

UNIVERSIDAD



DE VALLADOLID

E.T.S.I. TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA EN TECNOLOGÍAS ESPECÍFICAS DE
TELECOMUNICACIÓN, MENCIÓN EN TELEMÁTICA

Estudio del Estado del Arte de las Tecnologías de Desarrollo para Aplicaciones Web

Autora:

Elena Melero Villamuza

Tutoras:

Míriam Antón Rodríguez

M^a Ángeles Pérez Juárez

Valladolid, 18 de Julio de 2016

TÍTULO: **Estudio del Estado del Arte de las
Tecnologías de Desarrollo para
Aplicaciones Web**

AUTORA: **Elena Melero Villamuza**

TUTORAS: **Míriam Antón Rodríguez
M^a Ángeles Pérez Juárez**

DEPARTAMENTO: **Teoría de la Señal y Comunicaciones e Ingeniería
Telemática**

TRIBUNAL

PRESIDENTE: **Dña. M^a Ángeles Pérez Juárez**

VOCAL: **D. David González Ortega**

SECRETARIO **Dña. Míriam Antón Rodríguez**

SUPLENTE **D. Mario Martínez Zarzuela**

SUPLENTE **D. Francisco Javier Díaz Pernas**

FECHA: **18 de Julio de 2016**

CALIFICACIÓN:

Resumen de TFG

El objetivo principal de este Trabajo Fin de Grado es la realización de una revisión del estado del arte de diversas tecnologías utilizadas en el desarrollo de aplicaciones web, así como la generación de contenidos y la implementación de una batería de ejemplos prácticos para utilizar como apoyo a la docencia en asignaturas que abordan estas tecnologías.

En particular, se realiza un estudio del posicionamiento web o SEO, analizando las diferentes técnicas de desarrollo web que se utilizan en la actualidad. También se revisa y analiza el proceso de diseño de bases de datos relacionales, complementando la teoría con varios ejemplos. Por último, se estudia en profundidad el lenguaje SQL, basándose en el SGBD MySQL. Como complemento, se diseña una base de datos específica, que se utiliza como caso de estudio para realizar ejemplos prácticos con carácter didáctico.

Palabras clave

Bases de datos relacionales, SQL, MySQL, posicionamiento web, SEO

Abstract

The aim of this paper is the study of the state of the art of several web application development technologies, as well as the content generation and the implementation of practical examples, which will be used as learning material in related subjects.

In particular, it focuses on Search Engine Optimization (SEO) through the analysis of current web development technologies. On the other hand, this paper discusses the relational database design process using theory in combination with various examples. Lastly, SQL programming language (based on the MySQL RDBMS) is also explored. In addition, a specific database is designed in order to use it for practical examples.

Keywords

Relational database, SQL, MySQL, Search Engine Optimization, SEO

INDICE

1	INTRODUCCIÓN.....	6
	PRIMERA PARTE	8
2	POSICIONAMIENTO WEB.....	9
2.1	INTRODUCCIÓN AL POSICIONAMIENTO WEB	9
2.1.1	¿Cómo funciona un motor de búsqueda?	10
2.1.1	Factores a tener en cuenta	11
2.2	CONTENIDO	14
2.2.1	Calidad y actualidad.....	14
2.2.2	Palabras clave.....	15
2.3	HTML.....	17
2.3.1	Etiqueta <TITLE>	18
2.3.2	Metadatos (etiqueta <META>)	19
2.3.2.1	Meta Description	19
2.3.2.2	Meta Robots.....	19
2.4	ARQUITECTURA.....	20
2.4.1	Dominio	20
2.4.2	Contenido indexable.....	21
2.4.3	Estructura de enlaces	23
2.4.3.1	Robots.txt.....	24
2.4.3.2	Atributo REL="NOFOLLOW"	28
2.4.3.3	Archivo <i>Sitemap</i>	29
2.4.4	URLs.....	34
2.4.4.1	Contenido duplicado.....	36
2.5	OTRAS TÉCNICAS PENALIZADAS	41
2.6	FUTURO SEO	42
	SEGUNDA PARTE	44
3	SISTEMAS Y GESTORES DE BASES DE DATOS	45
3.1	INTRODUCCIÓN A LOS SISTEMAS Y GESTORES DE BASES DE DATOS	45
3.1.1	Sistemas de información (SI).....	45
3.1.2	De los ficheros a las bases de datos.....	46
3.1.3	Sistemas de Bases de Datos.....	47
3.1.4	Sistemas Gestores de Bases de Datos	48
3.1.5	Evolución de los modelos de BD y los SGBD.....	49
3.2	SISTEMAS DE BASES DE DATOS RELACIONALES.....	51
3.2.1	Definiciones básicas.....	51
3.2.2	Claves	52
3.2.3	Reglas de integridad	54
3.3	¿CÓMO DISEÑAR UNA BASE DE DATOS?	54
3.3.1	Introducción	54
3.3.2	Relaciones entre entidades	55
3.3.3	Normalización.....	56
3.4	EJEMPLOS DE DISEÑO	58
3.4.1	Ejemplo A.....	58
3.4.2	Ejemplo B.....	61
3.4.3	Ejemplo C	63
3.4.4	Ejemplo D	66

3.4.5	Ejemplo líneas de metro.....	69
4	LENGUAJE SQL	71
4.1	INTRODUCCIÓN	71
4.2	BASE DE DATOS LINEASMETRO	72
4.3	SQL DATA DEFINITION LANGUAGE (DDL)	72
4.3.1	Sentencia CREATE	73
4.3.1.1	Definición de campos	73
4.3.1.2	Creación de tablas a partir de registros existentes	81
4.3.1.3	Creación lineasmetro.....	82
4.3.2	Sentencia ALTER.....	85
4.3.3	Sentencia DROP.....	86
4.4	SQL DATA MANIPULATION LANGUAGE (DML).....	86
4.4.1	Consultas de Modificación: INSERT, UPDATE y DELETE	86
4.4.1.1	Sentencia INSERT	87
4.4.1.2	Sentencia UPDATE	91
4.4.1.3	Sentencia DELETE.....	93
4.4.2	Consulta de Selección: SELECT	94
4.4.2.1	Ejercicios.....	126
5	CONCLUSIONES	129
6	FUTURAS LÍNEAS DE DESARROLLO	130
7	BLOGRAFÍA.....	132
8	ANEXO 1.....	135

1 INTRODUCCIÓN

Vivimos en la era de Internet. No hay día que no realicemos una búsqueda en internet, lo utilicemos para comunicarnos con alguien, para saber qué tiempo va a hacer o consultemos alguna red social. Además, no tiene pinta de ser algo pasajero, más bien es una presencia que crece continuamente y que seguirá haciéndolo en el futuro.

Centrándonos en las páginas web, los usuarios no se conforman con obtener la información. Factores de usabilidad como la rapidez de carga, una buena experiencia móvil, facilidad a la hora de encontrar dicha información, etc. son esenciales. Si nuestro sitio web no satisface estas necesidades, probablemente haya otros cientos de páginas web que sí lo hagan. Estas circunstancias favorecen la aparición de numerosas tecnologías relacionadas con el desarrollo web, así como la rápida de evolución de otras ya existentes.

Especialmente demandadas son las páginas web dinámicas, que permiten una mayor interactividad con el usuario y que utilizan, entre otras muchas tecnologías, bases de datos, en las que almacenan la información que será mostrada al usuario en función de las acciones que realice. Para gestionar dichas bases de datos, resulta fundamental conocer en profundidad el lenguaje SQL, ya que uno de los sistemas gestores más populares en el ámbito del desarrollo web es MySQL. Debido a su importancia en el panorama web actual, diseño de bases de datos y lenguaje SQL, se estudian en el presente proyecto.

El desarrollar un sitio web que utilice las últimas tecnologías y que proporcione una excelente experiencia al usuario, aunque necesario, no sirve de nada si nuestra página web se pierde entre los miles de sitios web disponibles en internet. En la última década se ha establecido el uso de técnicas que nos ayudan a hacer visible nuestro sitio web en un mercado cada vez más competitivo, es decir, aquellas técnicas que tienen como objetivo mejorar la posición de la página entre los resultados de los buscadores. Estas técnicas reciben el nombre de posicionamiento web o SEO y constituyen el otro objeto de estudio de este proyecto.

Los temas se enfocan desde un punto de vista didáctica, ya que este proyecto tiene como objetivo la generación y actualización de documentación para las asignaturas Tecnologías para aplicaciones Web (TAW), perteneciente la titulación Grado en Ingeniería de Tecnologías Específicas de Telecomunicación, Mención en Telemática y para Laboratorio de Desarrollo de Sistemas Telemáticos (LDST), perteneciente al Grado en Ingeniería de Tecnologías de Telecomunicación. Ambas impartidas en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universidad de Valladolid.

En concreto, la parte dedicada al posicionamiento web, se desarrolla como un documento de introducción a la materia, que hasta ahora no se ha tratado en la asignatura y a la que previsiblemente, se dedicará un seminario. La parte de bases

de datos, tiene como objetivo la formalización de los apuntes existentes, en especial, la parte teórica de los mismos (teniendo en cuenta que en la asignatura sólo disponen de una clase para este tema), así como el diseño de una base de datos, que se utiliza en la sección dedicada a SQL para ejemplificar las operaciones que nos permite realizar dicho lenguaje.

El presente proyecto se divide en dos partes bien diferenciadas: el estudio del posicionamiento web [Sección 2] y el del diseño e implementaciones de bases de datos relacionales [Secciones 3 y 4].

La primera parte funciona como una guía de introducción al posicionamiento web o SEO, que juega un papel esencial en el éxito de una página web y que, actualmente, está en pleno auge. En dicha sección estudiamos algunos de los factores más influyentes en el posicionamiento web, junto con las técnicas que nos permiten manipular dichos factores y para finalizar, se da una pequeña visión de lo que se prevé que tendrá más importancia en un futuro próximo.

