]	Permisos para realizar todas las acciones posibles.
ALTER	Permite modificar tablas mediante ALTER TABLE.
CREATE	Permite el uso de CREATE TABLE
CREATE USER	Permite gestionar usuarios. Crearlos, borrarlos y renombrarlos.
CREATE VIEW	Permite crear vistas con CREATE VIEW.
DELETE	Permite borrar filas de una tabla.
DROP	Permite borrar tablas.

INSERT	Permite insertar filas en una tabla.
SELECT	Permite ver los datos de una tabla.
UPDATE	Permite modificar filas de una tabla.
USAGE	Significa que no tiene permisos. Puede conectarse pero no puede hacer nada.
GRANT OPTION	Permite propagar los permisos que un usuario tiene a otros usuarios de la bases de datos.



Nota

Se han omitido los permisos que no se utilizan normalmente.

Ya que se ha introducido el funcionamiento, se empezarán a ejecutar las primeras sentencias GRANT para probar. Se dará acceso a la base de datos “tienda” al usuario “paco”. Solo tendrá permiso de leer e introducir datos, no podrá ni borrar ni modificar filas:

```
GRANT SELECT, INSERT ON tienda.* TO 'paco'@'localhost';
```

Se comienza con la palabra **GRANT**, seguida de la lista de permisos que se le van a dar separados por comas. Luego viene la palabra **ON** seguida de los objetos aplicables separados por comas (en este caso solo uno). Finalmente, se utiliza la palabra **TO** seguida del nombre de usuario.

Cuando se dan privilegios a nivel de base de datos, se utiliza su nombre seguido de punto y asterisco.

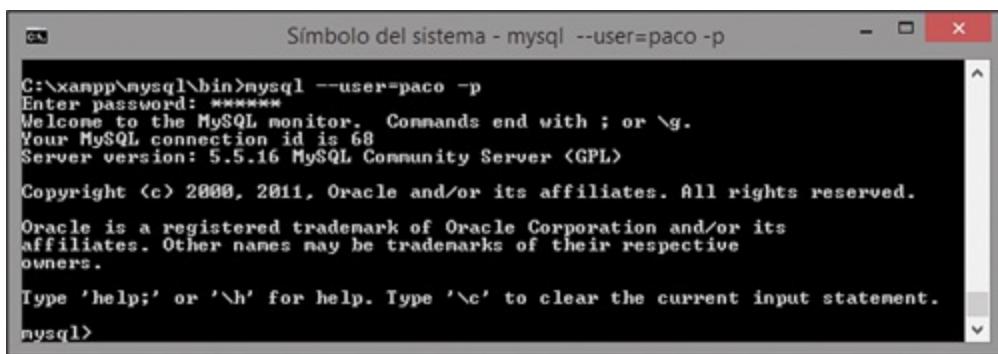


Importante

Esto quiere decir que se da acceso a la base de datos y a todas sus tablas.

Podría utilizarse `*.*` en caso de querer dar acceso a todas las bases de datos del sistema (incluidas las que se creen a posteriori).

Para probar estos cambios, se conectará con el usuario “paco”, recordando que su contraseña es “entr4r”. Existen muchas formas de conectarse a *MySQL*. Para estos ejemplos se utilizará el cliente de *MySQL* por línea de comandos, que está disponible en cualquier momento. Para utilizar un usuario específico, se invocará al comando con el parámetro user:



```
C:\xampp\mysql\bin>mysql --user=paco -p
Enter password: *****
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 68
Server version: 5.5.16 MySQL Community Server (GPL)

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

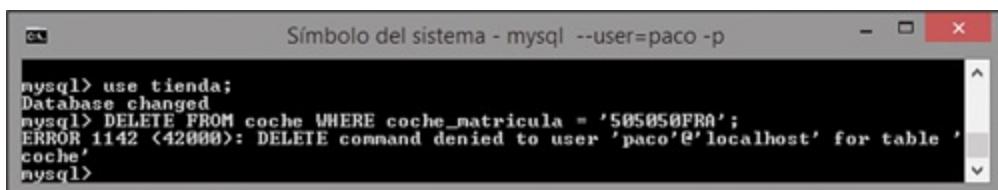
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Se utilizará el parámetro `--user=` seguido del nombre de usuario. Al estar en la máquina local, se utilizará el usuario ‘paco’@‘localhost’. Tras el nombre, se utiliza `-p`. Esto hará que se pida la contraseña antes de conectar. Podría utilizarse el parámetro `-password` seguido de la contraseña, pero es mucho más seguro introducirla manualmente, porque en ningún momento se hace visible.

Una vez introducido el password ya se accede al sistema.

Si se intenta borrar el coche de matrícula “505050FRA”, se verá que no es posible:



```
mysql> use tienda;
Database changed
mysql> DELETE FROM coche WHERE coche_matricula = '505050FRA';
ERROR 1142 (42000): DELETE command denied to user 'paco'@'localhost' for table 'coche'
mysql>
```

MySQL devuelve un error diciendo que el comando se deniega. Esto se debe a que se carece de los permisos. Sin embargo, sí podría crearse un nuevo coche. Puede probarse introduciendo un Audi R8:

```
Símbolo del sistema - mysql --user=paco -p
mysql> INSERT INTO coche (coche_id, modelo_id, coche_matricula, coche_fecha_matrículacion) VALUES (20, 33, '8552330CM', '2013-11-08');
Query OK, 1 row affected (0.03 sec)
mysql>
```

Esta operación sí está permitida.

Para dar permisos a nivel de tabla se utiliza una sintaxis muy similar, pero reemplazando el asterisco por el nombre de la tabla. Si se hace que el usuario sí tenga permiso para insertar y borrar filas de la tabla coche, el ejemplo que antes falló, ahora funcionaría:



Hay que ejecutar esta sentencia como **root**. Si se está conectado como “paco” no es posible porque no puede darse más permisos a sí mismo.

Ahora, pruebe el ejemplo anterior estando conectados como “paco”:

```
Símbolo del sistema - mysql --user=paco -p
mysql> DELETE FROM coche WHERE coche_matricula = '505050FRA';
Query OK, 1 row affected (0.03 sec)
mysql>
```

Lo mismo que se hizo antes, ahora sí funciona.

GRANT OPTION

Es posible dar permiso a un usuario para propagar sus permisos a otros usuarios; esto hay que hacerlo de forma controlada. Un usuario puede dar permiso para realizar las mismas operaciones que él está autorizado a efectuar. Para utilizar esta opción, al final de la sentencia GRANT se utiliza la frase WITH GRANT OPTION.

Revocar permisos

Además de poder dar permisos a usuarios, es posible eliminar esos permisos. Así se elimina el permiso que se dio antes:

```
REVOKE UPDATE, DELETE ON tienda.coche FROM 'paco'@'localhost';
```

La sintaxis es similar, en vez de GRANT se usa REVOKE y en vez de la palabra TO se utiliza **FROM**; si se intenta eliminar un privilegio que no se tiene, se obtiene un error.

Es posible utilizar ALL PRIVILEGES o GRANT OPTION a fin de eliminar todos los privilegios o quitar la opción de dar permisos a otros usuarios sobre ese objeto concreto.

Una vez se eliminan los permisos, si hay permisos de nivel superior, se aplicarán.



Ejemplo

Si se quitan todos los privilegios sobre la tabla “coche”, aún podría realizarse SELECT e INSERT, porque se tiene permiso para hacerlo en todas las tablas de la base de datos “tienda”.



Actividades

33. Cree un usuario de nombre “pepe”. Dele la contraseña que desee.
34. Haga que ese usuario solo pueda leer las tablas de hoteles y la vista de reservas para este año. Las demás tablas deben estar invisibles y las que estén disponibles deben ser de solo lectura.
35. Cámbiele la contraseña a ese usuario por otra distinta.



Aplicación práctica

Como administrador de bases de datos de la empresa, se le pide que gestione los permisos de los usuarios de la misma. Se tiene una base de datos con las siguientes tablas, pero sin gestión de permisos:

- Tabla “facturas”.
- Tabla “servicios”.
- Tabla “factura_servicio” (cada servicio de una factura).

Hay que gestionar los siguientes permisos:

- Los usuarios de tipo “empleado” solo pueden leer los datos de las tres tablas, no pueden realizar modificaciones. Los usuarios de este tipo son “teresa”, “paco”, “Alberto” y “jose”.
- Los usuarios de tipo “comercial” pueden leer los datos de las tres tablas y además, pueden crear facturas (insertar/modificar/borrar de las tablas de facturas y factura_servicio). Los comerciales que hay son “jose_manuel”, “Isabel” y “Montse”.
- Los usuarios de tipo “contable” pueden modificar facturas y leer datos de las tres tablas, pero solo pueden modificar facturas por si hay errores, pero no pueden borrarlas o crear facturas nuevas. El único contable que hay es “pepe”.
- Los usuarios de tipo “director” pueden modificar y leer todas las tablas. No obstante, el privilegio de crear y modificar tablas no deben tenerlo (solo pueden modificar los registros, no la estructura de las tablas). El director es “jose_maria”.

Cree las consultas SQL necesarias para crear estos permisos.

SOLUCIÓN

```
GRANT SELECT ON facturas TO 'teresa', 'paco', 'Alberto', 'jose', 'jose_manuel', 'Isabel', 'Montse', 'pepe';
GRANT SELECT ON servicios TO 'teresa', 'paco', 'Alberto', 'jose', 'jose_manuel', 'Isabel', 'Montse', 'pepe';
GRANT SELECT ON factura_servicio TO 'teresa', 'paco', 'Alberto', 'jose', 'jose_manuel', 'Isabel', 'Montse', 'pepe';

GRANT INSERT, UPDATE, DELETE ON facturas TO 'jose_manuel', 'Isabel', 'Montse';
GRANT INSERT, UPDATE, DELETE ON factura_servicio TO 'jose_manuel', 'Isabel', 'Montse';

GRANT UPDATE ON factura_servicio TO 'pepe';

GRANT INSERT, SELECT, UPDATE, DELETE ON * TO 'jose_maria';
```

10.4. Optimización de consultas

El rendimiento en una base de datos es un apartado fundamental. Si el acceso a los datos es lento, se estará creando un **cuello de botella** que lastrará a todas las aplicaciones que utilicen la base de datos.

Cuando una tabla se hace muy grande (más de 100.000 registros o más de 50MB de espacio en disco), el acceso a los datos se hace más delicado. Si se lanza una consulta que exija examinar todas las filas de la tabla, el SGBD tendrá que realizar una serie de comprobaciones por cada registro. Cuantos más datos tenga que analizar, más lento funcionará.



Consejo

Hay que tratar de aplicar los principios de optimización de consultas desde el principio cuando las escriba, pero sin obsesionarse demasiado.

Si se está desarrollando una aplicación informática, hay que asegurarse primero de que funciona y más tarde habrá tiempo de optimizarla.

Esto es así porque es conveniente agilizar el proceso de desarrollo de software y porque es mucho más fácil detectar los problemas de rendimiento cuando se producen.

En una consulta que lea una tabla enorme para generar un informe mensual, el rendimiento no es demasiado importante. Si una consulta tarda un par de minutos en ejecutarse no debería suponer ningún problema.

Se puede programar ese informe para que ese ejecute durante la madrugada; no pasaría nada, porque es una consulta muy poco frecuente.

Sin embargo, hay consultas que se ejecutan cada minuto en las que el rendimiento es crucial. Por ejemplo, en un blog se realizan decenas de consultas de SQL cada vez que se carga la portada. Debe haber una consulta que liste las entradas más recientes, que cuente el número de comentarios de cada entrada, etc.

Por eso primero se debe identificar en qué entorno se ejecutará la consulta que se está escribiendo, para aplicar una serie de optimizaciones u otras. Por lo general, en consultas poco frecuentes no debe preocuparse de ese tema. Hay que centrarse en las

consultas que se ejecutarán en un uso normal de la aplicación.



Nota

Si la consulta que lista las entradas recientes tarda 30 segundos en ejecutarse, la portada tardaría mínimo 30 segundos en empezar a cargar. Se perderían muchos visitantes, el sitio sería inestable y habría muchos bloqueos en las tablas.

Tras comprobar que la consulta requiere ser rápida en cualquier circunstancia, debe pensarse en el **número de filas que se deberán leer** para poder satisfacer las condiciones especificadas en la consulta. Esto es lo que en mayor medida determina si una consulta se ejecutará rápido o no. Si una consulta solo necesita evaluar una fila para funcionar, no importa lo complejas que sean las operaciones matemáticas o la cantidad de funciones de SQL que se utilicen, porque siempre será bastante rápida por sí sola. En cambio una consulta con una subconsulta dentro, en la que se evalúen un millón de filas y en la que por cada fila se tengan que evaluar otras tantas filas de la subconsulta, seguirá una **progresión geométrica** en cuanto a tiempo de ejecución y consumo de recursos del sistema.

Básicamente, toda la gestión de la optimización se basa en tener un esquema mental del número de filas que se deberán evaluar. No se debe pensar en el número de filas a evaluar en este preciso instante de tiempo, sino **ahora y en el futuro**. Es habitual encontrarse con aplicaciones que se **vuelven cada día más lentas** y esto es debido a que cada vez tienen que trabajar con más datos y no están preparadas para ello.

Las funciones de ordenación de resultados de eliminado de duplicados mediante SELECT DISTINCT y las agrupaciones son **delicadas en cuanto al rendimiento**. Al ordenar datos primero se deben evaluar todas las filas, y una vez se tenga la lista de filas que cumplen con las condiciones, se tiene que operar sobre ellas para ordenarlas. En una eliminación de duplicados se tienen que calcular todos los resultados y, a continuación, detectar las duplicidades. En una agrupación también se tiene que leer la lista entera de filas afectadas y luego realizar la serie de operaciones requeridas.

Estas funciones pueden demandar que el SGBD cree tablas temporales internamente para poder trabajar con las filas devueltas. Normalmente, la creación de tablas temporales añade tiempo de ejecución extra a las consultas.

Lo que se utiliza para garantizar un acceso a datos rápido con los datos actuales y futuros son **los índices**.



Recuerde

Los índices, tal y como se ha visto, consisten en estructuras de datos que permiten determinar la localización de una fila sin tener que ir buscando una por una.

Cabe destacar que en *MySQL* (y por norma general) cada vez que se define una **clave foránea en una columna**, esa columna pasa a estar **indexada**. Si se quisieran listar los modelos que son de la marca SEAT, el SGBD podrá saber cuáles son esas filas sin tener que comprobarlas una a una. Utilizaría el índice para ver qué modelos tienen el marca_id 18 e iría a la tabla “modelo” para devolver los datos de cada uno. Los modelos que no tengan marca_id 18 no serían utilizados en absoluto.



Importante

Los índices únicos también se comportan como índices normales y aumentan el rendimiento cuando se filtra por su valor.

La clave primaria suele ser el **índice de mayor fuerza** en una tabla. Cuando se filtran los registros de una tabla según el valor de su clave primaria, el rendimiento suele ser inmejorable.

A la hora de optimizar una consulta hay que tratar de utilizar índices, puesto que reducen el número de filas a evaluar. El rendimiento ideal se produce cuando el **número de filas a evaluar es igual al número de filas retornadas** por el SGBD, aunque no siempre esto es posible. Se pueden ejecutar consultas óptimas que evalúen más filas de las necesarias. Vea una serie de ejemplos:

```
/* Lento. Evalúa todo: */
SELECT * FROM modelo WHERE modelo_precio > 15000;

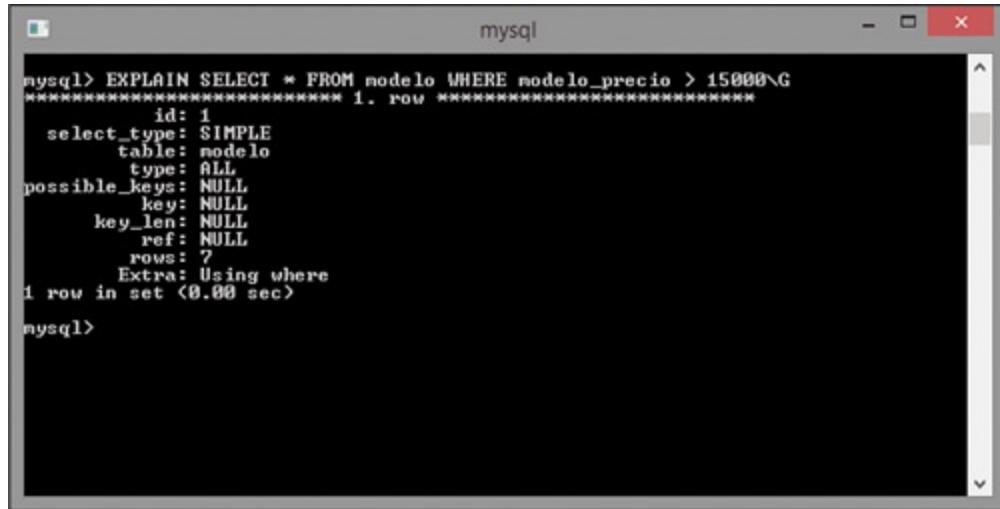
/* Más rápido, porque evaluamos solo las filas de Renault: */
SELECT * FROM modelo WHERE marca_id = 20 AND modelo_precio > 15000;