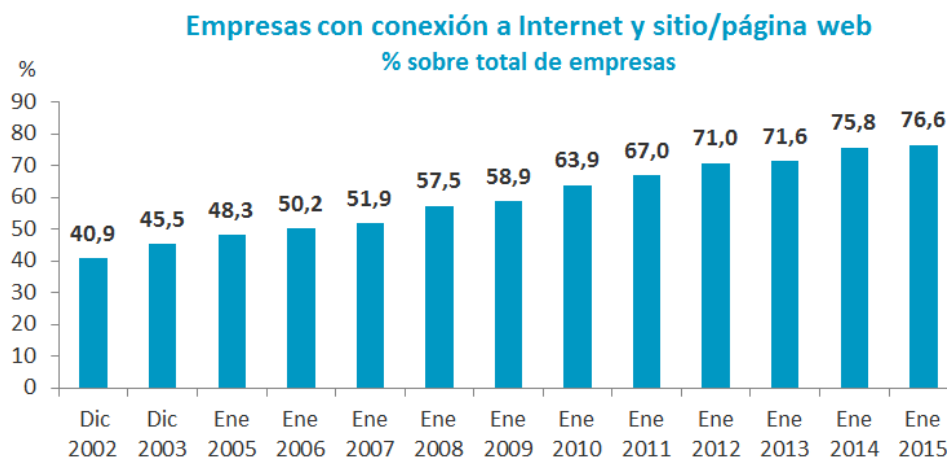
La segunda parte, está dedicada a las bases de datos. En la Sección 3, se realiza una introducción a los sistemas de bases de datos y su evolución, centrándonos más adelante en los sistemas relacionales y en el diseño de bases de datos relaciones. Se realizan varios ejemplos y el último de ellos, el más complejo se utiliza en la Sección 4, que se centra en el lenguaje SQL. Con ayuda de ejemplos realizados sobre dicha base de datos, estudiamos en detalle las diferencias sentencias proporcionadas por este lenguaje.

PRIMERA PARTE

2 POSICIONAMIENTO WEB

2.1 INTRODUCCIÓN AL POSICIONAMIENTO WEB

A lo largo del siglo XXI el uso de internet ha aumentado hasta convertirse en una presencia constante en el día a día de gran parte de la población (una media de 76% de la población europea utiliza internet diariamente según un estudio realizado por Eurostat en 2015). La evolución de las páginas web, el desarrollo de las tecnologías de conexión a internet, los dispositivos móviles y la aparición de las redes sociales han contribuido a ello. En esta sección nos centraremos en las páginas web, que desde la creación de la World Wide Web en 1992 han evolucionado y se han generalizado, especialmente en el ámbito empresarial. Actualmente, las páginas web, en combinación con una creciente presencia en redes sociales, son una parte fundamental del marketing de la empresa. Según una encuesta realizada por el Instituto Nacional de Estadística (INE), a principios del año 2015 un 76,6% de las empresas españolas encuestadas disponía de página web [ver Figura 1].



Fuente: INE

Figura 1: Evolución del número de empresas españolas con página web. Fuente: referencia [8]

En un entorno cada vez más competitivo y saturado, resulta difícil conseguir visibilidad para nuestro sitio web. El objetivo de una página web no sólo es que la visiten aquellos usuarios que conocen el nombre completo de la misma, sino atraer también a aquellos que no la conozcan, pero que puedan estar interesados en su contenido.

Aunque hay varios canales mediante los que los usuarios pueden llegar a una página web, el más común y por lo tanto, el que hay que tener más en cuenta es el de los **buscadores de internet** (Google, Bing, Yahoo!, Ask, etc.). Lo ideal es que cuando alguien busque información relacionada con el contenido de nuestro sitio

web, éste aparezca lo más arriba posible en los resultados del buscador. Aquí es donde entra en juego el **SEO (Search Engine Optimization)** u **optimización para motores de búsqueda**. El término SEO hace referencia al **conjunto de técnicas que se utilizan para mejorar la visibilidad de nuestra página web, su posicionamiento web**. Su objetivo es la construcción de un sitio web fácil de usar e intuitivo tanto para los buscadores, como para los usuarios. El uso de técnicas SEO sólo afecta a los resultados de búsqueda orgánicos, que son los que aparecen a la izquierda (a la derecha suelen encontrarse los patrocinados, que pagan por aparecer).

2.1.1 ¿Cómo funciona un motor de búsqueda?

El primer paso que debemos tomar antes de aplicar técnicas SEO es entender por qué las estamos aplicando y para ello es necesario comprender el funcionamiento de los motores de búsqueda.

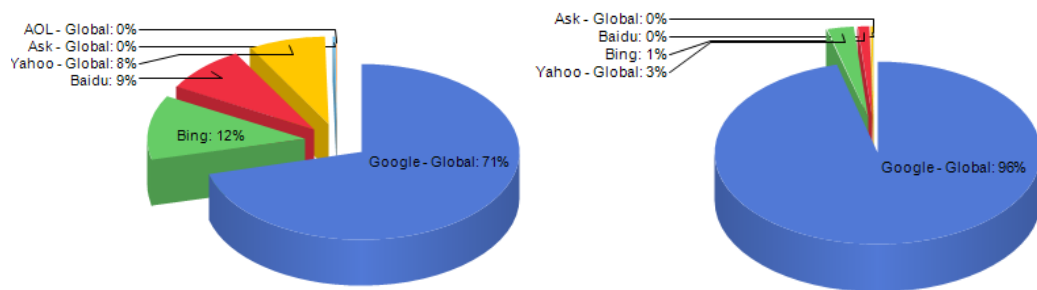


Figura 2: Cuota de mercado de los principales motores de búsqueda accediendo desde un PC (izquierda) y desde un dispositivo móvil (derecha) [13]

Existen varios tipos de motores de búsqueda, pero en este documento nos vamos a centrar en el funcionamiento de los **buscadores jerárquicos**, que es la categoría a la que pertenecen los buscadores más importantes del mercado (ej.: Google, Yahoo!, Baidu, Bing,...).

Cuando realizamos una consulta en un motor de búsqueda, éste produce los resultados de manera casi instantánea. ¿Cómo es posible?:

- **Rastreo**: el buscador jerárquico consta de un programa conocido como **araña o robot**¹, que se encarga del proceso de rastreo e indexado. Este programa recorre la web a través de los billones de enlaces disponibles interconectados entre sí, que referencian páginas web, documentos PDF, imágenes, etc.
- **Indexación**: la araña procesa las páginas rastreadas, descifrando su código entre otras cosas, y guardan determinada información en bases

¹ En inglés se conoce como *crawler* o *spider*.

de datos masivas (**índice**), que utilizarán más adelante para resolver las consultas de los usuarios.

- Publicación de resultados: cuando realizamos una consulta, el motor de búsqueda selecciona de entre las páginas de su índice aquellas que sean **relevantes** y las ordena según su **popularidad**. La relevancia y popularidad de un sitio web son determinadas por el buscador mediante complejos algoritmos que trabajan con cientos de variables. Estas dos características son las que vamos a tratar de influenciar con las técnicas SEO.

2.1.1 Factores a tener en cuenta

Como hemos dicho, hay cientos de factores que afectan al posicionamiento web en mayor o menor medida, por lo que es imposible tener en cuenta a todos. Lo importante es hacerse una idea general, para lo que vamos a utilizar una infografía [ver Figura 3] en la que se aprecian algunos de los factores más importantes.

En dicha infografía se utiliza una división que clasifica los factores en dos grupos:

- Factores que dependen exclusivamente del creador de la página web:
 1. **Contenido**: calidad y actualidad de las publicaciones o servicios ofrecidos, uso correcto de palabras clave, etc.
 2. **Arquitectura**: facilidad del rastreo realizado por las arañas de los buscadores, uso de diferentes versiones de una misma página (duplicados), adaptación de la página para dispositivos móviles, URLs descriptivas, velocidad, etc.
 3. **Código HTML**: contenido de las etiquetas <TITLE> y <META>, datos estructurados, utilización correcta de las jerarquías, etc.
- Factores que dependen de los usuarios de la página web y de otros sitios web:
 1. **Confianza**: autoridad dentro del ámbito de competencia, implicación y participación de los usuarios (comentarios, tiempo de permanencia en el sitio, etc.), trayectoria de la página a lo largo de su existencia, etc.
 2. **Enlaces**: enlaces de webs respetadas, textos ancla de los hipervínculos, etc.
 3. **Personal**: residencia (país, localidad, etc), historial de los usuarios, etc.
 4. **Reputación**: referencias desde redes sociales y presencia de la página en redes sociales.

En las siguientes secciones, estudiaremos algunos de estos factores más en profundidad, en particular los pertenecientes al primer grupo (ya que son en los que más podemos influir), así como las técnicas SEO que nos permitirán mejorar el posicionamiento de nuestro sitio web.

Por último, en la infografía se puede observar que algunos factores se han representado en rojo. Éstos son los que influyen de manera negativa en el posicionamiento web, en su mayoría relacionados con el uso de **técnicas SEO penalizadas**, también denominadas **técnicas *black hat*** o ***spam***, que veremos más adelante.