/* Rendimiento máximo. Evaluamos lo mismo que obtenemos: */
SELECT * FROM modelo WHERE marca_id = 20;
```

Se ven ejemplos de distintas filtraciones que tendrían distinto rendimiento. En la tabla de ejemplo jamás se verían diferencias de rendimiento porque es diminuta. Hay que pensar a largo plazo en qué pasaría si la tabla tuviera miles de filas o más. En este ejemplo, si se añadiera un índice sobre la columna `modelo_precio`, las dos primeras consultas se convertirían en 100% óptimas porque solo se evaluarían las filas concretas.

Para conocer la manera que tiene una consulta de ejecutarse, se coloca la palabra **EXPLAIN** delante del `SELECT`. Cada SGBD proporciona un funcionamiento distinto para el EXPLAIN, pero su utilidad básica es la de conocer el plan de ejecución de una consulta.

Este es el plan de ejecución de la primera consulta anterior:



The screenshot shows a terminal window titled "mysql" running on Windows. The command entered is "EXPLAIN SELECT * FROM modelo WHERE modelo_precio > 15000\G". The output is as follows:

```
mysql> EXPLAIN SELECT * FROM modelo WHERE modelo_precio > 15000\G
+-----+-----+
| id: 1 |      |
| select_type: SIMPLE |      |
|   table: modelo |      |
|       type: ALL |      |
| possible_keys: NULL |      |
|       key: NULL |      |
|    key_len: NULL |      |
|       ref: NULL |      |
|      rows: ? |      |
|     Extra: Using where |      |
+-----+-----+
1 row in set (0.00 sec)

mysql>
```

Se ha terminado la consulta con \G en vez de con punto y coma. Esto es algo especial del cliente por línea de comandos de *MySQL*. Si se utiliza \G los resultados, en vez de en una tabla, se **muestran en vertical**; esto es útil cuando los resultados no caben en pantalla.

Debe prestarse atención sobre todo en las columnas `type`, `rows` y `Extra`. En el tipo se dice la manera de acceder a los datos, en este caso es `ALL`, que significa un chequeo completo de la tabla. En `rows` se indica el número de filas que debe evaluar para poder determinar qué filas devolver. En `Extra` se suele proporcionar información extra sobre la consulta.

Es habitual ver la frase “Using where”, esto quiere decir que se evalúan el número de filas que se dice en “`rows`”, y por cada fila, se aplica una validación simple para ver

si cumple la condición o no.

Otra palabra que es habitual encontrarse es “Using temporary”, que quiere decir que se necesita crear una tabla temporal para satisfacer las condiciones.



Nota

Normalmente, una consulta que revise todas las filas y además necesite una tabla temporal tendrá un rendimiento pésimo cuando haya un gran volumen de datos.

El uso de EXPLAIN es bastante avanzado y en cada SGBD funciona de una manera distinta. Es conveniente leer la documentación del mismo para aprovechar al máximo sus ventajas.

11. Resumen

El SQL (*Structured Query Language*) es un lenguaje inspirado en el lenguaje natural en inglés que se utiliza para interactuar con sistemas gestores de bases de datos. Es un estándar. Todos los SGBD lo implementan y adaptan para satisfacer sus requisitos propios. Este libro se centra en el SQL orientado hacia MySQL porque el SQL estándar por sí solo no tiene una aplicación directa. No hay ningún sistema que lo implemente como forma única de funcionamiento.

Con el SQL puede definirse un modelo de datos, creando bases de datos con CREATE DATABASE y definiendo las tablas mediante CREATE, ALTER y DROP TABLE.

Una vez definido el modelo de datos puede procederse a manipular sus datos. Para ello se cuenta con las sentencias INSERT, UPDATE y DELETE, que permiten insertar, modificar y eliminar filas.

Cuando las tablas contienen información es posible lanzar consultas SELECT sobre la base de datos para obtener los datos que se deseen. Es habitual realizar uniones entre tablas y filtraciones para obtener las filas y columnas que se necesiten en la aplicación.

Además, el SQL proporciona métodos más avanzados para obtener datos, como son las subconsultas, que permiten anidar unas consultas dentro de otras para aplicar

filtraciones complejas u obtener datos de otras tablas. También pueden realizarse agrupaciones de datos, muy útiles para generar informes que de otra manera resultarían bastante complejos de realizar.

El SQL también cuenta con una sintaxis que permite gestionar los permisos de usuarios de la base de datos, dando acceso solo a determinadas áreas de la misma.



Ejercicios de repaso y autoevaluación

1. Cree una base de datos de nombre “flota”.

2. Cree tres tablas, de nombres “vehiculo”, “conductor” y “vehiculo_conductor”

Un vehículo puede ser conducido por varios conductores. Se necesita guardar la fecha en la que cada conductor empieza a conducir ese vehículo.

La tabla vehículo cuenta con los campos matrícula (clave primaria), número de bastidor (clave única), una marca y un modelo. Además, tiene un campo con la fecha de la próxima revisión. Puede estar nula, en caso de no saberse.

La tabla de conductores tiene los campos nombre completo, ID autonumérico, fecha de caducidad de su permiso de conducir y el nombre de su puesto en la empresa.

La tabla vehiculo_conductor es una tabla fruto de una relación de muchos a muchos. Contiene las claves foráneas de “vehiculo” y “conductor”, junto a un campo de fecha que guarda el dato de cuándo ese conductor empieza a utilizar ese vehículo. La clave primaria la forman la matrícula y el ID autonumérico de la tabla vehículo.

Las tablas deben estar en InnoDB.

- 3. Modifique la tabla de vehículos y añada un índice simple que englobe las columnas de marcas y modelos. Se utilizará para agilizar las búsquedas más adelante.**

- 4. Inserte registros de prueba en las tres tablas. Utilice sentencias INSERT extendidas (de varias filas por sentencia). Que haya al menos cuatro vehículos y cuatro conductores. Haga que no todos los conductores hayan empezado a usar algún vehículo, solo algunos.**

- 5. Modifique el conductor cuyo valor de la clave primaria sea 1. Aumente su fecha de validez del carnet de conducir en 10 años (no lo haga manualmente, sino utilizando DATE_ADD).**

- 6. Escriba una consulta que devuelva todas las marcas de vehículo existente. Sin duplicados.**

- 7. Escriba una consulta que muestre los conductores por orden de fecha de validez del permiso de conducir. Que muestre los 2 conductores que primero vayan a tener el permiso caducado.**

- 8. Escriba una consulta que muestre los conductores que tengan de apellido “Gómez” (puede no haber ninguno).**

9. Escriba una consulta que devuelva las columnas: matrícula, fecha de inicio de utilización del vehículo y nombre del conductor.

10. Escriba una consulta que muestre la matrícula de cada vehículo junto al número de conductores diferentes que lo han utilizado. Si nadie lo ha utilizado, que muestre un cero.

11. Escriba una consulta que muestre los datos de los vehículos que tengan que pasar la revisión antes de la media (pista: utilice DATEDIFF).

12. Escriba una consulta que muestre los datos de los vehículos que tengan que pasar la revisión antes de la media de su marca.

13. Cree una vista que contenga únicamente los vehículos que no han sido utilizados por nadie.

14. Cree un usuario de nombre ‘alex’ que tenga permiso para ver todas las tablas de la base de datos ‘flota’ pero que solo pueda ver los datos, que no pueda modificar nada.

-
-
-
- 15.** Escriba una consulta que muestre los conductores cuya fecha de caducidad del carnet de conducir sea mayor a 2015. A continuación, razoné cómo se podría mejorar el rendimiento de la consulta en caso de que la tabla tuviera miles de registros.

Capítulo 4

Lenguajes de marcas de uso común en el lado servidor

1. Introducción

Cuando varias aplicaciones o componentes informáticos necesitan comunicarse entre sí, han de contar con un **formato común** de tal manera que ambas partes puedan entenderse. Esas aplicaciones podrían comunicarse mediante mensajes de texto, mediante archivos binarios o cualquier formato que ambos entendieran. Lo importante es que todos los extremos conozcan el formato.

Los lenguajes de marcas se crearon para **unificar en un formato común** el intercambio de datos entre distintas aplicaciones. Consisten en ficheros de texto con un determinado formato. Su principal característica es que son legibles por computadores y por personas al mismo tiempo, cosa no muy fácil de conseguir.

Los lenguajes de marcas más conocidos son el HTML, el XML y LaTeX; este capítulo se centrará en el XML.

2. Origen e historia de los lenguajes de marcas. El estándar XML

Los lenguajes de marcas son un formato para **codificar documentos** que permite incorporar etiquetas que aportan más información sobre cada elemento. He aquí un ejemplo de documento en el lenguaje de marcado HTML para su mejor comprensión:

```
<p class="destacado">
    Los <strong>lenguaje de marcas</strong> consisten en <em>etiquetas</em> que acompañan a los datos.
    Estas etiquetas pueden indicar
</p>
<ul id="posibilidad-lista">
    <li>Diseño del elemento: cómo debe ser mostrado.</li>
    <li>Semántica: qué contiene ese dato. Cómo debe interpretarse.</li>
    <li>Identificación del lugar donde están los datos.</li>
</ul>
```

En el ejemplo, el texto se acompaña de etiquetas que determinan su diseño. La primera etiqueta <p> indica que ese texto forma un párrafo. Dentro hay palabras en negrita (strong) y en cursiva (em). Después hay una lista no ordenada (ul) compuesta por elementos (cada li).

Las etiquetas (por lo general) deben ser abiertas y cerradas para indicar cuándo acaban. Las etiquetas pueden contener atributos. Los **saltos de línea no son tenidos en cuenta** al leerse el documento, si todo estuviera en una sola línea funcionaría igual. Solo son añadidos para que resulten más **fáciles de leer**.

Estos lenguajes nacieron en IBM con el lenguaje GML (*Generalized Markup Language* / Lenguaje de marcas generalizado) en la década de los años 60. Este lenguaje se utilizaba para formatear texto. Fue evolucionando hasta el SGML, que era una versión estandarizada de GML lanzada en 1986.

A principios de la década de los 90, Tim Berners-Lee creó HTML, que era una aplicación práctica del SGML. HTML es el lenguaje usado por muchas de las páginas web de Internet y utiliza un lenguaje de marcado.



Nota

Todas las páginas web utilizan o bien HTML o XHTML (los cuales son parecidos pero no son iguales).

Más tarde, en 1998 se publicó la versión 1.0 de XML (*Extensible Markup Language* / Lenguaje de marcado extensible), que es otra interpretación de SGML pero con una sintaxis más parecida a HTML. Este formato se utiliza para el **intercambio de información** entre distintos programas informáticos. Está muy enfocado a Internet y servicios web. Sus principales propósitos son la simplicidad, la legibilidad y su soporte para caracteres Unicode (texto en lenguajes de todo el mundo).

XML se ha ido haciendo muy popular y se utiliza para cientos de escenarios diferentes. Por ejemplo, para XHTML, el cual es una evolución de HTML que utiliza las características de XML para producir documentos en HTML. También se utiliza para los documentos de *Open Office*, para protocolos de comunicación (chat) como XMPP, etc.



Actividades

-
1. Busque en Internet ejemplos de cómo es la sintaxis de SGML.

3. Características de XML

El XML es un lenguaje de marcado basado en documentos de texto con un determinado formato. Su principal característica es la legibilidad.



Sabía que...

Se pensó para compartir datos de forma estructurada, aunque ha ido evolucionando hasta permitir bases de datos basadas en XML, hojas de cálculo, documentos de texto, etc.

Un documento XML se compone de elementos y estos elementos pueden contener atributos, texto u otros elementos. Una característica de XML es que no se permiten errores de sintaxis en un documento de XML, debe ser 100% correcto para poder ser procesado, cosa que no suele ocurrir con HTML, por ejemplo.

3.1. El DOM

El DOM (*Document Object Model* / Modelo de objetos del documento) es una interfaz que permite comunicarse con documentos HTML y XML mediante **orientación a objetos**.

Los documentos XML son árboles anidados, con elementos que pueden contener a su vez otros elementos. Cada elemento puede tener una serie de atributos y puede albergar otros elementos también.

El DOM permite acceder y **manipular documentos** XML o HTML desde un lenguaje de programación. Es muy habitual utilizar la interfaz del DOM mediante Javascript para modificar los contenidos de una página HTML de forma dinámica. También es utilizado para leer documentos XML mediante un lenguaje que podría ser Java o PHP, por ejemplo.



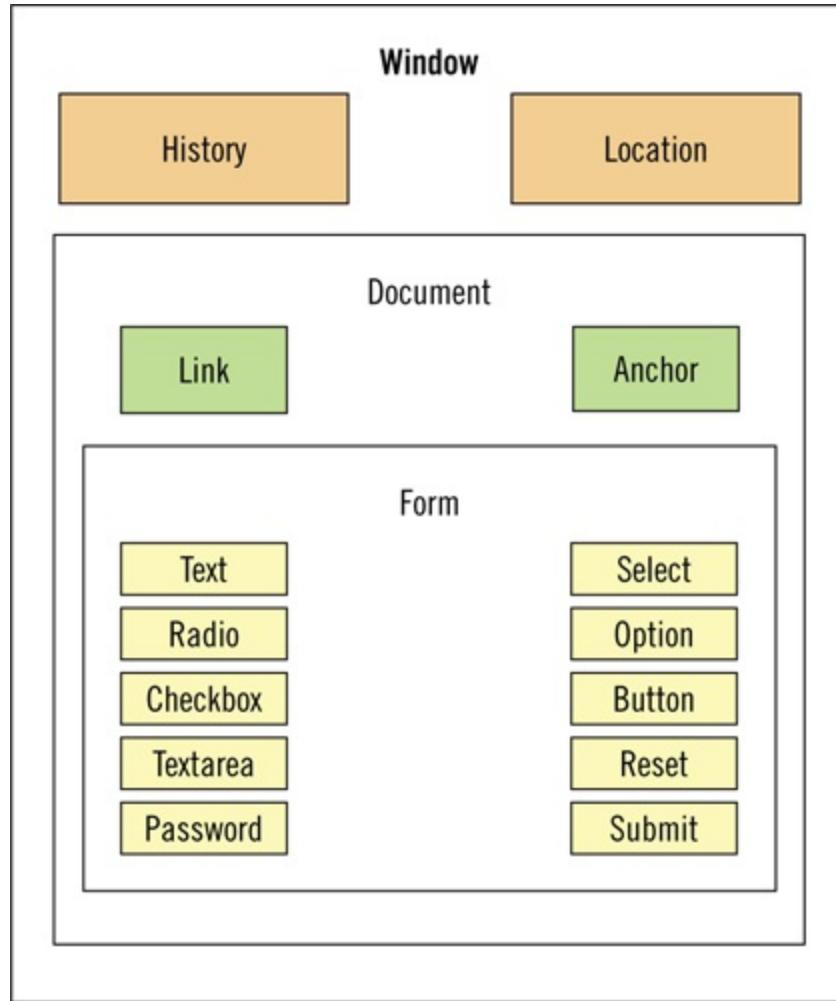
Nota

El DOM permite recorrer cada objeto del documento de forma recursiva, permite seleccionar todas las etiquetas de un mismo nombre o seleccionar un elemento según su atributo “id”.

En el apartado de “Programación de análisis XML mediante lenguajes en servidor” veremos cómo, utilizando el DOM y PHP, podremos leer un documento XML de forma dinámica.

El DOM se basa en dos elementos: “Window” y “Document”. “Window” se relaciona con el navegador del usuario, “Document” es el documento HTML/ XML que se está manipulando.

Aquí se ve un ejemplo de cómo se procesa el DOM. Se va anidando y agrupando conceptos. Todo va englobado dentro de “Document” y “Window”.



Actividades

2. Haga una lista de cinco aplicaciones o librerías que soporten el modelo del DOM.
-

3.2. Partes de un documento XML: marcas, elementos, atributos, etc.

Un documento XML se compone de etiquetas (marcas que delimitan cada elemento). Una etiqueta puede tener atributos o no. Toda etiqueta abierta debe ser cerrada posteriormente.

Aquí se muestra un ejemplo de documento XML para explicar cada parte:

```
<?xml version="1.0" encoding="utf-8" ?>
<!-- Comentario dentro de XML -->

<contactos>
    <contacto>
        <nombre>Jose María</nombre>
        <telefono tipo="movil">888-444-222</telefono>
    </contacto>

    <contacto>
        <nombre>Laura</nombre>
        <telefono tipo="fijo">888-111-000</telefono>
    </contacto>

    <contacto>
        <nombre>Pepe</nombre>
        <telefono tipo="movil">888-123-444</telefono>
    </contacto>

</contactos>
```

La primera línea especifica la versión de XML y la codificación de caracteres del documento. Si se introducen caracteres especiales y acentos, debe utilizarse utf-8. Esta línea es opcional, pero si se encuentra presente, debe estar en la primera línea del documento, el cual debe empezar directamente con esta declaración. Esta etiqueta es especial y es diferente a las definidas por el usuario y no es necesario cerrarla.



Nota

No es correcto que el documento empiece con espacios o saltos de línea. Debe empezar exactamente con los caracteres <?xml

A continuación, se ha introducido un comentario de XML para que se vea cómo se utilizan. Los comentarios pueden ser útiles para añadir pequeñas aclaraciones sobre el documento. Empiezan por “<!--“ seguido de un **espacio** y terminan en un espacio seguido de “-->”. Los espacios son importantes, porque de lo contrario no se está aplicando el comentario.

Todo documento XML **debe contener un elemento principal** que englobe los demás. En este caso, es el elemento <**contactos**>. Todas las etiquetas deben ser cerradas. Las etiquetas se abren mediante <**nombre_etiqueta**> y se cierran mediante

</nombre_etiqueta>. También es posible que una etiqueta se cierre a sí misma, en caso de que no tenga contenido, solo atributos. En ese caso, se utilizaría **<nombre_etiqueta />**.



Sabía que...

Al elemento principal también se le suele llamar “elemento raíz”.

Pueden contener atributos, que se especificarán en la etiqueta de apertura únicamente. Los atributos se colocan separados por espacios, siguiendo la estructura de nombre=”valor” (el valor va siempre entre comillas). Los atributos siempre son de tipo texto. Cuando se abre una etiqueta, su etiqueta de cierre debe tener exactamente el mismo tipo de letra mayúsculas/minúsculas que la etiqueta de apertura.

Los nombres de etiqueta deben empezar siempre con una letra. No pueden empezar con un número o carácter especial.

Dentro de este elemento principal pueden colocarse nuevos elementos. Es posible **anidar las etiquetas** todo lo que se quiera.

Para diferenciar, pueden distinguirse las partes de un documento XML de la siguiente manera:

- **Marcas:** `<contacto>` sería una marca (o etiqueta). Todo lo que vaya entre menor que y mayor que.
- **Elementos:** un elemento es lo que se encuentra entre un conjunto de marcas. Este es el elemento “contacto”.

```
<contacto>
  <nOMBRE>Jose María</nOMBRE>
  <TELEFONO tipo="móvil">888-444-222</TELEFONO>
</CONTACTO>
```

- **Atributos:** `tipo="móvil"` sería un atributo de la marca `<teléfono>`. **Todo atributo debe tener un valor y este debe ir entre comillas.**

Los saltos de línea no son tenidos en cuenta en el documento. Cuando se añaden espacios, **solo el primer espacio** es tenido en cuenta. Si se quisiera añadir más de un espacio explícitamente tendría que utilizarse el **carácter de espacio** (). El formato XML sí distingue entre minúsculas y mayúsculas, así que hay que asegurarse de que todas las etiquetas utilizan el mismo tipo de letra.

Dentro del contenido de una etiqueta **no es posible utilizar los caracteres** de menor que o mayor que, porque de lo contrario el intérprete podría pensar que se trata del cierre de una etiqueta. Para ello, habría que utilizar los caracteres especiales de < y > para < y > respectivamente (su nombre es una abreviatura de less than y greater than).

No obstante, es muy común que en un documento XML se añadan etiquetas que contengan texto en HTML o texto que no pueda convertirse, para ello se utilizan las secciones CDATA, que permiten utilizar cualquier carácter en una etiqueta. Las secciones CDATA se utilizan de la siguiente manera:

```
<?xml version="1.0" encoding="utf-8" ?>
<entradas>
  <entrada>
    <título>Entrada con HTML</título>
    <contenido><![CDATA[<p>Este contenido contiene <strong>etiquetas</strong></p>]]></contenido>
  </entrada>
</entradas>
```

Entre los segundos corchetes puede incluirse el texto que se quiera, sin restricciones.



Actividades

3. Cree un documento XML que almacene una lista de recetas de cocina. Cada receta tiene un nombre y una descripción de texto (la cual puede contener HTML dentro).
-



Aplicación práctica

Se le ha enviado esta tabla en formato Excel:

Horas	L	M	X	J	V
8:30 9:30	Sociales	Informática	Gimnasia	Sociales	Ética
9:30 10:30	Matemáticas B	Religión	Sociales	Tecnolog.	Gimnasia
10:30 11:30	Francés	Lengua	Lengua	Francés	Física y Q.
11:30	Recreo	Recreo	Recreo	Recreo	Recreo
12 a 13	Ética	Matemát. B	Francés	Tutoría	Matemát. B
13 a 13:50	Lengua	Inglés 2º Id	Física y Q.	Inglés 2º Id	Francés
14 a 14:45	Tecnología	Física y Q.	Tecnolog.	Informática	Lengua

Escriba un documento XML que permita compartir esa información con cualquier servicio web.

SOLUCIÓN

```
<horario>
  <dia>
    <nombre>Lunes</nombre>
    <horas>
      <hora valor="8:30 a 9:30">Sociales</hora>
      <hora valor="9:30 a 10:30">Matemáticas B</hora>
      <hora valor="10:30 a 11:30">Francés</hora>
      <hora valor="11:30 a 12:00">Recreo</hora>
      <hora valor="12:00 a 13:00">Ética</hora>
      <hora valor="13:00 a 13:50">Lengua</hora>
      <hora valor="14:00 a 14:45">Tecnología</hora>
    </horas>
  </dia>
  <dia>
    <nombre>Martes</nombre>
    <horas>
      <hora valor="8:30 a 9:30">Informática</hora>
      <hora valor="9:30 a 10:30">Religión</hora>
      <hora valor="10:30 a 11:30">Lengua</hora>
      <hora valor="11:30 a 12:00">Recreo</hora>
      <hora valor="12:00 a 13:00">Matemáticas B</hora>
      <hora valor="13:00 a 13:50">Inglés 2º Idioma</hora>
      <hora valor="14:00 a 14:45">Física y Química</hora>
    </horas>
  </dia>
  <dia>
    <nombre>Miércoles</nombre>
    <horas>
      <hora valor="8:30 a 9:30">Gimnasia</hora>
      <hora valor="9:30 a 10:30">Sociales</hora>
      <hora valor="10:30 a 11:30">Lengua</hora>
      <hora valor="11:30 a 12:00">Recreo</hora>
      <hora valor="12:00 a 13:00">Francés</hora>
      <hora valor="13:00 a 13:50">Física y Química</hora>
      <hora valor="14:00 a 14:45">Tecnología</hora>
    </horas>
  </dia>
```

```
<dia>
  <nOMBRE>Jueves</nOMBRE>
  <horas>
    <hora valor="8:30 a 9:30">Sociales</hora>
    <hora valor="9:30 a 10:30">Tecnología</hora>
    <hora valor="10:30 a 11:30">Francés</hora>
    <hora valor="11:30 a 12:00">Recreo</hora>
    <hora valor="12:00 a 13:00">Tutoría</hora>
    <hora valor="13:00 a 13:50">Inglés 2º Idioma</hora>
    <hora valor="14:00 a 14:45">Informática</hora>
  </horas>
</dia>
<dia>
  <nOMBRE>Viernes</nOMBRE>
  <horas>
    <hora valor="8:30 a 9:30">Ética</hora>
    <hora valor="9:30 a 10:30">Gimnasia</hora>
    <hora valor="10:30 a 11:30">Física y Química</hora>
    <hora valor="11:30 a 12:00">Recreo</hora>
    <hora valor="12:00 a 13:00">Matemáticas B</hora>
    <hora valor="13:00 a 13:50">Francés</hora>
    <hora valor="14:00 a 14:45">Lengua</hora>
  </horas>
</dia>
</horario>
```

3.3. Sintaxis y semántica de documentos XML: documentos válidos y bien formados

Como se ha mencionado anteriormente, todo documento XML debe ser válido y no contener errores de sintaxis, de lo contrario no podrá ser procesado. Aquí se irán viendo técnicas para asegurar que los documentos XML no contengan incorrecciones.

Para ello, debe comprobarse que no hay comillas sin cerrar, etiquetas sin cerrar, caracteres sin escapar, etiquetas no anidadas correctamente, etc.



Importante

XML proporciona una gran libertad al programador para definir las etiquetas que se quieran, lo cual abre la puerta a todo tipo de posibles errores.

Para validar documentos de XML pueden utilizarse servicios online como

<http://www.xmlvalidation.com/> o http://www.w3schools.com/xml/xml_validator.asp, ahí puede pegarse una cadena de texto en formato XML y dirá si contiene errores o no.

El W3C

El W3C (*World Wide Web Consortium*) es un consorcio que trabaja en el desarrollo de estándares para la *World Wide Web* (páginas y servicios web).

El estándar XML está regulado por el W3C, que es el organismo que propone cambios y documenta todas las características.

Cuando un estándar recibe la categoría de “**Recomendación W3C**” quiere decir que ya ha sido ampliamente probado tanto práctica como conceptualmente. El W3C lo propone como estándar y recomienda su uso.

El estándar XML está considerado “Recomendación W3C” y todos los documentos que cumplan la especificación XML de W3C deberían ser 100% válidos y legibles desde cualquier plataforma.

Casi todos los estándares tratados en este libro han recibido esta categoría por parte del W3C y su uso se encuentra ampliamente extendido.



Actividades

4. Busque la lista de estándares que han recibido la calificación de “Recomendación W3C”.
-

Semántica en XML

Existe un movimiento que quiere que la web sea semántica. La idea es que cada etiqueta **aporte un significado extra**, aparte de su valor; que los documentos sean legibles por una aplicación informática de tal forma que entienda cómo tratar cada dato en cada situación.

Esto se hace para buscar una web que sea **accesible para todo el mundo**, que sea posible procesar un documento en el navegador del ordenador sobremesa, desde el móvil de última generación o desde un lector para ciegos, por ejemplo.

Vea un ejemplo sencillo de dos etiquetas que pueden servir para lo mismo:

```
<span style="font-weight:bold;">Estoy en negrita</span>
<strong>Estoy en negrita</strong>
```

La primera etiqueta `` simplemente indica que el texto está en negrita. La segunda, ``, indica que esa frase tiene un **peso mayor**. El navegador hará que se vea en negrita como comportamiento estándar, pero además, sabrá que esa frase es importante.

Un lector para ciegos podrá leer esa palabra de forma diferente. Un buscador de Internet, cuando procese ese documento, sabrá que esas palabras tienen más importancia que las demás, etc.

La idea es que se introduzcan metadatos o convenciones de nombre estándar que aporten más información sobre cada elemento. En este caso, la convención es que **strong** significa “palabras importantes”.

Los espacios de nombres (*namespaces*)

Es posible que dos etiquetas tengan un mismo nombre pero tengan una **semántica** diferente. Para ello, existen los espacios de nombres, que permiten delimitar a qué se refiere cada conjunto de etiquetas.

Para incluir un espacio de nombres, se utiliza el atributo `xmlns` en la etiqueta que lo empleará. Todas las etiquetas que estén dentro de esa etiqueta padre heredarán ese espacio de nombres. Puede utilizarse el atributo `xmlns` junto a un prefijo si quiere añadir un prefijo delante del nombre de cada elemento.

```
<?xml version="1.0" encoding="UTF-8"?>
<estilos xmlns="http://www.misitio.com/espacio_de_nombres">
  <estilo>
    <nOMBRE>Fuerte</nOMBRE>
    <color negrita="true">Rojo</color>
  </estilo>
  <estilo>
    <nOMBRE>Claro</nOMBRE>
    <color>Verde</color>
  </estilo>
</estilos>

<?xml version="1.0" encoding="UTF-8"?>
<estilos xmlns:prefijo="http://www.misitio.com/espacio_de_nombres">
  <prefijo:estilo>
    <prefijo:nOMBRE>Fuerte</prefijo:nOMBRE>
    <prefijo:color negrita="true">Rojo</prefijo:color>
  </prefijo:estilo>
  <prefijo:estilo>
    <prefijo:nOMBRE>Claro</prefijo:nOMBRE>
    <prefijo:color>Verde</prefijo:color>
  </prefijo:estilo>
</estilos>
```



Nota

Esto es útil cuando se mezclan dos espacios de nombres que contienen etiquetas del mismo nombre.

En el primer ejemplo, simplemente se señala que se está utilizando ese espacio de nombres dentro de la etiqueta `<estilos>`. En el segundo, se utiliza el espacio de nombres pero se le añade un prefijo delante, de tal manera, que **al abrir** cada etiqueta, debe indicarse su nombre con el prefijo seguido de dos puntos primero, de lo contrario no se estará utilizando el espacio de nombres correctamente.

Lo habitual es utilizar espacios de nombres creados por el W3C o por otro tipo de estándar. Crear un espacio de nombres para uno mismo no es algo que se suela necesitar. Es más habitual utilizar documentos DTD o XSD.

Estos documentos se suelen utilizar para **aportar semántica**. Al enlazar a un espacio de nombres, cualquier aplicación que procese el documento puede saber **a qué se refiere cada etiqueta**.



Actividades

5. ¿Cuál es el namespace utilizado en documentos XML de formato KML (Google Earth)?

4. Estructura de XML

Los documentos XML deben estar bien estructurados, esta es la principal restricción. Todo documento XML debe contener **una** y solo una etiqueta padre **principal**. Dentro de esa etiqueta es donde se anidarán las demás etiquetas.

```
<?xml version="1.0" encoding="utf-8" ?>
<entradas>
  <entrada>
    <título>Entrada con HTML</título>
    <contenido><! [CDATA[<p>Este contenido contiene <strong>etiquetas</strong></p>]]>
  </entrada>
  </contenido>
</entradas>
```



Nota

La anidación de los elementos debe ser correcta. Toda etiqueta que se encuentre dentro de otra debe ser cerrada antes de cerrar su etiqueta padre.

Ahí se aprecia que **<entrada>** es cerrada antes de que se cierre **<contenido>**, lo cual es incorrecto.

Todas las etiquetas que se abran deben ser **cerradas dentro del nivel de anidación en que fueron abiertas**.

La tabulación y los saltos de línea son añadidos en los ejemplos para aumentar significativamente la legibilidad de los documentos XML, pero son prescindibles.

Es habitual encontrarse documentos XML en los que todo se encuentra en una línea para ahorrar espacio. Esto no supone ningún problema a la hora de procesar estos documentos. Su única desventaja es que pierden la legibilidad por parte de humanos.



Actividades

6. Arregle el siguiente documento XML de tal manera que no contenga errores:

```
<elemento requerido evento=sí>El signo > significa "mayor que"</Elemento>
<elemento>Este elemento es más sencillo.
```

4.1. Esquemas XML: DTD y XML Schema

Los documentos XML permiten **total libertad** al programador, que puede añadir tantas etiquetas como quiera con el nombre que haga falta. Esto presenta una gran ventaja porque se puede progresar muy rápidamente, pero por otra parte, no hay validaciones que determinen qué elementos deben encontrarse en un documento, lo cual es un problema cuando se trabaja en grandes organizaciones o cuando se trabaja con estándares de XML.

Las definiciones DTD y los XML Schemas se crearon para listar las **restricciones y estructura** de los datos con los que se trabaja. Puede elegirse qué etiquetas deben aparecer; qué nombre deben tener; su estructura de anidación; los atributos que deben tener y el tipo de dato que deben contener.

De esta manera, si junto a un documento XML se adjunta su definición DTD o su XML Schema, cualquier programador que vaya a hacer uso de ese documento, podrá hacerlo de una forma mucho más fiable, porque no solo tendrá que fiarse de los datos que se encuentren actualmente en el documento XML, sino que sabrá todas las restricciones de los mismos, por lo que podrá adaptarse a cualquier versión de ese documento sin problemas, sin importar los datos que contenga.

Las definiciones DTD y los XML Schema son dos sistemas diferentes que se utilizan con el mismo propósito: especificar las restricciones en la estructura y la sintaxis de

los documentos XML.



Nota

DTD tiene más limitaciones. XML Schema es una evolución que cubre más casos.

Normalmente, el documento XML enlazará a su definición de restricciones. En DTD es posible especificar todas las restricciones dentro del documento XML, pero lo habitual es tener las restricciones en un archivo aparte y enlazar a este desde XML.

DTD

Sirve para definir la estructura de un documento XML. Su principal fin es definir la estructura de los datos para poder utilizar una estructura común y tener una consistencia entre todos los documentos XML que utilicen la misma DTD.

Esto permite que todos los documentos XML que utilicen la misma DTD sigan un mismo proceso de validación.

La idea de un documento DTD es listar los nombres de etiqueta que se van a utilizar, junto a sus posibles atributos. Además, es posible especificar qué elementos pueden ir anidados dentro de ese elemento principal, para asegurar que la anidación es correcta.

Para enlazar a un documento DTD dentro de un documento XML, se utilizaría la etiqueta especial DOCTYPE:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE entradas SYSTEM "entradas.dtd">

<entradas>
  <entrada>
    <título>Entrada con HTML</título>
    <contenido><![CDATA[<p>Este contenido contiene <strong>etiquetas</strong></p>]]></contenido>
  </entrada>
</entradas>
```

Tras la definición de la versión de XML, se utiliza la etiqueta `<!DOCTYPE` seguida del nombre del elemento principal sobre el que se aplica la DTD (en este caso,

<entradas>). A continuación viene la palabra **SYSTEM** seguida del nombre del archivo en el que se encuentra la dtd, entre comillas.