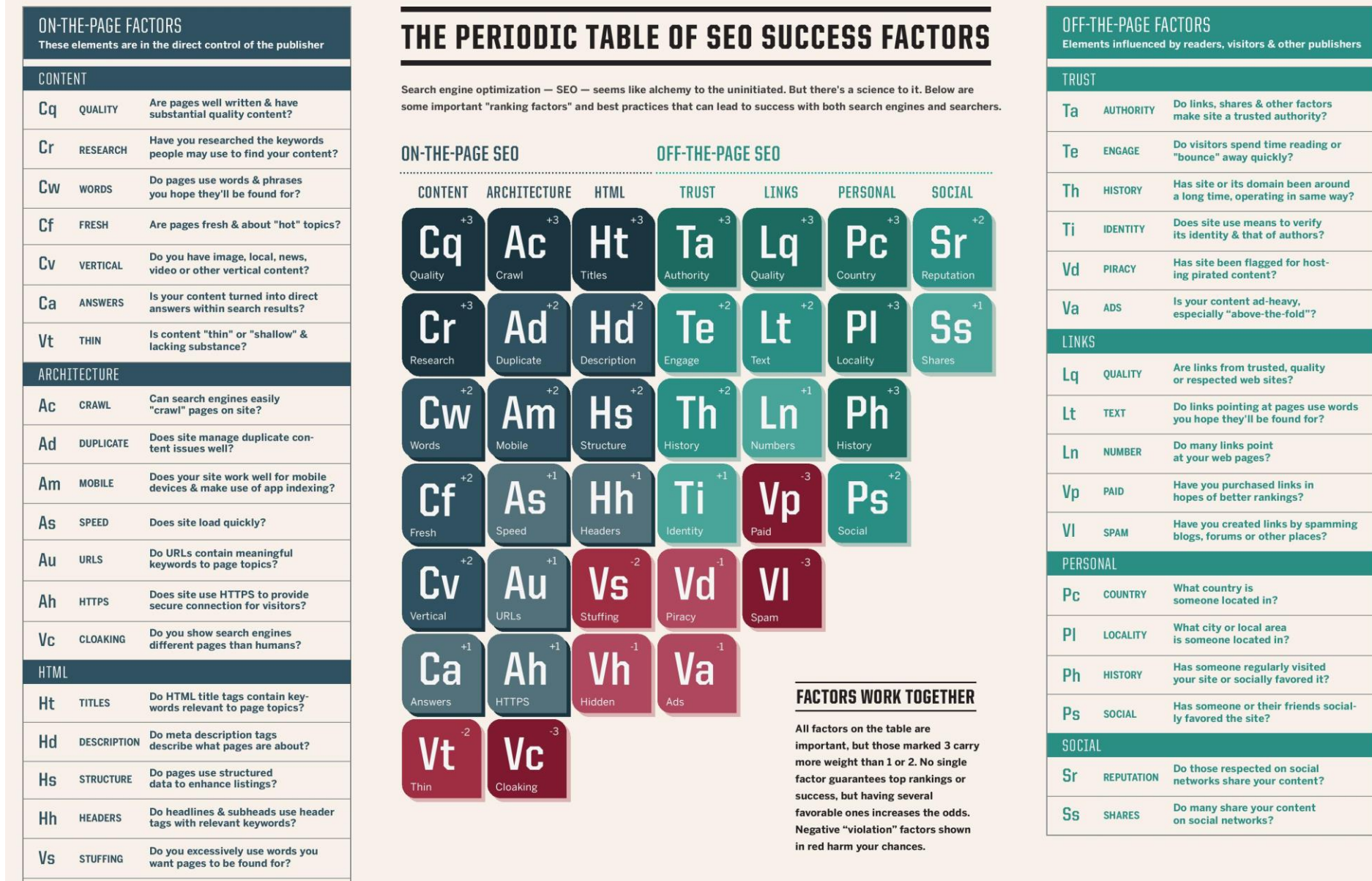


Figura 3: Infografía con algunos de los factores que influyen el posicionamiento web [18]

2.2 CONTENIDO

2.2.1 Calidad y actualidad

A la hora de desarrollar el contenido de nuestro sitio web, es fundamental tener en cuenta los siguientes factores:

1. **Calidad:** **la información o los servicios que ofrece nuestra página web deben ser de calidad, es decir, ofrecer algo original, diferente y útil.** No sirve de nada aplicar técnicas SEO si no ofrecemos a los usuarios contenidos de calidad, que les haga quedarse más tiempo en la página y volver a visitarla.

Los motores de búsqueda no son capaces de apreciar la calidad del contenido como lo hace una persona, pero han desarrollado técnicas que se acercan.

Uno de los factores que tienen en cuenta es el **tiempo de permanencia en el sitio web**. Cuando un usuario realiza una búsqueda, el buscador monitoriza su actuación: cuánto tiempo se queda el usuario en la página elegida, si vuelve a la página de búsqueda y selecciona otro resultado, etc.

En 2011, Google introdujo la **actualización Panda**, que cambió radicalmente la manera en que sus algoritmos calculaban la calidad del contenido de una web. Primero utilizaron personas para evaluar manualmente la calidad de miles de páginas y con los datos obtenidos, introdujeron máquinas que imitaban el proceso humano.

2. **Actualidad:** es conveniente actualizar la información, **ofrecer contenido nuevo**. En relación con este punto, Google utiliza un sistema denominado **QDF** (*Query Deserved Freshness*). Cuando el motor de búsqueda detecta un aumento de popularidad de una determinada consulta, pone en marcha el mecanismo QDF, que se encarga de buscar información nueva que tenga relación con la consulta. Las páginas que encuentre que contengan información actualizada aparecerán mejor posicionadas durante el tiempo en que la consulta sea popular. Herramientas como **Google Trends** [ver Figura 4], que proporciona una lista de los temas más populares en un momento determinado, nos ayudan a mantenernos al día.

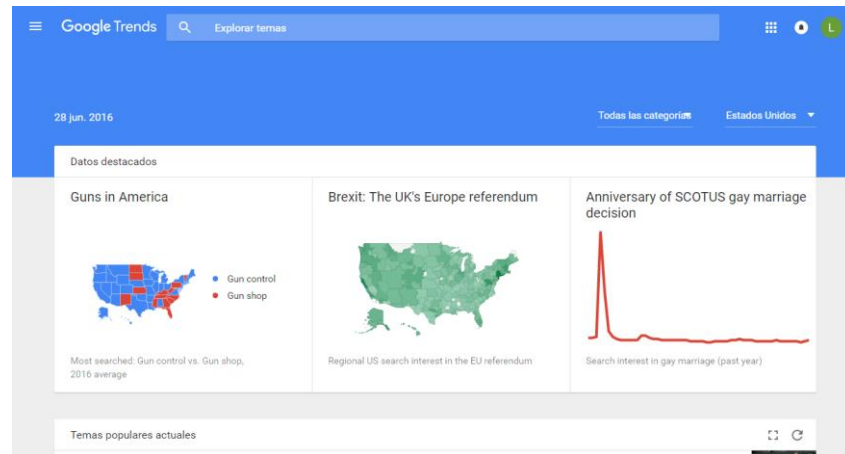


Figura 4: Página Google Trends

A pesar de ser dos factores muy importantes, **no existen técnicas SEO específicas para mejorar la calidad y la actualidad del contenido**. Dedicando tiempo, cada vez comprenderemos mejor qué es interesante para los usuarios de nuestra página y podremos modificar y mejorar su contenido en consecuencia

2.2.2 Palabras clave

El uso de las **palabras clave** o **keywords** es la base del proceso de búsqueda. **Si el motor de búsqueda es el canal mediante el que los usuarios se comunican con nuestra página web, las palabras clave son el lenguaje que utilizan**. Hemos visto cómo funcionan los motores de búsqueda y en particular, el proceso de indexación, en el que las páginas rastreadas se almacenan en bases de datos. Para proporcionar resultados más rápidos, las páginas se almacenan en diferentes **bases de datos basadas en palabras clave**.

Los motores de búsqueda inspeccionan las páginas web en busca de palabras clave para determinar la relevancia de dicha página con respecto a una consulta determinada. El primer paso, por lo tanto es **averiguar qué palabras o frases utilizarán nuestros potenciales usuarios**. Existen herramientas SEO que proporcionan información que nos puede resultar de ayuda. Por ejemplo, "www.wordtracker.com", permite introducir una palabra clave y nos proporciona aquellas relacionadas con el mayor volumen de búsquedas.

También es importante tener en cuenta que **los usuarios no siempre son expertos** en los temas tratados en nuestra página web, **lo que se refleja en las palabras clave que utilizan**. Por ejemplo, el desarrollador de una página dedicada al posicionamiento SEO, debe considerar que mientras habrá usuarios que conozcan el término SEO y lo utilicen en sus búsquedas, habrá otros muchos que no y que busquen simplemente cómo mejorar la posición de su página web.

¿Cómo optimizar el uso las palabras clave?

No existe una fórmula que nos indique el número exacto de veces que las palabras clave deben aparecer en nuestra página, pero conviene utilizarlas:

1. En la etiqueta <TITLE> al menos una vez.
2. Una vez en la parte superior de la página.
3. Por lo menos dos o tres veces (incluyendo variaciones) dentro del cuerpo del documento, es decir, dentro de la etiqueta <BODY>.
4. Por lo menos una vez en el atributo ALT de una imagen, útil para búsquedas por imagen.
5. Una vez en la URL.
6. Por lo menos una vez en la etiqueta <META>.

En la Figura 5 se puede ver un ejemplo de una página web que pretende atraer a usuarios buscando “Running Shoes”:



Figura 5: Ejemplo de uso de palabras clave [10]

Técnicas penalizadas

1. **Keyword Stuffing:** se nos podría ocurrir que dado que el uso de palabras clave es importante, cuántas más utilicemos, mejor. Sin embargo, **utilizar las palabras clave de manera indiscriminada** en nuestra página web está penalizado. **¿Cuántas veces son muchas veces?** No existe una respuesta estándar para esta pregunta, pero basta con utilizar el sentido común. Utilizando el ejemplo de la Figura

5, utilizar las palabras clave “Running Shoes” más veces de las que se ha hecho sería demasiado, ya que el texto es muy corto. El número de veces que utilizamos las palabras clave debe estar acorde a la cantidad de texto total que tenemos.

Otra variante de esta técnica, que también está penalizada, es **utilizar palabras clave que no tienen relación con el contenido de la página**. Por ejemplo, utilizar “Sagrada Familia” repetidas veces en una página cuya finalidad es la venta de zapatos.

2. *Texto oculto*: esta técnica consiste en **incluir texto oculto para los usuarios, pero visible para las arañas de los motores de búsqueda**. En este texto oculto se repiten las palabras clave una y otra vez con la intención de engañar a los buscadores. Normalmente, se pone el texto del mismo color o de un color parecido al del fondo, para que los lectores no lo vean. Para los buscadores, los colores son valores hexadecimales, de manera que pueden comprobar si se está utilizando esta técnica.

2.3 HTML

La información que muestran los buscadores [ver Figura 6] cuando seleccionan nuestra página web es importante tanto para el posicionamiento web, como para los usuarios, y depende de diferentes elementos del código HTML utilizado.



Figura 6: Apariencia de la página web de la ETSIT como respuesta a una consulta realizada en Google

2.3.1 Etiqueta <TITLE>

La etiqueta <TITLE> contiene el título de la página [ver Figura 7] y es el principal indicador del código HTML utilizado por los buscadores para entender de qué trata la página web. Por lo tanto, debe ser conciso y sobre todo, descriptivo.

Escuela Técnica Superior de Ingenieros de Telecomunicación ...

Figura 7: Contenido de la etiqueta <TITLE> mostrado por Google (ejemplo Figura 6)

¿Cómo optimizar la etiqueta <TITLE>?

1. **Palabras clave:** como ya vimos en la sección dedicada a las palabras clave, éstas deben aparecer en la etiqueta <TITLE> y **cuanto más próximas al comienzo del título, más afectarán al posicionamiento web.**
2. **Longitud:** los motores de búsqueda sólo muestran los primeros **50-60 caracteres del título** (aproximadamente), por lo que es conveniente ajustarse a este límite. Aun así, en ciertos casos puede ser recomendable utilizar más caracteres.
Por ejemplo, en la Figura 7 se puede observar que el título es más largo de lo que se muestra. El título completo es “Escuela Técnica Superior de Ingenieros de Telecomunicación – Universidad de Valladolid”, lo que hace al sitio web relevante tanto para búsquedas de la ETSIT, como para búsquedas de la Universidad de Valladolid (aunque en este último caso en menor medida, ya que se encuentra al final de la etiqueta).
3. **Branding:** los títulos de cada página deben ser únicos y representativos del contenido individual de cada una de ellas. Sin embargo, también puede ser útil mantener la palabra clave más importante en todos ellos, por ejemplo la marca o nombre de nuestra página web.
En la Figura 8, podemos ver un ejemplo de dos páginas cuyo título es prácticamente igual y cuya principal diferencia es la marca.



Figura 8: Ejemplo de *Branding*

2.3.2 Metadatos (etiqueta <META>)

Mediante la etiqueta <META> podemos proporcionar información adicional a los buscadores acerca de nuestra página. A continuación, veremos dos de los más importantes y cómo usarlos correctamente.

2.3.2.1 Meta Description²

Los datos incluidos en una etiqueta <META> de este tipo comprenden la **descripción del contenido de una página** y normalmente, los buscadores la muestran debajo de la URL [ver Figura 9].

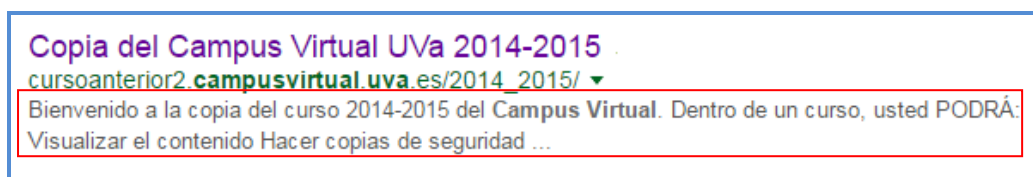


Figura 9: Contenido de la etiqueta <META> mostrado por Google (en rojo)

La utilización de esta etiqueta **no influye directamente en el posicionamiento web**, es decir, no es utilizada por los motores de búsqueda como factor para calcular la relevancia o popularidad de nuestro sitio web. Sin embargo, como ya dijimos en la introducción, las técnicas SEO están dirigidas a optimizar una web tanto para los buscadores, como para los usuarios y un correcto uso de esta etiqueta atraerá más usuarios.

Los buscadores muestran un **máximo de aproximadamente 160 caracteres**, por lo que conviene ceñirse a este límite. Dado que en este caso el texto sólo está dirigido a los usuarios, no sirve de nada una descripción más larga.

2.3.2.2 Meta Robots

Esta etiqueta **se utiliza para controlar ciertos aspectos del rastreo de las arañas a nivel de página**. Por defecto, al rastrear nuestra página web, las arañas indexan todas las páginas accesibles y siguen todos los enlaces que encuentran. Éste comportamiento se puede modificar incluyendo la etiqueta <META NAME="robots" CONTENT="Valores">, donde valores puede constar (entre otros) de los siguientes términos:

1. Index, Follow: ésta es la **combinación por defecto**, por lo que no hace falta utilizar la etiqueta si queremos que éste sea el comportamiento de

² Código HTML: <META NAME="description" CONTENT="Descripción de la página.">

la araña. INDEX indica que queremos que la página sea indexada y FOLLOW, que los enlaces sean seguidos.

2. Index, Nofollow: si queremos que la página sea indexada, pero que los enlaces no sean seguidos. Esta combinación puede ser **útil en páginas que contienen contenido generado por los usuarios**, ya que pueden añadir links no fiables, que perjudiquen tu posicionamiento web.
3. Noindex, Follow: si queremos que la página no sea indexada, pero sí que se sigan los enlaces. Útil para la página relativa a los derechos de autor o **para evitar contenido duplicado**, que como veremos más adelante, está penalizado.
4. Noindex, Nofollow: si queremos que la página no sea indexada ni los enlaces, seguidos.

2.4 ARQUITECTURA

Estructurar nuestra página web correctamente es esencial y para conseguirlo, hay que prestar especial atención a los procesos de rastreo e indexado de los motores de búsqueda. En los siguientes apartados veremos cómo evitar errores comunes que entorpecen la labor de las arañas, así como las prácticas más adecuadas para facilitar su cometido.

2.4.1 Dominio

El primer paso que hay que tomar a la hora de crear una página web es adquirir un dominio. Hace unos años, la mayoría de SEOs³ compraban los denominados **Exact Match Domains (EMDs)**, que son aquellos que coinciden con la palabra/frase clave objetivo de la página. Por ejemplo, si la página web pretendía atraer a usuarios que quisieran comprar libros, comprarían el dominio (si estuviera disponible) “comprarlibros.com”. A partir de 2010 este tipo de enfoque ha ido decayendo, debido a que los buscadores cada vez dan menos importancia a que el dominio coincida exactamente con las palabras clave.

³ El término SEO también se emplea para designar a los profesionales que se dedican al asesoramiento de posicionamiento web.

¿Qué hay que tener en cuenta al elegir un dominio?

1. Facilidad para memorizar: el nombre de dominio debe ser fácil de recordar, por lo que su longitud también debe ser reducida (en torno a los **15 caracteres** como máximo).
2. Uso de caracteres especiales: si utilizamos más de una palabra en el nombre de dominio, **puede ser necesario incluir un guion** para mejorar su legibilidad (ej.: comprar-libros.com), **pero en general su uso está desaconsejado**. Algunos buscadores lo interpretan como un indicador de *spam*, lo que puede afectar negativamente al posicionamiento de nuestra web.
3. Dominio raíz⁴: normalmente, a no ser de que nuestros usuarios potenciales estén limitados a un país, lo mejor es elegir el dominio raíz .com, ya que es el más usado (en torno al 75% de las páginas web tienen esta extensión) y por lo tanto, el que suelen utilizar los usuarios cuando buscan una página web. Además, el uso de dominios raíz poco conocidos como .info, .cc o .ws también es un indicador de *spam*.

2.4.2 Contenido indexable

Es importante tener siempre presente, que **los motores de búsqueda presentan ciertas limitaciones a la hora de comprender el contenido de las páginas web**. Entender estas limitaciones es clave para construir un sitio web de fácil navegación tanto para los buscadores como para los usuarios.

La mayoría del contenido de nuestra web debe estar en **formato texto en HTML**, lo que nos asegura que es visible para los motores de búsqueda. Los elementos que dan más problemas son imágenes, vídeos, contenido Flash, applets Java, etc. En la Figura 10 se puede observar una página web desarrollada completamente en Flash. Los motores de búsqueda no pueden ver su contenido, por lo que será muy difícil que esta página aparezca entre los resultados de búsqueda.

⁴ En inglés, *Top Level Domain* (TLD).

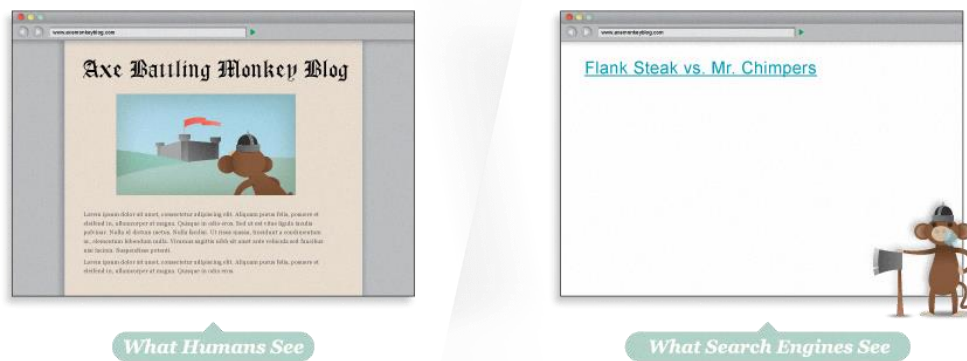


Figura 10: Comparación: página en Flash vista por una persona (izq.) y por un buscador (dcha.) [10]

¿Cómo optimizar el uso del contenido problemático?

Que haya contenido difícil de interpretar por los motores de búsqueda, no quiere decir que no podamos usarlo en absoluto. Al contrario, contenido visual (imágenes, vídeos, etc.) o interactivo (búsquedas internas, formularios, etc.) resulta atractivo y en muchos casos necesario para los usuarios. Los buscadores no van a ser capaces de entender este contenido igual que los humanos, pero podemos ayudarles:

1. **Imágenes:** conviene evitar los nombres de archivo tipo "IMG2345678.png" que no aportan ninguna información. Los **nombres de archivo deben ser descriptivos** (ej.: zapatillas-blancas-balonmano.png).