La palabra **SYSTEM** se utiliza cuando el fichero se encuentra en el ordenador local en que se encuentra el documento XML. Es posible utilizar la palabra **PUBLIC** cuando el fichero se encuentre en Internet. Entonces, “entre paréntesis” se incluiría la URL donde se encuentra el archivo dtd.

El archivo entradas.dtd contendría algo así como:

```
<!ELEMENT entradas (entrada)>
<!ELEMENT entrada (titulo,fecha_publicacion?,contenido)>
<!ELEMENT titulo (#PCDATA)>
<!ELEMENT contenido (#PCDATA)
```

La sintaxis es un poco compleja, pero fácil de entender. Por cada línea y etiqueta **<!ELEMENT** se especifica cada etiqueta que contendrá el documento XML.

La etiqueta padre es **entradas**, entre paréntesis aparecen las etiquetas hijo que puede tener. Si puede tener más de una, irán separadas por comas. Si una etiqueta puede tener una etiqueta hija pero opcionalmente, su nombre irá seguido de un signo de interrogación.

Si una etiqueta no tiene más elementos hijos y contiene texto, se utilizará la palabra **#PCDATA** para indicar que esa etiqueta debe contener texto simple.

Además de utilizar archivos externos para la DTD, puede incrustarse la DTD dentro del XML de la siguiente manera:

```
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE entradas [
  <!ELEMENT entradas (entrada)>
  <!ELEMENT entrada (titulo,fecha_publicacion?,contenido)>
  <!ELEMENT titulo (#PCDATA)>
  <!ELEMENT contenido (#PCDATA)
]>

<entradas>
  <entrada>
    <titulo>Entrada con HTML</titulo>
    <contenido><![CDATA[<p>Este contenido contiene <strong>etiquetas</strong></p>]]></contenido>
  </entrada>
</entradas>
```



Nota

Esto puede ser útil cuando la DTD no es compartida entre varios documentos, por lo que no compensa crear un archivo externo.

Un documento DTD está compuesto por líneas que definen el comportamiento de cada etiqueta posible en un documento XML. Cada línea contiene la definición de una etiqueta en particular.

Cada línea empieza con **<!ELEMENT** seguido del nombre de la etiqueta y termina con un símbolo **>**. Esta etiqueta no debe cerrarse, porque un documento DTD no sigue la sintaxis de XML.

Tras el nombre de etiqueta, entre paréntesis se lista el nombre de los elementos que esa etiqueta puede contener dentro. Si no va a contener etiquetas dentro, o sea, solo va a **contener texto**, se utiliza #PCDATA. En las declaraciones de etiquetas DTD siempre se usa PCDATA en vez de CDATA.

Cuando una etiqueta debe estar siempre vacía, en vez de listar las propiedades entre paréntesis, se utiliza la palabra clave **EMPTY**.

Cuando una etiqueta puede tener etiquetas dentro, dependiendo de cómo se ponga su nombre, puede indicarse una cosa u otra:

- Nombre de la etiqueta sin más (por ejemplo, **<!ELEMENT etiqueta (hijo1)>**): Debe haber una etiqueta **<hijo1>** dentro de **<etiqueta>**. Una exactamente, no vale que tenga más de una.
- Nombre seguido de signo más (**<!ELEMENT etiqueta (hijo+)>**): La etiqueta **<etiqueta>** puede tener varias etiquetas **<hijo>** dentro, una mínimo.
- Nombre seguido de asterisco (**<!ELEMENT etiqueta (hijo*)>**): La etiqueta **<etiqueta>** puede tener desde cero hasta muchas etiquetas **<hijo>**.
- Nombre seguido de pregunta (**<!ELEMENT etiqueta (hijo1?)>**): Puede haber una etiqueta **<hijo1>** dentro de **<etiqueta>**; es opcional.
- Dos nombres separados por barra vertical (**<!ELEMENT etiqueta (hijo1|hijo2)>**): la etiqueta **<etiqueta>** debe tener dentro una etiqueta que se llame o **<hijo1>** o **<hijo2>**.

Definir la lista de atributos utilizando DTD

Además de especificar qué etiquetas deben aparecer y qué pueden contener, es posible especificar qué atributos puede tener cada etiqueta y **qué valores** deben contener.

Para ello, se utilizan las etiquetas **<!ATTLIST**. A continuación se muestran varios ejemplos listando los atributos que puede tener la etiqueta producto:

```
<!ATTLIST producto precio CDATA #REQUIRED>
<!ATTLIST producto moneda (euro|libra|dolar) "euro">
<!ATTLIST producto descuento #IMPLIED>
<!ATTLIST producto tienda #FIXED "mitienda">
```

Para cada atributo se utiliza una etiqueta. Primero se pone ATTLIST, seguido del nombre de la etiqueta que contendrá el atributo. Luego se indica el nombre del atributo seguido de su tipo.

Su tipo puede ser CDATA (texto), una enumeración de posibles valores (como el segundo ejemplo) o ID (un atributo de valor único en todo el documento); hay otros tipos pero son menos utilizados.

Tras el tipo de dato, puede señalarse si es **requerido o no** y darle un **valor por defecto**.

Tras el tipo de dato, si se utiliza #REQUIRED significa que el atributo debe estar presente y tener un valor obligatoriamente. #IMPLIED significa que el atributo es opcional. #FIXED significa que el elemento tiene un atributo implícito de valor ya prefijado en la DTD. Si el usuario añade este atributo al XML, el procesador de XML deberá devolver un error.

Tras las restricciones, se le puede dar el valor por defecto del atributo, entre comillas.



Nota

No es posible darle un valor por defecto a un atributo #REQUIRED.



Actividades

-
7. Busque cuál es el DTD del estándar XHTML 1.0 Strict y lea sus restricciones.
-

XML Schema

XML Schema se creó para poder definir restricciones de datos mucho más complejas. XML Schema aporta muchas más características, pero las características de DTD son más que suficientes para la mayoría, por eso sigue siendo más popular que XML Schema.

La especificación de los XML Schema ocupa más de cien páginas (<<http://www.w3.org/TR/xmlschema-0/>>) y es criticada por su complejidad. Aquí se tratarán sus características básicas para conocer qué aporta con respecto a DTD.

Para enlazar a una especificación de XML Schema desde un documento XML se utiliza lo siguiente:

```
<?xml version="1.0" encoding="utf-8" ?>
<entradas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="entradas.xsd">
    <entrada>
        <título>Entrada con HTML</título>
        <contenido><!CDATA[<p>Este contenido contiene <strong>etiquetas</strong></p>]]></contenido>
    </entrada>
</entradas>
```

Dentro de la etiqueta padre se incluirían dos atributos, uno llamado `xmlns:xsi` con la URL de la especificación de XML Schema que va a utilizarse y otro llamado `xsi:noNamespaceSchemaLocation` con la localización del archivo con las definiciones de XML. Estos archivos suelen tener la extensión “xsd” (por XML Schema Definition).

Dentro de este archivo los contenidos serían similares a estos:

```

<?xml version="1.0" encoding="utf-8"?>
<xss:schema elementFormDefault="qualified" xmlns:xss="http://www.w3.org/2001/XMLSchema">
  <xss:element name="entrada">
    <xss:complexType>
      <xss:sequence>
        <xss:element name="titulo" type="xss:string" />
        <xss:element name="fecha_publicacion" type="xss:date" minOccurs="0" />
        <xss:element name="contenido" type="xss:string" />
      </xss:sequence>
    </xss:complexType>
  </xss:element>
</xss:schema>

```

La sintaxis es XML válido, a diferencia de los documentos DTD. Aquí hay que copiar el formato del ejemplo y tener en cuenta los atributos “name”, que determinan el nombre de cada etiqueta soportada. Cada etiqueta puede contener otras etiquetas dentro, que se añadirían utilizando xs:complexType y xs:sequence.

Cada etiqueta puede contener un tipo de dato. Si una etiqueta es opcional puede utilizarse el atributo minOccurs para ponerlo a cero (por defecto es 1).

4.2. Hojas de estilo XML: el estándar XSLT y XSL

XSLT es un sistema mediante el cual, mezclando un documento XML y un documento XSL, se genera otro documento en un **formato distinto**. Lo más habitual es utilizarlo para generar documentos XHTML, aunque sería posible generar otro tipo de documentos (PDF, texto, LaTeX, etc.).

Esto es útil para la creación de plantillas. Por ejemplo, un documento XML contiene los datos de una página web y un documento XSL contiene la estructura del XHTML que se va a generar. Mediante un proceso, podrían introducirse los datos del XML dentro de la plantilla y generar la página web.

El XML sería como este:

```

<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xsl" href="entradas-plantilla.xsl"?>
<entradas>
  <entrada>
    <titulo>Entrada más reciente</titulo>
    <contenido><! [CDATA[<p>Este contenido contiene <strong>etiquetas</strong></p>]]></contenido>
  </entrada>
  <entrada>
    <titulo>Primera entrada</titulo>
    <contenido><! [CDATA[<p>Esta es la primera entrada</p>]]></contenido>
  </entrada>
</entradas>

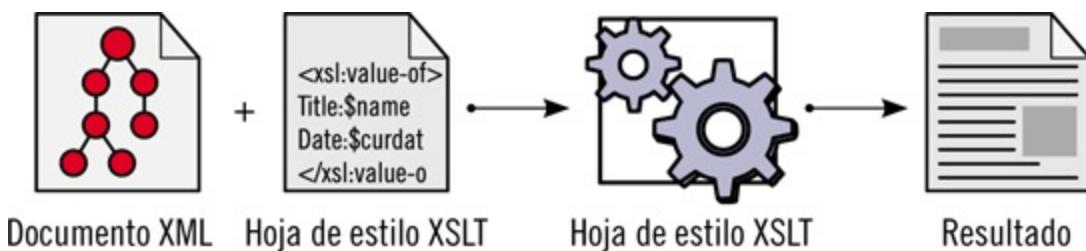
```

Se han incluido dos entradas para que se entienda mejor. Para enlazar a la hoja de estilos se utiliza `<?xml-stylesheet`. En href debe especificarse la ruta para acceder al archivo.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

<xsl:template match="/">
<html>
<head>
<title>Lista de entradas</title>
</head>
<body>
<h1>Lista de entradas</h1>
<ul>
<xsl:for-each select="entradas/entrada">
<li>
<h3><xsl:value-of select="titulo"/></h3>
<xsl:value-of select="contenido" disable-output-escaping="yes" />
</li>
</xsl:for-each>
</ul>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

El documento XSLT sigue esta estructura. Dentro de `xsl:template` se incluyen los contenidos de la plantilla. Pueden incluirse bucles mediante `xsl:for-each`. Los valores que se quieran reemplazar por los valores del documento XML, se incluyen mediante `xsl:value-of`. El atributo `select` contiene el nombre de la etiqueta que tiene el valor. El atributo `disable-output-escaping` se utiliza cuando el valor puede contener HTML y se desea que se muestre tal cual.



La manera de acceder a las etiquetas del XML es la que facilita XPath. Más adelante se verá cómo utilizar XPath.

Por un lado se encuentran los documentos XML, por otro, la hoja de estilos XSL. **XSLT es el proceso que transforma** ambos documentos en otro documento de un tipo diferente.

Este proceso lo hace una aplicación informática o una librería de software siguiendo el estándar de XSL.



Ejemplo

Cualquier navegador web moderno es capaz de mostrar un documento XML que enlace a una XSL.

Por lo general, no hay que preocuparse de cómo se hace la conversión del XSL. Lo habitual es contar con un programa o una librería que lo haga. Es necesario ocuparse únicamente de que el documento XML esté bien formado y que el documento XSL sea válido.



Actividades

8. Investigue algo sobre el procesador XLST de PHP.
-

4.3. Enlaces: XLL

El XLL (*XML Linking Language* o lenguaje de enlazado XML) es un lenguaje de marcado que sirve para crear enlaces internos y externos en documentos XML. Estos enlaces pueden ser otros documentos, imágenes o URL de Internet.

En HTML, la manera de enlazar es simplemente mediante la etiqueta `<a>`. En XML esto no funciona así, cualquier etiqueta puede funcionar como enlace. Para determinar que una etiqueta es un enlace, se utilizan los atributos `xlink:type` y `xlink:href`, para determinar el tipo de enlace y el lugar al que apuntan.

Para utilizar los XLinks debe declararse su espacio de nombres en la etiqueta principal del documento XML. He aquí un ejemplo:

```
<?xml version="1.0" encoding="utf-8" ?>
<enlaces xmlns:xlink="http://www.w3.org/1999/xlink">
  <enlace xlink:type="simple" xlink:href="http://www.google.es">Portada de Google</enlace>
  <enlace xlink:type="simple" xlink:href="http://es.wikipedia.org">wikipedia en Español</enlace>
</enlaces>
```

Para declarar el espacio de nombres simplemente debe copiarse el atributo xmlns:xlink tal cual se muestra en el ejemplo.



Sabía que...

En este ejemplo hemos creado un enlace simple. Es posible crear enlaces más complejos pero no lo veremos en este libro puesto que es un tema muy amplio y apenas tiene uso. Para una explicación más detallada habría que irse a la especificación oficial del estándar <<http://www.w3.org/TR/xlink/>>.

4.4. Agentes de usuario: XUA

XUA (*XML User Agents* / Agentes de usuario XML) es una iniciativa que se encuentra aún en construcción para que cada navegador web indique de forma explícita el conjunto de instrucciones de XML que soporta.

No hay nada que deba hacerse al respecto de XUA, es simplemente un estándar en el que se está trabajando para asegurarse de que todas las plataformas utilicen XML de forma correcta.

5. Estándares basados en XML

En los apartados anteriores se han visto complementos para documentos XML, para definir su estructura y sintaxis, para aplicarle estilos, para que enlacen a otros documentos, etc.

En este apartado se estudiarán estándares basados en XML. Son documentos con **sintaxis de XML que tienen una estructura determinada**. Normalmente, los

estándares de XML tienen, bien su propio documento DTD bien su espacio de nombres que debe enlazarse a nuestro XML para declarar que se está utilizando.

Al ser XML tan flexible, hay muchos estándares utilizados para distintos propósitos, como pueden ser la creación de páginas web (XHTML); firmas electrónicas; fuentes de noticias (RSS); hojas de cálculo (ODS); imágenes vectorizadas (SVG), etc.



Actividades

9. Busque ejemplos para conocer cómo es el estándar SVG.
-

5.1. Presentación de página: XHTML

XHTML es una derivación de HTML que utiliza XML para su sintaxis. En vez de usar HTML directamente, se emplea XHTML, que es HTML con una sintaxis de XML válido.

Sus principales características son las siguientes:

- Se pueden utilizar espacios de nombres de XML.
- En un documento HTML se permiten muchas incorrecciones sintácticas. Un documento XML debe ser 100% válido para ser procesado, por lo tanto, es más fácil de leer.
- XML es un formato muy extendido. Puede hacerse uso de herramientas pensadas específicamente para XML.