También es importante **utilizar siempre el atributo ALT de HTML**⁵ para proporcionar una mejor descripción de la imagen. No sólo es útil para los motores de búsqueda, sino también para los usuarios que no puedan ver la imagen (por ejemplo, debido a un ancho de banda reducido o a una discapacidad visual).

Parece superfluo mencionarlo, pero el **contexto** es muy importante para los buscadores. Si incluimos la imagen de las zapatillas de balonmano junto a un texto que trata de las últimas actualizaciones en materia SEO, estaremos confundiendo al buscador.

2. **Vídeos/Audios:** si queremos que el contenido de los audios o vídeos sea indexado, debemos proporcionar una **transcripción** de los mismos.

⁵ Código HTML: ``

3. Contenido Flash/Java: conviene no abusar de este tipo de contenido. Al igual que sucedía con las imágenes, un **contexto** textual adecuado, ayuda a los buscadores a entender este tipo de contenido.

Herramientas SEO

Existen varias herramientas que podemos utilizar para comprobar cómo ven los motores de búsqueda el contenido de nuestra página web. En la Figura 11 se ha utilizado la **caché de Google**, por ejemplo. La página “**SEO-browser.com**” también ofrece este servicio.



Figura 11: Comparación: página en vista por una persona (izq.) y por un buscador (dcha.) utilizando la caché de Google [10]

2.4.3 Estructura de enlaces

Para que los buscadores indexen el contenido de nuestras páginas web, primero tienen que haber sido capaces de rastrear las páginas en las que se encuentran. Es necesario tener una buena **estructura de enlaces**, que conecte todas nuestras páginas de manera eficiente, facilitando así el rastreo de las arañas. Si no lo hacemos correctamente, **puede que una o varias páginas de nuestra web sean inalcanzables para las arañas debido a enlaces rotos u ocultos** [ver Figura 12].



Figura 12: Estructura de enlaces en la que las páginas C y D son inalcanzables [10]

¿Por qué algunas de mis páginas son inalcanzables?

Algunos de los errores más comunes que tienen como consecuencia el que algunas páginas sean inalcanzables, son los siguientes:

1. Formularios y búsquedas internas: si tenemos **páginas únicamente accesibles tras rellenar y enviar un formulario**, lo más probable es que los motores de búsqueda no lleguen a encontrarlas. Otro error común es pensar que por tener una **herramienta de búsqueda interna**, las arañas la van a utilizar para encontrar todas las páginas de nuestro sitio web igual que lo haría un usuario. Aunque los buscadores son capaces tanto de realizar una búsqueda como de rellenar un formulario y de vez en cuando lo lleven a cabo, este comportamiento es la excepción y no la regla.
2. Enlaces en Java, Flash, etc.: como hemos visto en la sección anterior, el **contenido de tipo Java, Flash, etc. es problemático** para los buscadores, por lo que si tenemos enlaces embebidos, las páginas a las que apuntan permanecerán ocultas para los buscadores.
3. Número elevado de enlaces: las arañas **no siguen un número infinito de enlaces** en una determinada página. Si en una de nuestras páginas hay cientos de links, puede que algunos de ellos no sean seguidos.

El primer paso para tener una buena estructura de enlaces que facilite el rastreo de nuestro sitio web es evitar las prácticas mencionadas, pero no es lo único que podemos hacer.

Los buscadores asignan a cada sitio web un “presupuesto de rastreo”, es decir, un tiempo máximo de rastreo, por lo que es importante que lo hagan de manera eficiente y en la medida de lo posible, de acuerdo con nuestros intereses. Existen varias herramientas que nos permiten conseguir este objetivo. Entre ellas se encuentran los “Meta Robots”, que ya hemos tratado en la sección dedicada a HTML y que aparecen definidos en el **Estándar de Exclusión de Robots**⁶. A continuación estudiaremos otras dos herramientas que también fueron definidas por el citado estándar, así como el uso de archivos *Sitemap*.

2.4.3.1 Robots.txt

Robots.txt⁷ es un archivo de texto, mediante el cual podemos dar ciertas instrucciones a los motores de búsqueda sobre cómo rastrear nuestro sitio web

⁶ En inglés, *Robots Exclusion Protocol* (REP). Grupo de estándares web creado en 1994 precisamente para regular los procesos de rastreo e indexado de los buscadores.

⁷ El nombre del archivo debe escribirse siempre en minúsculas: robots.txt.

(definido por REP en 1994/1997). Este archivo debe encontrarse en el directorio raíz de la página web (ej.: <https://ejemplo.com/robots.txt>) y admite las siguientes etiquetas y caracteres:

Tabla 1: Parámetros admitidos en el archivo robots.txt

Etiqueta / Caracter	Función	Ejemplo de uso
#	Precede a un comentario.	#Ejemplo de archivo robots.txt
User-agent:	Especifica el robot/s a los que se dirige la instrucción.	User-agent: Googlebot
Allow:	Indica que el directorio puede ser rastreado.	Allow: /directorio-ejemplo/
Disallow:	Indica que el directorio no debe ser rastreado.	Disallow: /directorio-ejemplo/
Sitemap:	Indica dónde se encuentra el archivo <i>Sitemap</i> [ver Sección __].	Sitemap: ejemplo.com/archivo-sitemap.xml
*	Comodín. Se puede usar en las URL o en User-agent.	User-agent: * (dirigido a todos los buscadores)
\$	Indica el final de una URL.	Disallow: /*.pdf\$ (indica que todas las URL acabadas en .pdf no deben ser rastreadas)

En la Figura 12 se puede ver un ejemplo de archivo robots.txt, en el que se indica a los robots de Google (Googlebot) y Bing (bingbot)⁸ que no rastreen el directorio /temp/ ni los archivos PDF.

```
# Archivo robots.txt para la página https://ejemplo.com

User-agent: Googlebot
User-agent: bingbot
Disallow: /temp/
Disallow: /*.pdf$
```

Figura 12: Ejemplo 1 de archivo robots.txt

En el segundo ejemplo [ver Figura 13], indicamos a la araña de Google que puede rastrear todas las páginas y prohibimos totalmente el acceso al resto de robots. Hay que tener en cuenta, sin embargo, que **los protocolos especificados en REP se basan en la colaboración de los robots**. Esto significa, que robots con intenciones maliciosas pueden no seguir las instrucciones especificadas. Por esta

⁸ Un listado detallado de nombres de robots se puede encontrar en la página web <http://www.useragentstring.com/pages/useragentstring.php>.

razón, conviene evitar el uso de este archivo para bloquear el acceso a páginas que contengan información sensible (páginas de registro, por ejemplo).

```
# Archivo robots.txt para la página https://ejemplo.com

User-agent: Googlebot
Disallow:

User-agent: *
Disallow: /
```

Figura 13: Ejemplo 2 de archivo robots.txt

¿Cuál es la diferencia entre el `DISALLOW` de robots.txt y el `NOINDEX` de los “Meta Robots”?

El `DISALLOW` utilizado en robots.txt, indica que una página no debe ser rastreada, pero no evita que ésta sea indexada y que, si el buscador la considera relevante, aparezca entre los resultados de una búsqueda. Eso sí, con mucha menos información [ver Figura 14].

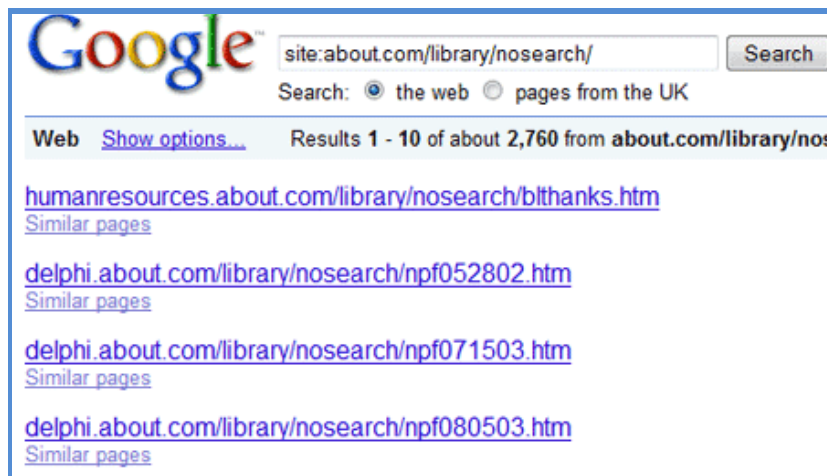


Figura 14: Resultados de la búsqueda en Google de una URL bloqueada mediante el uso de `DISALLOW` [11]

El valor `NOINDEX` en los “Meta Robots”, por el contrario, permite que la página sea rastreada, pero evita que sea indexada y por lo tanto, evita también que aparezca entre los resultados de una consulta.

En general, para bloquear el acceso de los robots a una página se recomienda el uso de los “Meta Robots” en lugar del archivo robots.txt.

Herramientas SEO

Hay que tener cuidado al crear el archivo robots.txt, ya que podemos bloquear páginas de manera inintencionada. Existen varias herramientas que nos ayudan a evitar errores:

1. Herramientas de comprobación: permiten comprobar que hemos redactado el archivo correctamente, como el **Probador de robots.txt de Google**⁹. Esta herramienta nos advierte de errores en la lógica y sintaxis del archivo y nos permite comprobar si una determinada URL está bloqueada o no. Otra variante de esta herramienta, también proporcionada por Google, es el **Informe Recursos bloqueados**¹⁰, que muestran una lista de los recursos bloqueados en las reglas del archivo robots.txt de nuestra página web.
2. Herramientas de generación: este tipo de herramientas nos ayuda a **generar el archivo desde cero**. En la Figura 15 se puede ver una captura de pantalla de “<http://tools.seobook.com/robots-txt/generator/>”, que ofrece esta funcionalidad. Aunque nos permite generar las reglas como queramos, también advierte de errores en la sintaxis del archivo. Como por ejemplo, que una de las reglas esté duplicada.

⁹ Disponible en <https://support.google.com/webmasters/answer/6062598?hl=es>

¹⁰ Disponible en https://support.google.com/webmasters/answer/6153277?hl=es&ref_topic=6065797

Action	Robot	Files or directories
<input checked="" type="radio"/> Disallow <input type="radio"/> Allow	msnbot	/ <input type="button" value="Add"/>
This is duplicate rule		
<input checked="" type="radio"/> Disallow <input type="radio"/> Allow	Googlebot-Image	/*.png\$ <input type="button" value="Copy"/> <input type="button" value="Delete"/>
<input checked="" type="radio"/> Disallow <input type="radio"/> Allow	msnbot	/ <input type="button" value="Copy"/> <input type="button" value="Delete"/>
<input checked="" type="radio"/> Disallow <input type="radio"/> Allow	Slurp	temp <input type="button" value="Copy"/> <input type="button" value="Delete"/>
You should start url with /		

Figura 15: Interfaz de una herramienta de generación de archivos robots.txt

2.4.3.2 Atributo REL="NOFOLLOW"

Otra de las herramientas que nos permite optimizar el rastreo de nuestra web es la **utilización del atributo de enlaces REL con el valor NOFOLLOW**¹¹ (definido por REP en 2005). Cuando se lo añadimos a un enlace, la mayoría de los buscadores lo ignorarán y no lo seguirán. Google los trata como si fueran texto HTML, ni los sigue ni los utiliza para el posicionamiento web. Bing tampoco los utiliza para el posicionamiento, pero sí que los sigue con el objetivo de descubrir nuevas páginas.

Este atributo **nos permite aumentar la granularidad con la que controlamos los enlaces** de una página. Mientras que el valor NOFOLLOW en los "Meta Robots" afectaba a todos los enlaces de una página, en el atributo REL afecta a un único enlace.

¿En qué enlaces debo utilizar el atributo REL="NOFOLLOW"?

1. Sitios no fiables: los motores de búsqueda dan mucha importancia a los enlaces entre páginas web, ya que a partir de ellos determinan

¹¹ Código HTML: `¡Regístrate aquí!`

factores como la popularidad o la confianza de las páginas. Por esta razón, **cuando no estemos seguros de la fiabilidad de un determinado sitio web**, es recomendable utilizar `NOFOLLOW`.

2. Registro: las arañas no tienen ninguna razón para convertirse en miembros de nuestro sitio web, por lo que **no tiene sentido que sigan los enlaces en los que invitamos a los usuarios a registrarse**.
3. Enlaces de pago: como veremos más adelante [Sección 2.5], **está penalizado pagar por enlaces para mejorar nuestra popularidad**. Si queremos tener enlaces de publicidad del tipo PPC (Pago Por Click), pero evitar que nos perjudiquen, basta con añadir el atributo `REL` con el valor `NOFOLLOW` a este tipo de enlaces.

2.4.3.3 Archivo Sitemap

Los **sitemaps**, como la palabra indica, son una especie de **mapa de nuestro sitio web**, que proporciona información a las arañas para que rastreen nuestro sitio web de manera más inteligente. En este archivo podemos **enumerar las URL** de nuestra página que queremos que sean indexadas, así como información relativa a las mismas (importancia, frecuencia de actualización, etc.), **y los recursos que contienen** (imágenes, videos, etc.).

La creación de este archivo no nos garantiza que todas las URL incluidas vayan a ser indexadas, pero los motores de búsqueda siempre recomiendan el uso de *sitemaps*, porque les **ayuda a hacerse una idea de la estructura del sitio web más rápidamente**. Además, también **se evita el problema de las páginas descolgadas**. Aunque se nos haya pasado algún enlace oculto para los buscadores, al incluir todas las URL en el archivo *Sitemap*, los robots serán capaces de encontrarlo.

Los archivos Sitemap pueden ser de **tipo XML, RSS o texto**. El de tipo texto [ver Figura 16] es el más sencillo y consiste en incluir todas las URL (una por línea) hasta un máximo de 50.000, utilizando codificación UTF-8. No aporta tanta información como los otros dos tipos.

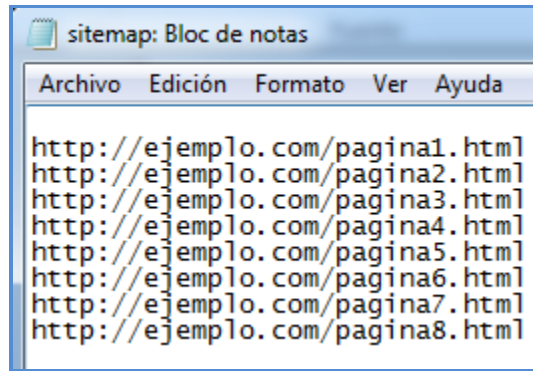


Figura 16: Ejemplo de archivo sitemap.txt

En este documento estudiaremos en más profundidad los *sitemaps* de tipo XML, que es el formato recomendado.

Sitemap de tipo XML

A continuación, se presenta una tabla con las etiquetas que pueden o deben formar parte de nuestro archivo *Sitemap* de acuerdo con el protocolo estándar¹²:

Tabla 2: Etiquetas XML admitidas en el protocolo estándar de Sitemap

Etiqueta	Función
<urlset> (obligatorio)	Encapsula el archivo. Hace referencia al protocolo estándar.
<url> (obligatorio)	Etiqueta principal de una URL. Encapsula el resto de etiquetas que aportan información sobre dicha URL.
<loc> (obligatorio)	URL de la página. Debe comenzar con un protocolo (ej.: http) y finalizar con /. Longitud máxima: 2048 caracteres.
<lastmod> (opcional)	Indica la fecha de última modificación del archivo, que debe encontrarse en formato Fecha y hora W3C ¹³ .
<changefreq> (opcional)	Indica la frecuencia aproximada de actualización de la página. Los valores permitidos son: <ul style="list-style-type: none"> - always (para páginas que cambian en cada nuevo acceso) - hourly - daily - weekly - monthly - yearly - never (para páginas archivadas) Los buscadores no tienen por qué rastrear las páginas exactamente con la frecuencia indicada, pero les proporciona información general.

¹² El protocolo estándar Sitemap (definido en <http://www.sitemaps.org/>) no incluye las extensiones XML para recursos de videos, imágenes, móviles y noticias.

¹³ Estándar definido en <https://www.w3.org/TR/NOTE-datetime>

<priority> (opcional)	Indica la prioridad de una dirección URL en relación al resto de URLs del sitio web. Los valores permitidos se encuentran entre 0.0 y 1.0, siendo 0.5 la prioridad por defecto. Los motores de búsqueda tienen la prioridad en cuenta a la hora de seleccionar una u otra URL de un mismo sitio web, pero no influye en el posicionamiento web de la página. Por esta razón, asignar a todas nuestras páginas prioridad 1.0 no es una manera inteligente de usar este atributo, ya que no estamos dando buena información al buscador.
---------------------------------------	---

Para tenerlo más claro, a continuación podemos ver un ejemplo de archivo Sitemap en el que se utilizan todas las etiquetas que aparecen en la Tabla 2 a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">

  <url>
    <loc>http://ejemplo.com/pagina1.html</loc>
    <lastmod>2016-07-01T18:00:15+00:00</lastmod>
    <changefreq>daily</changefreq>
    <priority>1.0</priority>
  </url>

  <url>
    <loc>http://ejemplo.com/pagina2.html</loc>
    <changefreq>weekly</changefreq>
    <priority>0.9</priority>
  </url>

  <url>
    <loc>http://ejemplo.com/pagina3.html</loc>
    <lastmod>2016-06-23</lastmod>
    <changefreq>weekly</changefreq>
  </url>
```

```

<url>
  <loc>http://ejemplo.com/pagina4.html</loc>
  <priority>0.3</priority>
</url>

<url>
  <loc>http://ejemplo.com/pagina5.html</loc>
  <lastmod>2004-11-15</lastmod>
  <changefreq>never</changefreq>
</url>

</urlset>

```

Una vez redactado, debemos guardar el archivo. La **ubicación** del mismo no es trivial, ya que **determina el grupo de URLs que se pueden incluir** en el mismo. Por ejemplo, tomando el ejemplo que acabamos de ver, el archivo *Sitemap* debería estar almacenado en “http://ejemplo.com/sitemap.xml”. Esta ubicación le da permiso para incluir todas las URL que empiecen por “http://ejemplo.com/”, pero no las que empiecen, por ejemplo, por “https://ejemplo.com/”.

Otro aspecto a tener en cuenta es el tamaño. Un **archivo Sitemap no puede contener más de 50.000 URLs ni ocupar más de 10MB**, por lo que si nuestro archivo supera estos valores hay que dividirlo en varios *sitemaps*. Conviene entonces, utilizar un **índice de Sitemap** que contenga todos los *sitemaps*. En la Tabla 3 se pueden ver las etiquetas que se permite incluir en el índice y a continuación, un ejemplo.

Tabla 3: Etiquetas XML admitidas en el archivo índice de *Sitemap*

Etiqueta	Función
<sitemapindex> (obligatorio)	Encapsula el archivo.
<sitemap> (obligatorio)	Etiqueta principal de un archivo <i>Sitemap</i> . Encapsula el resto de etiquetas que aportan información sobre dicho <i>Sitemap</i> .
<loc> (obligatorio)	URL del Sitemap.

<code><lastmod></code> (opcional)	Indica la fecha de última modificación del archivo, que debe encontrarse en formato Fecha y hora W3C ¹⁴ .
--	--

```
<?xml version="1.0" encoding="UTF-8"?>
<sitemapindex xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">

  <sitemap>
    <loc>http://ejemplo.com/sitemap1.xml.</loc>
    <lastmod>2016-01-01T18:23:17+00:00</lastmod>
  </sitemap>