Las diferencias con el HTML convencional son las siguientes:

- El documento debe empezar con la declaración de la versión de xml (<?xml version="1.0" encoding="utf-8" ?>).
- Debe utilizarse una declaración DTD que determine la versión.
- Toda etiqueta abierta debe ser cerrada. En HTML era correcto emplear
 para un salto de línea. En XHTML hay que utilizar
 o
</br> para especificar cuándo acaba la etiqueta.
- Los valores de los atributos de las etiquetas deben ir siempre entre comillas.
- Los nombres de las etiquetas y los atributos deben ir siempre en minúsculas para unificar criterios.
- Todo atributo debe tener un valor siempre. No es posible hacer algo como

<input type="text" readonly />

- Cuando se añaden recursos como hojas de estilo o código Javascript incrustado en el HTML, ha de utilizarse la etiqueta **CDATA**.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es">
  <head>
    <title>XHTML de ejemplo</title>
    <script type="application/javascript">
      <![CDATA[
        function init() {
          alert("Página cargada");
        }
      ]]>
    </script>
  </head>
  <body onload="init()">
    <p>Ejemplo de página <br />Compatible con XHTML</p>
  </body>
</html>
```

Ahí se aprecia un ejemplo de XHTML. Como se ve, la sintaxis es XML 100% válido. Se utiliza la primera línea para especificar la versión de XML y la codificación de caracteres y el DOCTYPE, para especificar el DTD.



Nota

Cualquier programador que conozca XML podría aprender XHTML fácilmente. Y cualquier programador que conozca HTML tendría que simplemente aprender las restricciones de XML para empezar a utilizarlo.



Actividades

10. Busque información sobre el validador de XHTML creado por el W3C.

5.2. Selección de elementos XML: Xpath y XQuery

XML tiene dos estándares diseñados para localizar elementos concretos dentro de un documento XML. Son útiles para leer estos documentos de forma sencilla. Los estándares son el XPath y el XQuery. XPath es la sintaxis básica para referirse a elementos de un documento XML. XQuery es un lenguaje de programación más complejo construido sobre la base de XPath que permite realizar tareas más avanzadas, como mezclar varios documentos XML en uno o realizar filtraciones complejas.

XPath

XPath (abreviación de *XML Path Language*) sirve para **localizar información dentro de un documento** XML. Selecciona y hace referencia a los elementos de un documento XML.

Para todos los ejemplos y explicaciones de XPath se utilizará el siguiente documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<productos>
    <producto activo="si">
        <titulo>Limpiador de madera</titulo>
        <precio moneda="eur">2.99</precio>
    </producto>
    <producto activo="si">
        <titulo>Friegasuelos</titulo>
        <precio moneda="eur">1.50</precio>
    </producto>
    <producto>
        <titulo>Bayeta de microfibra</titulo>
        <precio moneda="gbp">1</precio>
    </producto>
</productos>
```

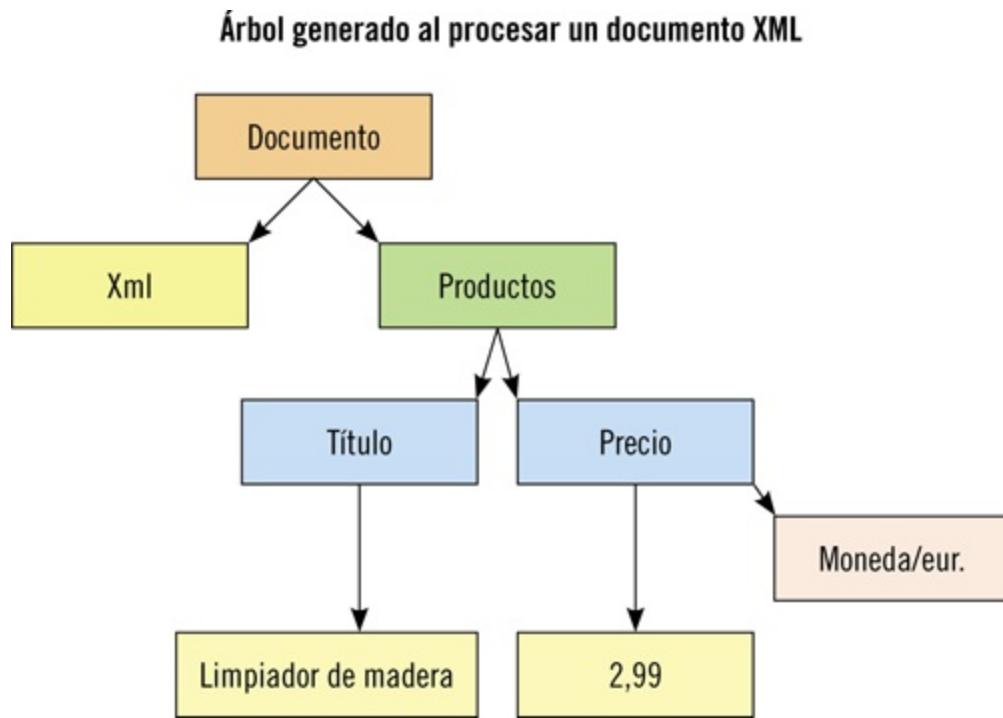


Nota

El documento de especificación completo se encuentra en <http://www.w3.org/TR/xpath20/>.

Al ser leídos, los documentos XML son procesados por un parser (analizador) de XML, que construye todo el **árbol de nodos** (siguiendo un modelo como el del DOM).

Este árbol comienza en el elemento raíz, el cual va recorriendo cada elemento que hay dentro hasta que llega a los nodos vacíos o a los que contienen solo texto.



El parser va procesando los elementos y detectando nodos. Los cuatro tipos de nodo más habituales son estos:

- **El nodo raíz:** se corresponde al documento XML en general. Solo hay un nodo raíz. No debe confundirse con el elemento raíz. El elemento raíz en nuestro XML de ejemplo sería “productos”. El nodo raíz está por encima de él.
- **Nodos elemento:** cualquier elemento del documento es un nodo elemento. Cualquier etiqueta XML se corresponde a un nodo elemento.
- **Nodos de texto:** cuando un nodo no tiene hijos pero tiene texto, tiene un hijo que es un nodo de texto. Un nodo de texto no tiene hijos, solamente está compuesto por los caracteres del texto.
- **Nodos atributo:** un elemento puede tener una serie de atributos. Estos atributos son procesados como nodos. Los nodos atributo no son hijos del elemento que los contiene, sino que están a su mismo nivel, son accesorios al mismo. Un nodo atributo tiene un nombre, un valor y un espacio de nombres aplicable (si se da el caso).

XPath es un **lenguaje de expresiones**, no un lenguaje estándar. La comunicación con él se efectúa mediante instrucciones simples, como si de una consola de comandos se tratase.



Ejemplo

Lenguajes estándar serían PHP, C, Java, etc.

La principal función de XPath son las “**location paths**” (caminos de localización). Consisten en expresiones con la “ruta” para acceder a una serie de nodos. Estas rutas son parecidas a las de un sistema UNIX. Por ejemplo, esta sería una expresión XPath:

```
/productos/producto
```

La sintaxis y el concepto son parecidos, pero aquí se está trabajando con árboles que pueden contener elementos del mismo nombre; en un sistema de ficheros no es posible tener dos carpetas o archivos del mismo nombre en el mismo sitio.

Aquí lo que se estaría haciendo es colocarse en el nodo raíz —que solo puede tener un elemento—; a continuación, se buscan **todos** los elementos de tipo “productos” (en este caso solo puede haber uno, pero podrían ser varios si el documento estuviera mal formado) y después, se buscan todos los elementos “producto” que se encuentren dentro de elementos “productos”. Por cada elemento “productos” se buscan los elementos “producto”.

Para utilizar XPath se precisa un **testeador de XPath**, que es una herramienta que recibe como parámetros un documento XML, una instrucción XPath y devuelve los resultados que se corresponden con la *location path*. La herramienta que se utilizará es <<http://www.freeformatter.com/xpath-tester.html>>, la cual se encuentra online, por lo que no hay que descargar ningún programa.

En XML Input se pega el documento XML que va a utilizarse o se selecciona la URL del documento (si este se encuentra publicado en Internet).



Importante

En XPath expression es donde se introduce la expresión XPath que se quiere evaluar. En XPath Result aparecen los resultados.

Predicados

Los predicados en XPath son **filtraciones**, como el WHERE de SQL. Se seleccionarán todos los productos que se encuentran activos:

```
/productos/producto[@activo="si"]
```

```
XPath result:  
Element='<producto activo="si">  
    <titulo>Limpiador de madera</titulo>  
    <precio moneda="eur">2.99</precio>  
</producto>'  
Element='<producto activo="si">  
    <titulo>Friegasuelos</titulo>  
    <precio moneda="eur">1.50</precio>  
</producto>'
```

Los predicados son la ruta de los elementos (*location path*), junto a posibles filtraciones por atributo. Las filtraciones por atributo se especifican entre corchetes. El nombre del atributo va precedido del prefijo `@`. El valor debe ir entre comillas porque los atributos de XML siempre son de tipo texto. Se pueden hacer múltiples filtraciones de atributo para un mismo elemento, poniendo unos corchetes detrás de otros.

Es posible indicar instrucciones como `/productos/producto[@activo]` que simplemente filtran los productos que tengan atributo “activo” (dando igual su valor).

Además del signo igual, puede utilizarse mayor que, menor que y distinto que.

También es posible filtrar por el ID de un elemento. Los ID deben estar declarados en la DTD, de lo contrario no funcionan. La idea de los ID es que un elemento puede tener un identificador único a lo largo de todo el documento.

```
id("producto_299")/titulo
```



Nota

Para filtrar por el elemento que tenga un ID específico, se utiliza la función id().

Suponiendo que nuestro documento tuviera un atributo ID por cada producto, esto seleccionaría sus títulos.

Location paths

Se han visto rutas sencillas, en las que se selecciona toda la ruta para llegar a esos elementos, pero a veces puede interesar saltar pasos o seleccionar elementos sin saber su nombre exacto, sino su lugar en el árbol:

- **Selectores descendentes** (selecciona todos los elementos que desciendan de un elemento recursivamente):

```
//producto
```

Se quieren todos los productos, no importa qué elementos padre tengan encima.

```
/productos//titulo
```

Se quieren todos los títulos de productos.

- **Selectores padre.** Son difíciles de entender porque dan un paso atrás al seleccionar. Requieren mucha práctica. Sirven para hacer filtraciones sobre elementos hijo pero cuando solo se quiere seleccionar el elemento padre.

```
//producto/precio[@moneda="eur"]/..
```

Todos los productos cuyo precio esté en euros. Primero parece que se selecciona solo el elemento “precio”, pero con los dos puntos se vuelve un paso atrás, de tal manera que solo se seleccionan los elementos “producto”.

```
//producto[@activo="si"]/precio[@moneda="eur"]/../titulo/text()
```

Otro ejemplo más complejo, que devuelve el título de los productos que estén activos y cuyo precio esté en euros. Primero se filtra precio, luego se sube un nivel en el árbol y se vuelve al elemento título para obtener su texto.

Selectores

Los selectores son el último extremo de la *location path*. Son el elemento final, lo que se quiere ver; son como el SELECT de SQL. Pueden seleccionarse nodos elemento, nodos atributo o nodos de texto.

Estos son varios ejemplos:

- Seleccionar todos los productos (devuelve un árbol de nodos):

```
/productos/producto
```

También es posible utilizar un asterisco, si se quieren seleccionar todas las etiquetas que haya sin importar su nombre:

```
/productos/*
```

- Selecciona todo lo que cuelgue de <productos>, sin importar cómo se llame.
- Seleccionar todos los atributos “moneda” de productos activos:

```
/productos/producto[@activo=”si”]/precio/@moneda
```

Seleccionar todos los nombres de producto:

```
/productos/producto[@activo=”si”]/titulo/text()
```

Cuando se seleccionan atributos, su nombre tiene el prefijo arroba. Cuando se quiere ver el texto, se utiliza la función text().

Cuando se trabaja con nodos elemento, además, es posible utilizar algunas funciones especiales para ser más específicos con **qué es lo que se selecciona**:

- Posición del elemento:

```
//producto[position()=2]
```

```
//producto[2]
```

o bien:

Devuelve el segundo producto. El índice empieza en el número 1. El nodo cero no existe.

- Último elemento:

```
//producto[last()]
```

Devuelve el último producto.

Es posible realizar algunas operaciones matemáticas sencillas con estas funciones:

```
//producto[not(position()=last())]
```

Productos que no sean el último.

```
//producto[(position() mod 2) != 0]
```

Devuelve un producto sí un producto no. O lo que es lo mismo, productos ubicados en posiciones impares.



Aplicación práctica

Imagine que está trabajando en una aplicación que accede a los contenidos de un documento XML utilizando una aplicación escrita en C#. Cuenta con el siguiente documento:

```

<resultados>
  <pelicula puntuacion="9">
    <titulo>Cadena perpetua</titulo>
    <año>1994</año>
  </pelicula>
  <pelicula puntuacion="9">
    <titulo traducción_literal="si">El padrino</titulo>
    <año>1972</año>
  </pelicula>
  <pelicula puntuacion="8">
    <titulo traducción_literal="si">Pulp Fiction</titulo>
    <año>1994</año>
  </pelicula>
  <pelicula puntuacion="8">
    <titulo>El bueno, el feo y el malo</titulo>
    <año>1966</año>
  </pelicula>
  <pelicula puntuacion="8">
    <titulo traducción_literal="si">El caballero oscuro</titulo>
    <año>2008</año>
  </pelicula>
</resultados>

```

Se necesitan las expresiones XPath para poder acceder a los siguientes nodos:

1. **Expresión para obtener las dos primeras películas cuya puntuación sea ocho.**
2. **Expresión para obtener el año de publicación de todas las películas.**
3. **Expresión para obtener todos los elementos “pelicula” cuyo título sea una traducción literal.**

SOLUCIÓN

1. `//pelicula[@puntuacion='8'][position() <= 2]`
2. `//año/text()`
3. `//pelicula/titulo[@traducción_literal='si']/..`

XQuery

XQuery es a XML lo que SQL es a un SGBD. Permite seleccionar qué datos interesan de una serie de documentos XML y qué condiciones deben cumplir para ser incluidos. Está construido sobre la base de XPath. La manera de referirse a cada elemento del árbol de nodos es mediante expresiones XPath.

Con XQuery pueden combinarse varios documentos XML para hacer lo que en SQL sería un JOIN.



Nota

Pueden ordenarse los resultados y aplicar filtraciones sobre los mismos (para devolver solo los que interesen).

Normalmente, cuando hay que procesar uno o varios documentos XML, lo que se hará será leerlos utilizando algún lenguaje de programación, utilizando librerías que implementen el DOM o que lean el XML de otra manera. Los documentos XML se leerán y todas las filtraciones se harán a mano, para ir descartando y manipulando los datos. Esto resulta útil cuando se quieren hacer pequeñas filtraciones u operaciones sencillas, porque se tardará menos tiempo en hacerlas y no supondrán mayor problema.

XPath sirve para realizar estas modificaciones sobre **grandes volúmenes de datos** y cuando las filtraciones son complejas y requieren de mucho trabajo de programación. Al leer los documentos XML se suelen encontrar dos grandes problemas: hacer **uniones entre varios documentos** y el **espacio en memoria**, porque normalmente habrá que volcar los documentos completos para poder trabajar con ellos. XQuery proporciona una interfaz más sencilla para comunicarse con documentos XML de forma unificada y más eficiente.

Para utilizar XQuery se necesitará una herramienta que la soporte. Esta es una característica muy avanzada de XML y poco extendida. Hay poco software que lo soporte de forma sencilla y la mayoría es software de pago. Kernow es una opción gratuita (y libre) que puede emplearse. En la pestaña de **XQuery Sandbox** se podrán probar las consultas que se realicen.



Nota

Kernow se puede descargar desde <http://kernowforsaxon.sourceforge.net/>. Tras 100 usos pide que se haga una donación, pero es opcional. Es simplemente para animar a los usuarios a donar.

XQuery utiliza un conjunto de expresiones llamadas **FLOWR** (se pronuncia como *flower*):

- FOR: itera sobre los elementos de una expresión escrita en XPath.
- LET: declara variables.
- WHERE: filtra los resultados.
- ORDER BY: ordena los resultados.
- RETURN: genera los resultados, después de haberse filtrado y ordenado.