  <sitemap>
    <loc>http://ejemplo.com/sitemap2.xml.</loc>
    <lastmod>2016-06-01</lastmod>
  </sitemap>

</sitemapindex>
```

Una vez que hemos creado el sitemap (o varios con su índice asociado), el último paso consiste en **informar a los motores de búsqueda** de dónde se encuentra. Esto se puede hacer mediante el **archivo robots.txt** (opción recomendada), como hemos visto en la Sección 2.4.3.1, pero no es la única opción. También se puede utilizar la **interfaz de envío de los diferentes motores de búsqueda** o enviando una **solicitud HTTP**.

Herramientas SEO

Al igual que en el caso de los archivos robots.txt, las herramientas SEO disponibles se dividen en herramientas de comprobación y de generación:

1. Herramientas de comprobación: permiten comprobar que hemos redactado el archivo *Sitemap* correctamente. Google tiene para ello la **herramienta de pruebas de sitemaps de Search Console**¹⁵.

¹⁴ Estándar definido en <https://www.w3.org/TR/NOTE-datetime>

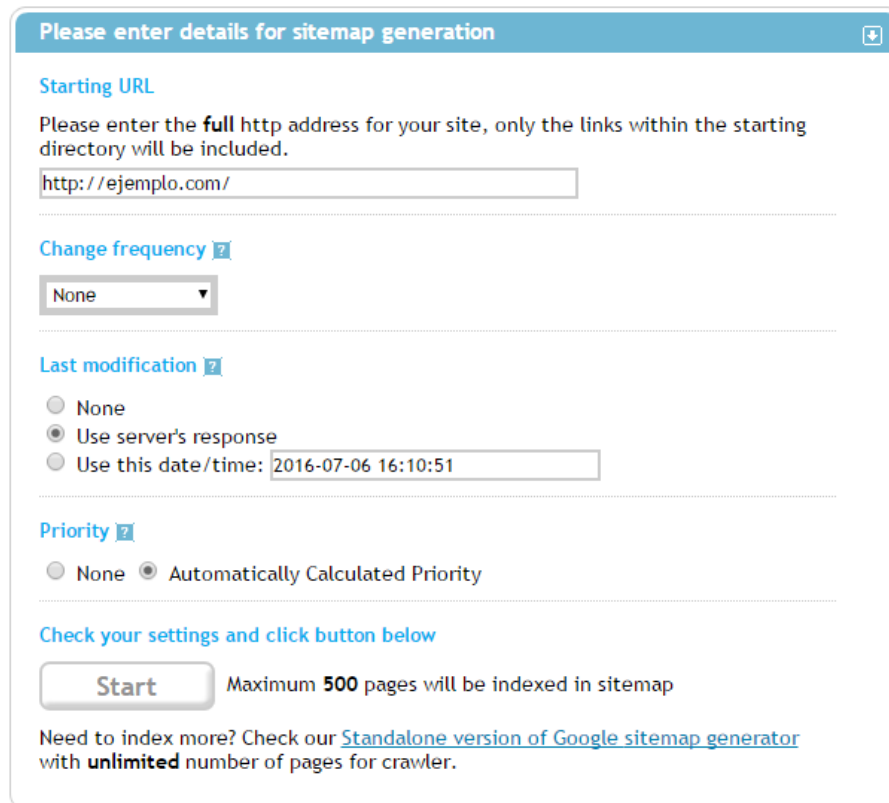
¹⁵ Disponible en

https://support.google.com/webmasters/answer/183668?hl=es&ref_topic=4581190#sitemapformat

2. Herramientas de generación: este tipo de herramientas nos ayudan a **generar el Sitemap desde cero**. En la Figura 17 se puede ver una captura de pantalla del generador de *sitemaps* la página **XML-Sitemaps.com**. Una vez rellenado el cuestionario inicial, el generador rastrea nuestra página web y genera el archivo XML Sitemap. Además, también nos proporciona el número de páginas de nuestro sitio web y una lista de enlaces rotos.

Bing también ofrece una herramienta de generación de sitemaps¹⁶, que tiene en cuenta las directivas especificadas en el archivo robots.txt (en caso de que tengamos uno) a la hora de añadir o no una URL al *Sitemap*.

En la página “<https://code.google.com/archive/p/sitemap-generators/wikis/SitemapGenerators.wiki>”, Google proporciona una extensa lista de generadores de *sitemaps*.



The screenshot shows a web form titled "Please enter details for sitemap generation". It includes several sections: "Starting URL" with a text input field containing "http://ejemplo.com/"; "Change frequency" with a dropdown menu set to "None"; "Last modification" with radio buttons for "None", "Use server's response", and "Use this date/time" (with a date/time input field showing "2016-07-06 16:10:51"); and "Priority" with radio buttons for "None" and "Automatically Calculated Priority". At the bottom, there is a "Start" button and a note: "Maximum 500 pages will be indexed in sitemap". A link for a "Standalone version of Google sitemap generator" is also present.

Figura 17: Interfaz de una herramienta de generación de archivos *Sitemap*

2.4.4 URLs

Íntimamente ligadas a la estructura de enlaces, las URLs también juegan un papel importante en el posicionamiento web. Tomando la URL

¹⁶ Disponible en <https://www.microsoft.com/en-us/download/details.aspx?id=39060>

“<http://www.zara.com/abrigos/abrigo-blanco.html>”, la podemos diseccionar de la siguiente manera: **protocolo** (<http://>), **nombre de dominio** (www.zara.com) y **ruta** ([/abrigos/abrigo-blanco.html](http://www.zara.com/abrigos/abrigo-blanco.html)), que a su vez se puede subdividir en nombre de categoría ([abrigos](http://www.zara.com/abrigos/)) y nombre de recurso ([abrigo-blanco.html](http://www.zara.com/abrigos/abrigo-blanco.html)).

En la Sección 2.4.1, se expusieron ciertos aspectos a tener en cuenta a la hora de elegir el nombre de dominio. En esta sección, nos centraremos en la ruta.

¿Cómo optimizar la estructura de las URL?

Existen ciertas prácticas que debemos considerar a la hora de construir las URL de nuestro sitio web. El objetivo de estas prácticas es obtener las denominadas **semantic URLs**¹⁷, que son URLs fáciles de entender tanto para los usuarios, como para los motores de búsqueda y que organizan el contenido de las páginas de forma lógica e intuitiva.

A continuación, se listan algunos de los parámetros más importantes:

1. **Longitud: lo más corta dentro de lo posible.** La URL no sólo se muestra en los resultados del buscador, sino también en la barra de direcciones del navegador o en las redes sociales cuando los usuarios la comparten. Si es muy larga, no aparecerá al completo, sino cortada.
2. **Legibilidad:** para mejorar la legibilidad de las URLs **se recomienda usar guiones (-) entre las palabras**, ya que otros separadores, como espacios (%20), signos de suma (+) o guiones bajos (_) no son entendidos por algunas aplicaciones web. Por ejemplo, la URL “<http://www.zara.com/abrigos/abrigo-blanco.html>” se prefiere antes que otras variantes como “<http://www.zara.com/abrigos/abrigoblanco.html>” o “<http://www.zara.com/abrigos/abrigo%20blanco.html>”.
3. **Información:** una buena URL debe proporcionar **información sobre el contenido de la página**, por lo que conviene evitar URLs como “<https://www.amazon.de/gp/product/1250075610>”, que a pesar de ser legible, no proporciona absolutamente ninguna información. Una versión mejorada sería “<https://www.amazon.de/product/book/shelter-jung-yu>”.
4. **Palabras clave:** también es recomendable que las **palabras clave aparezcan en la URL**, sin caer en el *keyword stuffing* y siempre y cuando proporcionen información sobre el contenido real de la página a la que hacen referencia.

¹⁷ También conocidas como *clean URLs*, *user-friendly URLs* o *search engine-friendly URLs*.

5. URLs estáticas: aunque Google ha declarado¹⁸ que es capaz de interpretar correctamente URLs dinámicas, sigue siendo **preferible utilizar URLs estáticas cuando sea posible**, ya que las dinámicas no cumplen los puntos que acabamos de citar, que no están enfocados únicamente a los buscadores, sino también a los usuarios.

Si nuestro sitio web es dinámico (obtiene el contenido de una base de datos en función de ciertos parámetros cuando un usuario lo solicita), las URLs también serán dinámicas, pero se pueden convertir a estáticas [ver Tabla 4] usando herramientas como **mod_rewrite** para Apache o **ISAPI_rewrite** para Microsoft. De todas maneras hay que tener cuidado al realizar la conversión. Es preferible no convertir las URL dinámicas a convertirlas mal.

En este documento no vamos a describir en detalle el proceso de conversión, ya que es algo extenso. Se puede encontrar una guía para `mod_rewrite` en la página “<http://httpd.apache.org/docs/1.3/misc/rewriteguide.html>”.

Tabla 4: URLs dinámicas vs. URLs estáticas

URL dinámica	URL estática
<code>http://example.com/index.php?page=name</code>	<code>http://example.com/name</code>
<code>http://example.com/index.php?page=consulting/marketing</code>	<code>http://example.com/consulting/marketing</code>
<code>http://example.com/cgi-bin/feed.cgi?feed=news&frm=rss</code>	<code>http://example.com/news.rss</code>

2.4.4.1 Contenido duplicado

Un problema muy común, especialmente en sitios web con muchas páginas, y que influye negativamente en el posicionamiento web (está penalizado), es el contenido duplicado. Esto ocurre cuando tenemos **diferentes URLs que apuntan a una misma página**, es decir, a un mismo contenido (**true duplicate**). Las arañas indexarán las diferentes URLs, pero al ver que el contenido es duplicado, a sus ojos, la calidad del contenido del sitio web disminuye.

Para que el robot considere dos páginas como contenido duplicado, no tienen por qué tener el 100% del contenido idéntico (**near duplicate**). Más adelante veremos ejemplos concretos de este tipo.

¹⁸ Fuente: <https://webmasters.googleblog.com/2008/09/dynamic-urls-vs-static-urls.html>

¿Qué soluciones existen para deshacerse del contenido duplicado?

1. Redirección 301: el código de estado HTTP 301 redirecciona el tráfico de una URL. Indica, tanto a los usuarios, como a los buscadores, que la página se ha movido permanentemente a otra URL.