La sintaxis XQuery consiste en una mezcla de expresiones FLOWR, código XML y expresiones XPath. Dominar este lenguaje requiere mucho conocimiento, simplemente aquí se ha mencionado en qué consiste y para qué sirve.

5.3. Firma electrónica: XML-Signature y Xades

XML Signature (Firma XML) define una sintaxis XML utilizada para firmas electrónicas. Se utiliza para firmar un documento de tal forma que se pueda garantizar su autenticidad.

Para firmar, se utiliza la etiqueta <**Signature**>, la cual albergará toda la información de la firma.

Hay tres maneras de firmar:

- Firma envoltorio (*enveloped*): el código de firma se encuentra dentro del documento XML que se quiere firmar.
- Firma envolvente (*enveloping*): la información XML que se quiere firmar está dentro del código de la firma.
- Firma independiente (*detached*): el código de firma se refiere a un documento XML distinto al actual.

Para firmar un documento es necesario contar con la clave privada de firma.

Aquí hay un ejemplo de documento con firma envoltorio:

```

<?xml version="1.0"?>
<pedidos>
  <pedido>
    <id>939999</id>
    <total>29.88</total>
  </pedido>

  <signature xmlns="http://www.w3.org/2000/09/xmldsig#">
    <SignedInfo>
      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
      <Reference URI="">
        <Transforms>
          <Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
        </Transforms>
        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
        <DigestValue>V0i1Den0qBzcslw7EkJfhw013/I=</DigestValue>
      </Reference>
    </SignedInfo>
    <SignatureValue>zxs85xswIVfbmjF3I...</SignatureValue>
    <KeyInfo>
      <KeyValue>
        <RSAKeyValue>
          <Modulus>
            05sRwFq8qy6gHmSMc3JuvtXTgpByqqUpK178G06EqXvu+8P6b1gI9KrNSKI3LFUAD
            H0Aiuh49h1/KCPybe2u+Dqwg2syyVG9gvVGONZXSGJuwBREjQ7Ba683WIubkQuQ2
            r10jgx+7SBuyTqjzm05gmw1asyrXC9kDMCLPx7K3k3s=
          </Modulus>
          <Exponent>
            AQAB
          </Exponent>
        </RSAKeyValue>
      </KeyValue>
      <KeyInfo>
    </Signature>
  </pedidos>

```

El documento principal es `<pedidos>`, pero en `<Signature>` se introduce la información sobre la firma.

No se entrará en más detalles, solamente se ilustra el concepto con un ejemplo, puesto que el tema de la firma con XML es muy amplio. La especificación completa del W3C tiene una longitud cercana a las 100 páginas para explicarla en profundidad.



Nota

La especificación completa del W3C se encuentra en <http://www.w3.org/TR/xmldsig-core/>.

XADES

XADES son las siglas de XML *Advanced Electronic Signatures* (firma electrónica avanzada XML). No es más que un conjunto de extensiones al estándar XML Encryption confeccionado para adecuarse a un uso como firma electrónica.

Define una serie de campos extra que lo hace más apto para ser utilizado como firma electrónica.

5.4. Cifrado: XML-Encryption

Cuando se envía **información sensible** —como números de tarjeta de crédito o datos personales confidenciales—, es común recurrir a técnicas de cifrado de datos, de tal manera que sea muy difícil o directamente imposible leer esos datos por parte de alguien inesperado.

El cifrado siempre suele ir de la mano de los conceptos **clave privada y clave pública**. Una clave privada consiste en unos datos que el emisor y receptor nunca publicarán y siempre mantendrán en secreto. La clave pública consiste en unos datos que se pueden compartir. Es al **mezclar la clave privada con la pública** cuando se pueden descifrar los datos que se cifraron.

El método más habitual para utilizar en XML consiste en el cifrado mediante la clave pública. El receptor de la información difunde su clave pública. El emisor lee esa clave pública, cifra su mensaje utilizando esa clave pública y la envía al receptor. El receptor podrá descifrar el mensaje utilizando su clave privada.

Solo el receptor que tenga la clave pública utilizada al enviar el mensaje podrá leer el mensaje.

Los mensajes se cifran utilizando una utilidad de cifrado que debe estar instalada, tanto en el emisor como en el receptor.

Para utilizarla, por un lado estará el archivo XML original (sin cifrar y por otro lado, una **plantilla XML de cifrado**, la cual tendrá una forma similar a la siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
XML Security Library example: Original XML
doc file before encryption (encrypt3 example).
-->
<EncryptedData
    xmlns="http://www.w3.org/2001/04/xmlenc#"
    Type="http://www.w3.org/2001/04/xmlenc#Element">
    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc"/>
    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
        <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
            <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
            <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
                <KeyName/>
            </KeyInfo>
            <CipherData>
                <CipherValue/>
            </CipherData>
        </EncryptedKey>
    </KeyInfo>
    <CipherData>
        <CipherValue/>
    </CipherData>
</EncryptedData>
```

Después, se utilizará una utilidad como por ejemplo el comando **xmlsec** (XML Security Library, el cual permite cifrar y descifrar documentos XML.

5.5. Otros estándares de uso común

XML se utiliza en muchos estándares relacionados con el **desarrolloweb**. Otros estándares de uso habitual en programación web son las fuentes de noticias y los mapas de sitio, los cuales se estudiarán a continuación.

RSS (Fuentes de noticias)

Los RSS (*Really Simple Sindication* / Suscripción realmente sencilla) son **fuentes de datos** que genera un sitio web (un blog o un periódico, por ejemplo con la lista de últimas publicaciones del sitio).

Cualquier usuario que conozca la URL de ese RSS podrá añadirla a su **lector de feeds** y recibir las últimas actualizaciones ahí. Es útil porque un mismo usuario puede seguir decenas de blogs y páginas distintas y leer todas las últimas actualizaciones desde un mismo sitio. También puede recibir alertas cuando un sitio haya publicado algo nuevo, de tal manera que reciba las actualizaciones al momento.

Para que un sitio web pueda ofrecer un RSS, tiene que tener un archivo XML público

que esté enlazado desde su HTML, diciendo que esa es la fuente de noticias disponible para el sitio.

Los RSS deben tener una estructura como la siguiente:

```
<?xml version="1.0" encoding="UTF-8" ?>
<rss version="2.0">
<channel>
    <title>Noticias de mi sitio</title>
    <description>Las últimas noticias de mi página web</description>
    <link>http://www.mipagina.com</link>

    <item>
        <title>Última entrada</title>
        <description>El contenido va aquí.</description>
        <link>http://www.mipagina.com/entrada3.html</link>
        <pubDate>Mon, 26 Nov 2013 17:15:00 +0000 </pubDate>
    </item>

    <item>
        <title>Penúltima entrada</title>
        <description>El contenido va aquí.</description>
        <link>http://www.mipagina.com/entrada2.html</link>
        <pubDate>Mon, 20 Nov 2013 12:03:00 +0000 </pubDate>
    </item>

</channel>
</rss>
```

Primero, en **<channel>** se indica el nombre, descripción y URL de la página web.

Seguidamente, en cada **<item>** se indican las últimas entradas. Las más recientes deben ir más arriba. Por cada una, se incluye un título, una descripción, la URL específica para ese elemento y la fecha de publicación. Las fechas se ponen siempre en el formato RFC 2822.



Sabía que...

Al formatear una fecha utilizando PHP, la palabra de formato a utilizar es “r” (<http://php.net/manual/es/function.date.php>).

El lector de *feeds* leerá el XML y detectará las entradas que no han sido leídas.

Cada elemento puede contener más propiedades, pero por ahora bastará con las imprescindibles.

En la página web debe añadirse una etiqueta <link> para enlazar la fuente de *feeds* con la página:

```
<link href="https://www.mipaginaJ.com/rss.xml" rel="alternate" type="application/rss+xml" title="Sortea2.com RSS Feed"/>
```



Actividades

11. Busque y lea el RSS del periódico El País.
-

XML Sitemaps (Mapas de sitio)

Cuando se tiene una página web pública, normalmente interesa que los **buscadores** la **indexen** para que así aparezca en los resultados de búsqueda y se obtengan más visitas. Los buscadores tienen sistemas muy potentes para tratar de descubrir las URL de un sitio, pero no siempre es fácil y es posible que se le escapen algunas.

Para eso existen los XML Sitemaps, para crear un **archivo** en el que se **listen todas las URL** de nuestro sitio y que los buscadores puedan leerlo.

El formato es el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>http://www.misitio.com/pagina1.html</loc>
    <lastmod>2013-11-25</lastmod>
    <changefreq>weekly</changefreq>
    <priority>0.6</priority>
  </url>
  <url>
    <loc>http://www.misitio.com/pagina2.html</loc>
    <lastmod>2013-10-28</lastmod>
    <changefreq>monthly</changefreq>
    <priority>0.7</priority>
  </url>
</urlset>
```

El elemento principal es el **<urlset>**, el cual debe utilizar el espacio de nombres que se muestra en el ejemplo.

Dentro, en cada **<url>** se lista una URL de nuestro sitio. Solo el elemento **<loc>** es obligatorio. En **<lastmod>** se introduce la fecha de última modificación, en formato año-mes-día. En **<changefreq>** es posible introducir always, hourly, daily, weekly, monthly, yearly o never; dependiendo de la frecuencia con que se actualice la página. En **<priority>** debe indicarse la importancia de la página del 0 al 1. No vale de nada decir que todas las URL tienen importancia 1 porque entonces los buscadores ignorarán ese valor.

Si el sitemap se encuentra en **<http://www.misitio.com/sitemap.xml>** los propios buscadores probarán para ver si está y lo utilizarán. De lo contrario, debe añadirse una referencia al archivo robots.txt, la palabra **Sitemap** seguida de un espacio y la URL del sitemap. Por ejemplo:

Sitemap **http://www.misitio.com/sitemap.xml**

El fichero robots.txt se localiza en el directorio raíz utilizado por el dominio. Deberá estar visible desde **<http://www.misitio.com/robots.txt>** para que sea utilizado.



Sabía que...

El fichero robots.txt se utiliza para dar instrucciones a motores de búsqueda y otros servicios web sobre qué páginas pueden rastrear y cuáles no.



Actividades

12. Escriba un documento XML con un sitemap ficticio que incluya las URL de una página web de anuncios clasificados.
-

6. Análisis XML

En esta sección se verán aplicaciones e interfaces para comunicarse con documentos en formato XML. Lo positivo de que XML esté tan extendido y que se utilice tanto, es que hay una gran comunidad de desarrolladores creando utilidades y herramientas que utilizan este estándar.



Importante

Hay gran cantidad de herramientas online gratuitas (muchas de ellas libres también) que pueden ayudar en el día a día al trabajar con XML.

6.1. Herramientas y utilidades de análisis

Aquí se mostrarán diversas aplicaciones y servicios web que pueden ayudar a trabajar con los documentos XML.

Validar documentos XML

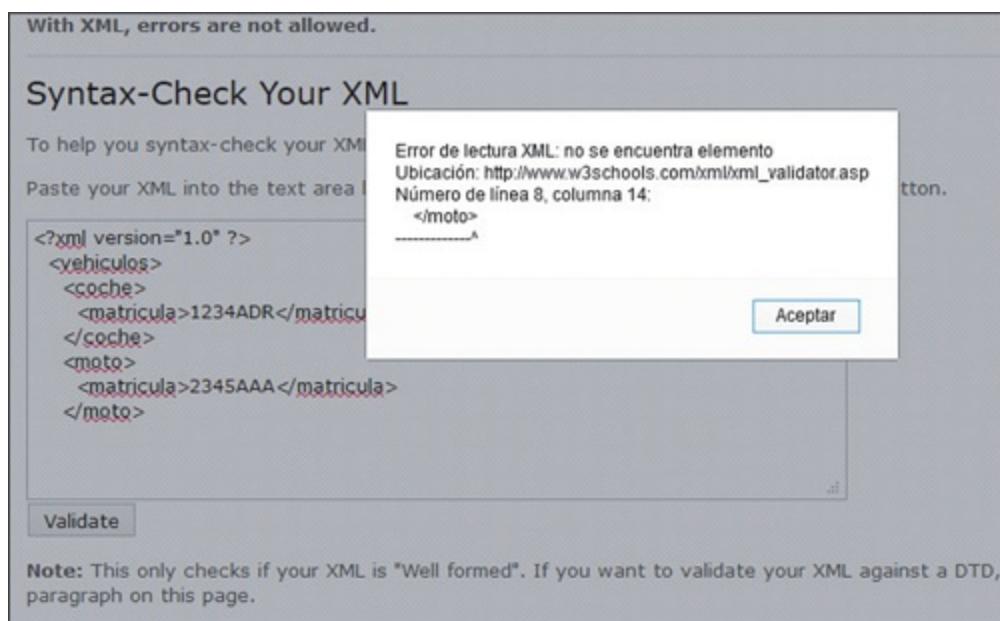
La primera herramienta que se va a ver es una sencilla y sirve para validar documentos XML. Si no se obtienen errores, puede estar seguro de que el documento **podrá ser leído desde cualquier plataforma**.

Hay que tener en cuenta que cuando se valida un documento, solamente se podrá ver si lo que se ha enviado es correcto en este preciso instante. Normalmente, se generan los documentos XML dinámicamente, por lo que su contenido puede variar con el tiempo. Es posible que en el futuro, el contenido de un elemento contenga el carácter "<" o algo así y ya no sea un documento válido. Un validador muy útil es el de <http://www.w3schools.com/xml/xml_validator.asp>. Ahí se puede pegar un documento xml en el cuadro de texto o indicar la URL de un documento existente que esté en Internet. Ese validador es muy simple, porque da la responsabilidad de la validación al navegador del usuario, por lo que cada navegador da mensajes de error distintos y en el idioma del usuario.

Ahora se probará a validar este documento, el cual es incorrecto:

```
<?xml version="1.0" ?>
<vehiculos>
  <coche>
    <matricula>1234ADR</matricula>
  </coche>
  <moto>
    <matricula>2345AAA</matricula>
  </moto>
```

Los resultados que aparecen en Firefox son estos:



Indica que falta algo detrás de </moto>, lo cual es cierto, porque se ha dejado la etiqueta <ve



Actividades

13. Utilice esta herramienta para probar que los anteriores ejercicios que ha realizado a lo largo del capítulo tenían una sintaxis correcta.

Inspector visual de XML

XML sigue una estructura de árboles anidados unos dentro de otros. Esta es una estructura muy fácil de representar gráficamente, sin código XML.

Se utilizará el servicio online <<http://xmlgrid.net/>> para subir un documento XML y poder manipularlo gráficamente, sin necesidad de tocar ninguna línea de XML.

Para probar, se utilizará el siguiente documento XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<productos>
  <producto>
    <nombre>Filetes de añojo de primera</nombre>
    <precio>€5.95</precio>
    <descripcion>Peso aproximado: 400g.</descripcion>
  </producto>
  <producto>
    <nombre>Filete de salmón salvaje</nombre>
    <precio>€7.95</precio>
    <descripcion>Con piel. Peso aproximado de la unidad: 600g.</descripcion>
  </producto>
  <producto>
    <nombre>Aceite de oliva. Garrafa 5l.</nombre>
    <precio>€8.95</precio>
    <descripcion>Aceite de Oliva Virgen Extra.</descripcion>
  </producto>
  <producto>
    <nombre>Lomo de atún</nombre>
    <precio>€4.50</precio>
    <descripcion>Fresco. Peso aproximado, 300g.</descripcion>
  </producto>
  <producto>
    <nombre>Solomillo de cerdo</nombre>
    <precio>€6.95</precio>
    <descripcion>Peso aproximado: 500g.</descripcion>
  </producto>
</productos>
```

Debe copiarse este XML en el cuadro de texto que aparece en la portada de su web. También es posible guardar el documento localmente y subirlo utilizando la opción **Open File**:

The screenshot shows the XmlGrid.net interface. At the top, there's a navigation bar with links like Demo, FAQ, XML Editor, XML Viewer, XML To Text, CSV To XML, XSD To XML, XML To XSD, XPath, Excel To XML, and Forum. Below the navigation bar is a toolbar with buttons for Input Box, Open File, By URL, Create XML, and JSON Editor with converter. The main area contains an input box containing the following XML code:

```
<nombre>Lomo de atún</nombre>
<precio>€4.50</precio>
<descripcion>Fresco. Peso aproximado, 300g.</descripcion>
</producto>
<producto>
<nombre>Solomillo de cerdo</nombre>
<precio>€6.95</precio>
<descripcion>Peso aproximado: 500g.</descripcion>
</producto>
</productos>
```

Below the input box is a "Submit" button. A section titled "View, Edit and Validate XML Document" follows, with instructions to load XML data and validate it against W3C XML 1.0 specifications. It also mentions that if no errors are found, a "Well-Formed XML" stamp will be displayed.

Una vez seleccionado el documento, se presiona **Submit**. Ahí se verá el árbol de nuestro XML. Si se hace clic en las flechas, se desplegarán los elementos que hay en ese árbol:

The screenshot shows the same XmlGrid.net interface after submission. The XML document is now presented as an expandable tree structure. The root element is <productos>. Under it, there is a node for <producto (5)>, which is expanded to show a table with five rows of data:

<> nombre	<> precio	<> descripcion
Filetes de atún de pesca	€5.95	Peso aproximado : 400g.
Filete de salmón salmón	€7.95	Con piel . Peso aproximado de la unidad : 600g.
Acete de Oliva Garrata Sl	€8.95	Acete de Oliva Virgen Extra.
Lomo de atún	€4.50	Fresco. Peso aproximado , 300g.
Solomillo de cerdo	€6.95	Peso aproximado : 500g.

At the bottom left, there's a "Help" section with the following list:

- expand an element: click icon ► in front of the element name.
- collapse an expanded element: click icon ▲ in front of the element name.
- show popup menu: click the icon ↗ or ⓘ in front of an element or attribute.
- delete a comment: double click the comment that you want to work on, then select Delete command from the popup.
- edit the value of an element, attribute or comment: place the cursor to where you want to change, use the delete

Ahí aparecen los datos en formato de tabla, lo cual es mucho más fácil de leer. Es posible hacer clic en cualquier elemento y seleccionar **Show XPath** para ver la ruta

XPath para cada elemento.



Importante

También puede cambiarse el nombre o valor de cualquier elemento.

Puede alternarse entre la vista de un documento XML normal y la de árbol utilizando los botones **TextView** y **GridView** respectivamente.

Cuando se haya terminado de realizar modificaciones, lo ideal es irse a la **TextView** y copiar el XML resultante.

Limpador de XML

Como se vio anteriormente, los saltos de línea en XML son prescindibles. Los saltos de línea y los espacios solo son útiles para los humanos, para leer la información más rápidamente.

Si se trabaja con un documento XML que se encuentra formateado de forma poco legible, se puede utilizar un servicio online llamado <http://www.freeformatter.com/xml-formatter.html> el cual recibe un documento XML y lo devuelve formateado de forma 100% correcta.

Este servicio permite pegar un documento o especificar la URL de un documento publicado en Internet. Ahora se probará con el documento del ejemplo anterior pero muy mal formateado:

```
<?xml version="1.0" encoding="UTF-8"?> <productos> <producto> <nombre>Filetes de ajojo de primera</nombre> <precio>€5.95</precio> <descripcion>Peso aproximado: 400g.</descripcion> </producto> <producto> <nombre>Filete de salmón salvaje</nombre> <precio>€7.95</precio> <descripcion>con piel. Peso aproximado de la unidad: 600g.</descripcion> </producto> <producto> <nombre>Aceite de oliva. Garrafa 5l.</nombre> <precio>€8.95</precio> <descripcion>Aceite de oliva virgen Extra.</descripcion> </producto> <producto> <nombre>Lomo de atún</nombre> <precio>€4.50</precio> <descripcion>Fresco. Peso aproximado, 300g.</descripcion> </producto> <producto> <nombre>Solomillo de cerdo</nombre> <precio>€6.95</precio> <descripcion>Peso aproximado: 500g.</descripcion> </producto> </productos>
```

Si se pega esto en su servicio web, podrá formatearse y obtener una versión mucho más fácil de leer:

The screenshot shows a web-based XML formatter. On the left, there is a sidebar with a tree view of various tools and services, including:

- HTML Validator
- XML Validator - XSD
- XSD Generator
- XPath Tester
- Credit Card Number Generator & Validator
- Regular Expression Tester
- Encoders & Decoders**
- Url Encoder & Decoder
- Base 64 Encoder & Decoder
- QR Code Generator
- Code Minifiers
- JavaScript Minifier
- CSS Minifier
- Converters
- XSLT (XSL Transformer)
- XML to JSON Converter
- JSON to XML Converter
- CSV to XML Converter
- Epoch Timestamp To Date
- Cryptography
- Message Digester
- HMAC Generator
- String Escaper & Utilities
- String Utilities
- HTM Escape
- XML Escape
- Java and .Net Escape
- JavaScript Escape
- CSV Escape
- SQL Escape
- Web Resources
- Locale Ipsum Generator
- LESS Compiler
- List of MIME types
- HTML Entities
- Uri Parser / Query String Splitter
- ISO country list - HTML select snippet
- USA state list - HTML select snippet
- Canada province list - HTML select snippet

The main area contains the following interface elements:

- Option 1: Copy-paste your XML document here**: A text area containing sample XML code.
- Option 2: Or type in the URL to your XML file**: A text input field with the value "http://www.example.com/myfile.xml".
- Indentation level:** A dropdown menu set to "3 spaces per indent level".
- Force output to new window:** A dropdown menu set to "No".
- FORMAT XML**: A blue button.
- Formatted XML:** A large text area showing the input XML code properly indented and formatted.
- COPY TO CLIPBOARD** and **SELECT ALL**: Buttons at the bottom of the formatted XML area.

O bien se verán los resultados dentro de la misma pantalla (como se muestra en la captura) o bien se marcará **Yes** el campo de **Force output to new window**, el cual abrirá una nueva pestaña del navegador con el documento XML que generado.



Importante

Este servicio es útil cuando se trata con algún documento XML creado por terceros, cuando se carece de documentación o cuando se necesita acceder visualmente al documento.

6.2. Programación de análisis XML mediante lenguajes en servidor

Aquí se enseñará cómo crear y leer documentos XML mediante el lenguaje de servidor PHP, que es uno de los más habituales en lo que a programación web se refiere.

Creación de documentos XML de forma simple

Para crear un documento XML lo más sencillo es simplemente crear un **documento de texto** con las etiquetas escritas en él, como si de una cadena de texto cualquiera se tratase.

Normalmente crear un documento XML es lo más sencillo de hacer, lo complejo suele ser leer documentos creados por terceros.

Ahora se mostrará un archivo PHP que genera un documento XML válido con entradas RSS. Se supone que las entradas vienen desde una variable llamada \$entradas que se ha calculado más arriba:

```
1 <?php
2 echo '<?xml version="1.0" encoding="UTF-8"?>';
3 echo '<rss version="2.0">';
4 echo '<channel>';
5 echo '<title>Mi Blog - Últimas entradas</title>';
6 echo '<link>http://miblog.com</link>';
7 echo '<description>Mi blog sobre moda</description>';
8 echo '<language>es</language>';
9
10 foreach ($entradas as $entrada)
11 {
12     echo '<item>';
13     echo '<title><![CDATA['.$entrada["titulo"].']]></title>';
14     echo '<link>'.$entrada["url"].'</link>';
15     echo '<pubDate>'.$entrada["fecha"].'</pubDate>';
16     echo '<description><![CDATA['.$entrada["contenido"].']]></description>';
17     echo '</item>';
18 }
19
20 echo '</channel>';
21 echo '</rss>';
```

El título y la descripción pueden contener datos que hagan el XML inválido, por lo que van dentro de una cláusula CDATA.

En este ejemplo ningún elemento tiene un atributo. Pero en caso de tenerlo, puede que interese utilizar la función addslashes(de PHP si su valor depende de una variable de PHP, para escapar las comillas que pudiera tener y para asegurar que no invalida nuestro XML.

Esta es la manera más sencilla de crear documentos XML, porque se ve directamente cómo va a quedar el documento, por lo que es más rápido. El problema es que hay que tener en cuenta que cualquier dato que sea dinámico debe ser tratado con el **máximo cuidado**, porque puede contener un valor que haga que el XML deje de ser válido. Otro problema es que la responsabilidad de que la estructura sea correcta depende 100% del

programador.



Importante

Si se le olvida cerrar una comilla o etiqueta, el documento será inválido.

Más adelante se verá otra manera de crear estos documentos.



Actividades

14. Cree un documento XML mediante PHP que liste los productos de una tienda. Cada producto tiene un nombre, un precio y una descripción. El precio tiene un atributo que indica la moneda en que se encuentra. Los datos vienen de un array llamado \$productos.
-

Leer documentos XML con PHP

Anteriormente, se explicó en qué consistía el **DOM**. Era una interfaz que permitía comunicarse con documentos XML de una forma orientada a objetos. PHP soporta nativamente un conjunto de clases que implementan el DOM, de tal manera que pueden leerse documentos HTML y XML sin necesidad de librerías externas.



Nota

La documentación completa sobre estas clases se puede encontrar en <<http://us2.php.net/manual/es/intro.dom.php>>.

Para probar, se utilizará este documento, el cual se leerá desde PHP. Es una respuesta real de la API de Google, la cual devuelve información sobre una dirección postal:

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <GeocodeResponse>
3   <status>OK</status>
4   <result>
5     <formatted_address>Plaza de Callao, 28013 Madrid, España</formatted_address>
6     <address_component>
7       <long_name>Plaza de Callao</long_name>
8       <short_name>Plaza de Callao</short_name>
9     </address_component>
10    <address_component>
11      <long_name>Madrid</long_name>
12      <short_name>Madrid</short_name>
13      <type>locality</type>
14      <type>political</type>
15    </address_component>
16    <address_component>
17      <long_name>Madrid</long_name>
18      <short_name>M</short_name>
19      <type>administrative_area_level_2</type>
20      <type>political</type>
21    </address_component>
22    <address_component>
23      <long_name>Comunidad de Madrid</long_name>
24      <short_name>Comunidad de Madrid</short_name>
25      <type>administrative_area_level_1</type>
26      <type>political</type>
27    </address_component>
28    <address_component>
29      <long_name>España</long_name>
30      <short_name>ES</short_name>
31      <type>country</type>
32      <type>political</type>
33    </address_component>
34    <address_component>
35      <long_name>28013</long_name>
36      <short_name>28013</short_name>
37      <type>postal_code</type>
38    </address_component>
39    <geometry>
40      <location>
41        <lat>40.4200236</lat>
42        <lng>-3.7058124</lng>
43      </location>
44      <location_type>APPROXIMATE</location_type>
45      <viewport>
46        <southwest>
47          <lat>40.4115284</lat>
48          <lng>-3.7218198</lng>
49        </southwest>
50        <northeast>
51          <lat>40.4285178</lat>
52          <lng>-3.6898050</lng>
53        </northeast>
54      </viewport>
55    </geometry>
56  </result>
57</GeocodeResponse>
--
```

Como puede apreciarse, devuelve la lista de componentes de la dirección, junto con información sobre en qué consiste ese componente. Al final también aporta coordenadas sobre el lugar, lo cual es muy útil.

Ahora se creará un script de PHP que muestre primero la dirección formateada, después la lista de elementos separada por comas. Luego pondrá las coordenadas en la forma (\$latitud, \$longitud:

Se supone que se ha leído el archivo en forma de texto y que dicho texto se encuentra en la variable \$xml:

```

1 <?php
2 $dom = new DOMDocument();
3 $dom->loadXML($xml);
4
5 $result = $dom->getElementsByTagName('result')->item(0);
6
7 $formatted_address = $result->getElementsByTagName('formatted_address')->item(0);
8
9 echo '<h1>' . $formatted_address->nodeValue . '</h1>';
10
11 $array_componentes = array();
12 foreach ($result->getElementsByTagName('address_component') as $address_component)
13 {
14     $long_name = $address_component->getElementsByTagName('long_name')->item(0);
15
16     $array_componentes[] = $long_name->nodeValue;
17 }
18
19 echo '<h2>' . implode(', ', $array_componentes) . '</h2>';
20
21 $location = $result->getElementsByTagName("location")->item(0);
22
23 $lat = $location->getElementsByTagName("lat")->item(0);
24 $lng = $location->getElementsByTagName("lng")->item(0);
25
26 echo '<h3>(' . $lat->nodeValue . ', ' . $lng->nodeValue . ')</h3>';
27

```

Primero se crea un objeto de la clase `DOMDocument`, que representará el documento que se va a leer. A continuación, se llama al método `loadXML` con un parámetro que es la cadena de texto en formato XML que se utilizará.

El método `loadXML` espera un XML 100% válido y si encuentra cualquier tipo de deficiencia, devolverá muchos mensajes de error. Si hay que trabajar con documentos formateados de forma ligeramente incorrecta, puede interesar cargar el XML utilizando `@dom->loadXML`, que esconderá todas las advertencias y tratará de leer el XML como pueda.



Importante

Además del método `loadXML`, existe el método `loadHTML`, si se prefiere trabajar con HTML en vez de XML.

Una vez se ha cargado el XML, en el objeto `$dom` podrá hacerse pleno uso de la interfaz DOM para acceder a todos los elementos del documento.

La manera de acceder de forma directa a los elementos es mediante los métodos `getElementsByTagName` y `getElementById`. El primero devuelve un objeto de tipo `DOMNodeList` y el segundo, uno de tipo `DOMElement`.

Un objeto `DOMNodeList` contiene una lista de elementos (nodos y uno de tipo `DOMEElement` contiene un elemento en particular).

Lo habitual es recorrer nodos hasta dar con el `DOMEElement` deseado.

Con `$dom->getElementsByName('result')` se obtiene una lista de etiquetas con nombre **result**. En este caso solo hay una y en caso de que hubiera varias, solo se desea trabajar con la primera, por lo que puede encadenarse el método `item()`, el cual devuelve el `DOMEElement` cuyo índice sea el parámetro recibido.

En este caso, se selecciona el elemento número cero, que es el primero (se empieza en cero).

Un objeto de tipo `DOMNodeList` puede ser recorrido de forma muy similar a un objeto de tipo `DOMDocument`, podrá volver a usarse `getElementsByName` sobre él.

Estos objetos son iterables, por lo que es posible utilizarlos en un bucle *foreach*. En cada iteración, la variable de iteración contendrá el `DOMEElement` actual.

Cada `DOMEElement` puede tener una serie de atributos y un valor general (que es el contenido de la etiqueta. Para obtener su valor se utiliza la propiedad `nodeValue`. Para acceder a un atributo, se emplea el método `getAttribute`, pasando por parámetro el nombre del atributo deseado.

Es posible hacer un uso mucho más complejo, pero solo queremos ver un ejemplo de un uso real (y útil del conjunto de clases del DOM para PHP).

Crear documentos XML utilizando las clases DOM de PHP

Antes se estudió cómo crear documentos XML de forma sencilla, tratándolos como texto normal. Ahora se creará un documento utilizando **orientación a objetos** y sin necesidad de escribir una sola línea de código XML.

Esto aporta la ventaja de ser un método más **potente y fiable**, con la contrapartida de que cae mucho el **rendimiento** y que pierde un poco de flexibilidad, porque no se dispone de tanta libertad.

Se mostrará cómo construir un documento XML utilizando el DOM, es decir, creando elementos, anidando nodos dentro y dando valor a sus atributos. Para probar, se creará

el sitemap de una página web:

```
1 <?php
2 $urls = array(
3     "http://www.misitio.com" => 1,
4     "http://www.misitio.com/página1.html" => 0.7,
5     "http://www.misitio.com/otra_pagina.html" => 0.5,
6     "http://www.misitio.com/página3.html" => 0.6,
7     "http://www.misitio.com/registro.php" => 0.4,
8     "http://www.misitio.com/login.php" => 0.4,
9 );
10
11 $dom = new DOMDocument("1.0");
12 $dom->encoding = "utf-8";
13
14 $padre = $dom->createElement("urlset");
15 $padre->setAttribute("xmlns", "http://www.sitemaps.org/schemas/sitemaps/0.9");
16
17 foreach ($urls as $url => $prioridad)
18 {
19     $url_elemento = $dom->createElement("url");
20
21     $loc_elemento = $dom->createElement("loc");
22     $texto = $dom->createTextNode($url);
23     $loc_elemento->appendChild($texto);
24
25     $priority_elemento = $dom->createElement("priority");
26     $texto = $dom->createTextNode($prioridad);
27     $priority_elemento->appendChild($texto);
28
29     $url_elemento->appendChild($loc_elemento);
30     $url_elemento->appendChild($priority_elemento);
31
32     $padre->appendChild($url_elemento);
33 }
34
35 $dom->appendChild($padre);
36
37 header('Content-Type: application/xml; charset=utf-8');
38 echo $dom->saveXML();
39
```

Primero, se parte de un *array* asociativo con las URL y su prioridad. Esto está puesto “a mano”; lo normal es que esa lista viniera de una base de datos o se leyera de algún sitio.

Después, se crea un objeto de la clase DOMDocument vacío. En este ejemplo, se especifica que sea de la versión 1.0 y que tenga el encoding UTF-8.

Luego, en vez de utilizar loadXML, se crearán elementos y se adjuntarán al documento.

Al principio, si no se añade ninguna etiqueta, el documento contendrá solamente <?xml version="1.0" encoding="utf-8"?>. Primero habrá que crear la etiqueta padre, que tiene que tener de nombre **urlset**. Para ello, se emplea el método createElement(. El parámetro que recibe es el nombre de la etiqueta.

Este método devuelve un objeto de la clase DOMElement. Ahora, si se quiere manipular este elemento, habrá que referirse a él por medio de este objeto que se acaba de crear.

Hay que darle el espacio de nombres que haga que sea un sitemap, por lo que se usará el método setAttribute. Su primer parámetro es el nombre del atributo y el segundo es su valor.

Tenga en cuenta que se ha creado el elemento pero **todavía no se ha incluido al documento**. Simplemente se ha creado una referencia a él, pero para que se incluya al documento hay que utilizar el método appendChild(del elemento al que se quiera incluir. Hasta que no se hace \$dom->appendChild(\$padre, el elemento no es incluido al documento.

Antes de incluirlo, se recorre el *array* de URL, para crear cada elemento de <url>.

Primero se crea el elemento <url>. A continuación, se crean los elementos <loc> y <priority>. Los sitemaps permiten más etiquetas, pero se han utilizado solo estas dos, recuerde que solo <loc> es obligatoria.

Las etiquetas <loc> y <priority> contienen texto, en vez de otras etiquetas, por lo que debe crearse un nodo de tipo texto, utilizando \$dom->createTextNode(pasando el texto a utilizar como parámetro.

Primero se crea la etiqueta <loc>, luego se crea el nodo de texto que utiliza <loc>, después se añade ese nodo a <loc> y por último, se añade <loc> a <url>.

Los nodos de texto suelen causar bastante confusión. Se crea un nodo de texto y luego se añade a la etiqueta principal. **No es posible** utilizar el mismo método que al leer los documentos, cuando bastaba con llamar al atributo nodeValue para obtener el texto.

Una vez se tiene el elemento <url> completo, se añade al elemento padre (urlset).

Cabe destacar que se utiliza programación orientada a objetos, por lo que no es necesario que los elementos se añadan cuando ya han sido completados. Aquí se realiza como convención y para que sea más legible, pero podrían crearse los elementos, utilizar appendChild y después definir su contenido y propiedades.

Cuando se termina de crear el documento, se añade una cabecera HTTP expresando que el contenido que se muestra es de tipo XML.

Para convertir el objeto \$dom en XML válido, se utiliza el método saveXML(), el cual devuelve una cadena de texto en formato XML.

Esta manera de manipular documentos XML aporta también la ventaja de que es posible **leer un documento XML y manipularlo in situ**. No siempre se van a crear documentos XML desde cero, sino que es posible que haya que leer un XML para generar otro.



Nota

Esto es posible porque puede utilizarse loadXML() y luego seguir manipulando ese documento igual que como se ha visto aquí.



Actividades

15. Cree el mismo documento RSS que se vio en el apartado de “Creación de documentos XML de forma simple” pero utilizando el conjunto de clases del DOM.
-



Aplicación práctica

Cree el siguiente documento XML utilizando las clases del DOM que PHP proporciona:

```

<?xml version="1.0" encoding="UTF-8"?>
<productos>
    <producto activo="si">
        <titulo>Limpiador de madera</titulo>
        <precio moneda="eur">2.99</precio>
    </producto>
    <producto activo="si">
        <titulo>Friegasuelos</titulo>
        <precio moneda="eur">1.50</precio>
    </producto>
    <producto>
        <titulo>Bayeta de microfibra</titulo>
        <precio moneda="gbp">1</precio>
    </producto>
</productos>

```

SOLUCIÓN

```

1  <?php
2  $productos = array(
3      array("titulo" => "Limpiador de madera", "activo" => "si", "precio" => 2.99, "moneda" => "eur"),
4      array("titulo" => "Friegasuelos", "activo" => "si", "precio" => 1.50, "moneda" => "eur"),
5      array("titulo" => "Bayeta de microfibra", "activo" => NULL, "precio" => 1, "moneda" => "gbp"),
6  );
7
8  $dom = new DOMDocument("1.0");
9  $dom->encoding = "utf-8";
10
11 $padre = $dom->createElement("productos");
12
13 foreach ($productos as $producto)
14 {
15     $producto_elemento = $dom->createElement("producto");
16
17     if ($producto["activo"] !== NULL)
18     {
19         $producto_elemento->setAttribute("activo", $producto["activo"]);
20     }
21
22     $titulo_elemento = $dom->createElement("titulo");
23     $texto = $dom->createTextNode($producto["titulo"]);
24     $titulo_elemento->appendChild($texto);
25
26     $precio_elemento = $dom->createElement("precio");
27     $texto = $dom->createTextNode($producto["precio"]);
28     $precio_elemento->appendChild($texto);
29     $precio_elemento->setAttribute("moneda", $producto["moneda"]);
30
31     $producto_elemento->appendChild($titulo_elemento);
32     $producto_elemento->appendChild($precio_elemento);
33
34     $padre->appendChild($producto_elemento);
35 }
36
37 $dom->appendChild($padre);
38
39 header("Content-Type: application/xml; charset=utf-8");
40 echo $dom->saveXML();

```

7. Uso de XML en el intercambio de información

Como se ha ido viendo, el principal motivo por el que se desarrolló XML era para compartir información entre diversas plataformas. Esto es así porque el formato está muy bien estructurado, se comparte en formato de texto convencional y es muy fácil de leer.

Está implementado en multitud de protocolos que son de uso diario en programación. En este apartado se verán algunos de ellos.

Codificación de parámetros

Es común utilizar XML para comunicar diversas aplicaciones informáticas. Cuando una aplicación o servicio necesita comunicarse con otro, deben compartir un formato común que les permita entenderse.

XML se puede utilizar para enviar la lista de parámetros que una aplicación debe procesar.

Por ejemplo, un navegador que utilice AJAX, puede comunicarse con el servidor web mandando la lista de parámetros utilizando XML. A lo largo de este apartado veremos diversos protocolos basados en XML que envían los parámetros utilizando XML.

Ficheros de configuración basados en XML

Como ya se ha visto, una de las principales ventajas de XML es que resulta fácil de leer por humanos y por computadores. Por ello, es ideal para crear ficheros de configuración, porque el usuario puede entenderlos fácilmente y manipularlos sin apenas riesgo.

Son muchas las aplicaciones informáticas que utilizan XML para sus ficheros de configuración. Por ejemplo, al desarrollar aplicaciones utilizando el entorno .NET de Microsoft o al realizar configuraciones en aplicaciones de Android, se utiliza el formato XML.



Nota

Además, XML utiliza una estructura de anidamiento, lo cual es de ayuda para agrupar distintos parámetros o para establecer jerarquías.

AJAX

AJAX significa *Asynchronous Javascript And XML* (Javascript y XML asíncrono). Se utiliza en programación web en miles de páginas web distintas. Sirve para poder interactuar con el servidor web sin necesidad de recargar la página.

En sus inicios, AJAX siempre se utilizaba junto a XML, pero en la actualidad se utiliza también junto a JSON, por lo que la equis de AJAX no tiene tanto sentido ahora.

El protocolo AJAX consiste en enviar una serie de parámetros GET o POST a través de Javascript. El servidor recibe esos parámetros y devuelve contenido en formato XML con los datos que necesita el cliente en ese momento. Javascript recibe ese XML, lo lee y utiliza sus datos de la forma en que haga falta.

API

Una API (*Application Programming Interface* / Interfaz de programación de aplicaciones) es un servicio que permite que un programa se conecte a una fuente de datos externa.

Existen API para muchas cosas, por ejemplo una API de Google Maps en la que se envía una dirección en formato texto y Google devuelve unas coordenadas de GPS exactas.

Podría cargarse esta URL:

```
<http://maps.googleapis.com/maps/api/geocode/xml?address=1600+Amphitheatre+Parkway,+Mountain+View,+CA&sensor=false>
```

Ese está solicitando información sobre la dirección “1600 Amphitheatre Parkway, Mountain View, CA” y Google ofrece multitud de datos sobre la dirección exacta, las coordenadas, etc., en formato XML.



Nota

Podría crearse un pequeño programa que procesara ese XML y utilizara esos datos para algo, como se vio antes con PHP.



Actividades

16. ¿Por qué se utiliza XML para intercambiar información entre diversas aplicaciones?
-

XML-RPC

XML-RPC significa Protocolo de Llamada a Procedimientos. Sirve para utilizar XML y HTTP para **comunicar dos programas distintos**. Fue desarrollado originalmente para Microsoft en 1998.

Sirve para que un programa llame a otro y decirle qué debe hacer. El programa (a) manda un mensaje XML al programa (b). Este lee qué es necesario hacer, lo realiza y manda una respuesta XML a (a) explicando los resultados.

Es un sistema fuertemente tipado y pensado para aplicaciones bastante fiables.



Nota

XML-RCP puede ser utilizado en API que requieran de mucha seguridad o gran complejidad.

SOAP

SOAP significa *Simple Object Access Protocol* (Protocolo Simple de Acceso a Objetos). Fue creado como evolución de XML-RPC. Es más extensible y se creó para suplir las limitaciones que XML-RPC tenía al trabajar con clases y objetos.

Está aprobado por el W3C y se utiliza para aplicaciones en las que un alto nivel de fiabilidad sea un requisito imprescindible.

8. Resumen

Los lenguajes de marcado permiten compartir información entre diversos sistemas utilizando documentos de texto que, además de ser fáciles de leer por computadores, son fáciles de leer por humanos.

Los lenguajes de marcado por excelencia son el HTML y el XML. Este capítulo se ha centrado en XML. Sus grandes bazas son el soporte de caracteres internacionales, su gran difusión y su gran legibilidad.

XML consiste en etiquetas que se colocan entre signos de menor que y mayor que (<>), las cuales pueden contener otras etiquetas dentro o bien contener valores textuales. Cada etiqueta puede tener una serie de atributos. Toda etiqueta abierta debe ser cerrada.

XML está pensado para ser utilizado en servicios de la World Wide Web, como pueden ser las páginas web o servicios web.

El W3C es el órgano que regula la especificación oficial de XML y de la mayoría de sus estándares. Algunos de los estándares más usados son los de las plantillas DTD para restringir qué elementos deben aparecer en un documento XML y cómo. También son comunes las XSLT (hojas de estilos de XML, que permiten transformar un documento XML simple en otro documento como un documento HTML, LaTeX, etc. Cualquier documento cuyo formato interno sea texto.

XML proporciona diversos estándares para comunicarse con sus documentos como el DOM, XPath y XQuery. Todos ellos son estándares que buscan reflejar el árbol de nodos.

Dada la gran difusión de XML existen muchas utilidades que pueden ayudar para trabajar con documentos en este formato. Se cuenta con validadores, editores visuales, formateadores, etc.

Una manera muy habitual de leer y manipular documentos XML es mediante **lenguajes de servidor** como PHP o JSP. Es posible crear documentos XML dinámicamente cuyos contenidos provengan de diversas fuentes de datos como una base de datos o una API. También es posible leer documentos ya creados de tal forma que puedan utilizarse como una fuente de datos más.



Ejercicios de repaso y autoevaluación

1. ¿De qué lenguaje proviene el XML?

- a. Del SGML
- b. Del HTML
- c. Del XHTML

2. Determine si las siguientes afirmaciones son verdaderas o falsas:

El DOM es un API que permite manipular documentos HTML y XML.

- Verdadero
- Falso

Solo es necesario cerrar una etiqueta abierta cuando esta tiene contenido dentro.

- Verdadero
- Falso

Los únicos atributos cuyo valor va entre comillas son los que contienen texto.
No es necesario utilizar comillas para valores numéricos o booleanos.

- Verdadero
- Falso

El fin de la web semántica es que cada elemento aporte información entendible por aplicaciones informáticas, además de por humanos.

- Verdadero
- Falso

3. ¿Qué ocurre si el contenido de una etiqueta contiene caracteres especiales como > o <?

- a. No pasa nada.
- b. Es posible escapar esos caracteres utilizando > y <.
- c. Puede encerrarse todo el contenido entre las etiquetas CDATA.

- d. No es posible almacenar esa información.

4. ¿Cuál es el papel del W3C con respecto al XML?

5. ¿Por qué XML está tan extendido y se utiliza en tantos ámbitos distintos?

6. ¿Qué es el siguiente documento?

```
<!ELEMENT noticia (descripcion)>
<!ELEMENT descripcion (#CDATA)>
<!ELEMENT url (#CDATA)>
```

- a. Una declaración DTD
- b. Un documento XML Schema
- c. Una hoja de estilos XML (XSL).

7. Las hojas de estilos son útiles principalmente porque...

- a. ... permiten dar un diseño más vistoso al leer un documento XML.
- b. ... permiten especificar qué elementos deben aparecer en un documento XML junto a la lista de restricciones.
- c. ... permiten convertir un documento XML a otro formato.

8. ¿Qué ventajas aporta XHTML con respecto a HTML?

9. XPath se utiliza para...

- a. ... manipular documentos XML
- b. ... seleccionar uno o varios elementos de un documento XML.

- c. Es una alternativa al DOM.
- 10. Dado el siguiente documento, escriba las expresiones XPath que solucionarían cada caso:**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE entradas [
    <!ELEMENT libros (libro)>
    <!ELEMENT libro (titulo,autor)>
    <!ELEMENT titulo (#PCDATA)>
    <!ELEMENT autor (nombre,apellidos?)>
    <!ELEMENT nombre (#PCDATA)>
    <!ELEMENT apellidos (#PCDATA)>
]>
<libros>
    <libro codigo="120">
        <titulo>Desarrollo de aplicaciones web</titulo>
        <autor>
            <nombre>Juan</nombre>
            <apellidos>Pérez</apellidos>
        </autor>
    </libro>
    <libro codigo="88">
        <titulo>Programación orientada a objetos</titulo>
        <autor extranjero="si">
            <nombre>John</nombre>
            <apellidos>McRoberts</apellidos>
        </autor>
    </libro>
    <libro codigo="200">
        <titulo>Programación en entorno del cliente</titulo>
        <autor extranjero="si">
            <nombre>Philip</nombre>
            <apellidos>Stewart</apellidos>
        </autor>
    </libro>
</libros>
```

- a. Acceda al título del libro cuyo código sea 88.
b. Acceda al segundo libro de la lista.
c. Acceda a los libros cuyo autor sea extranjero.

- 11. ¿Qué propósito tiene el lenguaje XQuery?**
-
-

12. Los estándares de XML Signature y XML Encryption requieren herramientas externas para ser generados:

- Verdadero
- Falso

13. El estándar RSS se utiliza para...

- a. ... proporcionar una fuente de información actualizada con enlaces a un sitio web.
- b. ... contener las últimas noticias de un periódico.
- c. ... compartir el contenido de un sitio web utilizando un formato único, de tal forma que pueda ser utilizado en cualquier otra plataforma.
- d. Todas las respuestas anteriores son correctas.

14. ¿Los documentos XML se pueden crear y leer desde cualquier lenguaje de programación del entorno del servidor?

- a. Sí, todos suelen contar con librerías que permiten trabajar con estos documentos.
- b. Sí, pero es común que haya que pasar el texto a mano para separar la información de tal manera que sea usable.
- c. No. Solo PHP y JSP cuentan con librerías para esto.

15. ¿Para qué sirve AJAX?

Bibliografía

Monografías

- DESONGLES C., Juan: *Ayudantes técnicos de informática de la Junta de Andalucía.* Sevilla: Editorial MAD S.L., 2005.
- DUBOIS, P., HINZ, S. y PEDERSEN C.: *MySQL 5.0. Certification Study Guide.* MySQL AB. MySQL Press, 2006.
- RAMOS, M.J., RAMOS, A. y MONTERO, F.: *Sistemas gestores de bases de datos.* Editorial Mc Graw Hill. 2006.
- SCHWARTZ, Baron [et al.]: *High Performance MySQL.* California: O'Reilly, 2008.

Textos electrónicos, bases de datos y programas informáticos

- Introducción a XML Signature y XML Encryption por Philippe Camacho, de: <<http://users.dcc.uchile.cl/~pcamacho/tutorial/web/xmlsec/xmlsec.html>>.
- Manuales de W3 Schools, de: <<http://www.w3schools.com/>>.