2. “Meta Robots”¹⁹: podemos usar los “Meta Robots” con los valores “NOINDEX, NOFOLLOW” o “NOINDEX, FOLLOW”.
3. Etiqueta canonical: en 2009 se creó una nueva etiqueta, que permite indicar en la cabecera de las páginas duplicadas mediante el atributo `REL="CANONICAL"` cuál es nuestra URL preferida para dicho contenido²⁰.
4. Utilizar los propios buscadores: tanto Google como Bing nos permiten indicar explícitamente las URLs que queremos que desaparezcan de sus índices [ver Figuras 18 y 19] o los parámetros de las URLs que queremos que ignoren [ver Figuras 20 y 21].

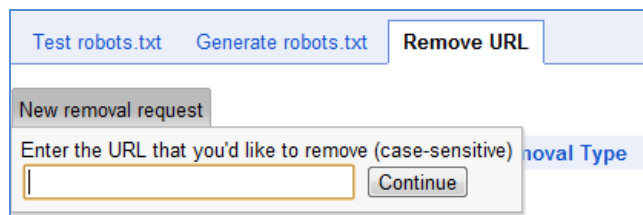


Figura 18: Interfaz de la herramienta de Google para eliminar URLs [12]

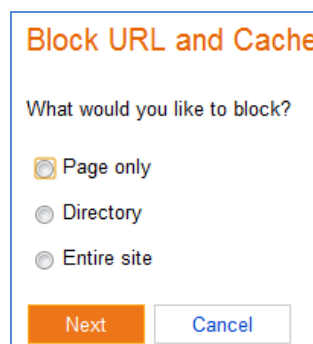


Figura 19: Interfaz de la herramienta de Bing para eliminar URLs [12]

¹⁹ También se puede utilizar el archivo robots.txt, pero su uso está desaconsejado.

²⁰ Código HTML: `<LINK REL="CANONICAL" HREF="https://ejemplo.com"/>`

Parameter	URLs monitored	What Googlebot should crawl	
utm_source	12	Let Googlebot decide (default)	Edit / Reset
utm_medium	11	Let Googlebot decide (default)	Edit / Reset
utm_campaign	7	Let Googlebot decide (default)	Edit / Reset
id	5	Let Googlebot decide (default)	Edit / Reset
text	5	Let Googlebot decide (default)	Edit / Reset

Figura 20: Interfaz de la herramienta de Google para bloquear parámetros URL²¹ [12]

Parameter Δ	Status	Source	Date
ft	Disabled	Bing	11/7/2011
id	Disabled	Bing	11/7/2011
max	Disabled	Bing	11/7/2011
msg	Disabled	Bing	11/7/2011

Figura 21: Interfaz de la herramienta de Bing para bloquear parámetros URL [12]

¿Por qué tengo contenido duplicado en mi sitio web?

Hay muchos errores que pueden causar la aparición involuntaria de contenido duplicado. A continuación se listan algunos de los más comunes y las soluciones más adecuadas a cada problema:

1. Versiones de la página web con y sin “www”: si los buscadores han indexado tanto “http://ejemplo.com”, como “http://www.ejemplo.com”, conviene utilizar redireccionamiento 301. Adicionalmente, Google permite establecer el dominio preferido en los ajustes de configuración del sitio web.
2. Duplicados de la página principal: nuestra página principal puede estar indexada como “http://www.ejemplo.com” o “http://www.ejemplo.com/index.html”. Este problema también puede resolverse mediante redireccionamiento 301, pero también es útil añadir a la página preferida la etiqueta canonical.
3. Identificadores de sesión: si utilizamos identificadores de sesión para seguir a los usuarios que utilizan nuestra página (“http://www.ejemplo.com/?session=123456789”), puede que se indexen páginas para cada identificador de sesión, creando miles de

²¹ En la interfaz de esta herramienta se tienen los parámetros que Google ha encontrado en las URLs de nuestro sitio web a la izquierda. Por defecto, el robot de Google rastrea (y posiblemente indexa) dichas URLs. Si queremos evitarlo, se puede hacer click en *edit*, para indicar el comportamiento deseado.

duplicados. Lo recomendable es eliminar completamente este número de la URL, ya que no aporta ninguna información adicional, y almacenarlo en una *cookie* o, si no hay otra opción, bloquear el parámetro (ej.: *session*) mediante las herramientas de Google y Bing.

4. Rutas duplicadas: es normal tener varias rutas que lleven a una misma página, el problema aparece cuando dichas rutas generan URLs duplicadas.

Por ejemplo, un blog de reseñas de libros puede que tenga las críticas ordenadas en varias categorías: por autor, por temática, por año, etc., por lo que una misma crítica puede ser accedida siguiendo varias rutas. Esto puede dar lugar a URLs duplicadas: “<http://bloglibros.com/jane-austen/orgullo-y-prejuicio.html>”, “<http://bloglibros.com/criticas-2016/orgullo-y-prejuicio.html>”, “<http://bloglibros.com/clasicos/orgullo-y-prejuicio.html>”, etc.

Éste problema se puede erradicar evitando desde un principio utilizar URLs basadas en ruta. Por mucho que parezca que favorece la navegación de cara al usuario, una página debería tener una URL única. Suponiendo que hayamos cometido el error y las URL hayan sido indexadas por los buscadores, las mejores opciones son el uso de la redirección 301 y la definición de URLs canónicas.

5. Ordenación/Filtrado de búsquedas: si tenemos búsquedas internas en nuestro sitio web, es probable que también ofrezcamos la posibilidad de ordenar los resultados (de manera ascendente o descendente, por ejemplo) y que esto dé lugar a URLs duplicadas: “<http://www.tiendaelectronica.com/search.php?keyword=portatil>” y “<http://www.tiendaelectronica.com/search.php?keyword=portatil&sort=desc>”. Ocurre lo mismo con los filtros: “<http://www.tiendaelectronica.com/search.php?keyword=portatil>” y “<http://www.tiendaelectronica.com/search.php?keyword=portatil&price=800>”.

Estos son ejemplos de *near duplicates*. En el primer caso, las páginas son diferentes, ya que el orden cambia, pero el contenido es el mismo. En el segundo, parte del contenido cambia, pero las páginas filtradas y sin filtrar tiene muchas partes en común. En estos casos, lo mejor es bloquear completamente las páginas duplicadas utilizando los “Meta Robots” con el valor `NOINDEX`.

6. Contenido paginado: cuando un artículo o una página de productos son muy largos, el contenido se suele dividir en páginas más cortas, lo que se denomina contenido paginado (“<http://www.tiendaelectronica.com/search.php?keyword=portatil>”, “<http://www.tiendaelectronica.com/search.php?keyword=portatil?page=2>”, etc.). Aunque el contenido de cada página varíe, estas páginas

tienen muchos elementos en común (etiquetas `<TITLE>`, `<HEAD>`, etc.), dando lugar a *near duplicates*.

Existen varias soluciones para este problema. Podemos utilizar los “Meta Robots” con el valor `NOINDEX, FOLLOW` en todas las páginas, excepto en la primera. De esta manera los robots no las indexarán, pero sí que rastrearán el contenido y seguirán los enlaces. Crear una página *View All*, en la que se visualice todo el contenido e identificarla como página canónica.

Google también soporta los atributos `REL` con los valores `NEXT` y `PREV`²², pero los demás buscadores no, así que no es una solución óptima.

Herramientas SEO

Existen varias herramientas que nos permiten encontrar contenido duplicado en nuestro sitio web. En las **herramientas para *webmasters* de Google** podemos encontrar una lista de duplicados de las etiquetas `<META>` y `<TITLE>` [ver Figura 22], que aunque no es una lista definitiva de contenido duplicado, nos proporciona pistas para encontrarlo.

Meta description	Pages
Duplicate meta descriptions	149
Long meta descriptions	0
Short meta descriptions	0
Title tag	Pages
Missing title tags	0
Duplicate title tags	73
Long title tags	0

Figura 22: Lista de duplicados de las etiquetas `<META>` y `<TITLE>` en Google. [12]

Aun así, estas herramientas no son infalibles y normalmente es necesario comprobarlo también manualmente. Un buen sitio para empezar, es comprobar los errores más comunes que hemos comentado. Una forma fácil de comprobarlos es utilizar la búsqueda de Google con el comando “**site:**”. En la Figura 23 se pueden ver algunos ejemplos:

²² En la primera página se añade el siguiente código en la cabecera: `<LINK REL="NEXT" HREF="http://www.tiendaelectronica.com/search.php?keyword=portatil?page=2"/>`
 En la página dos: `<LINK REL="PREV" HREF="http://www.tiendaelectronica.com/search.php?keyword=portatil"/>`
`<LINK REL="NEXT" HREF="http://www.tiendaelectronica.com/search.php?keyword=portatil?page=3"/>`
 Y así sucesivamente.

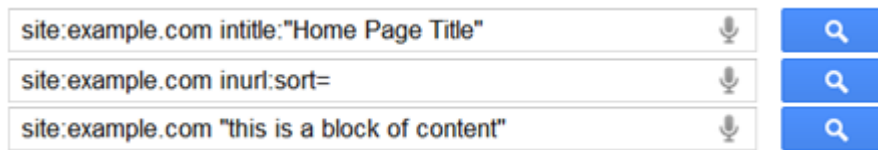


Figura 23: Ejemplos de uso del comando “site:” para encontrar contenido duplicado. [12]

En combinación con el **operador “intitle:”** nos permite comprobar si tenemos duplicados de una página determinada (en el ejemplo, de la página principal). El **operador “inurl:”** se puede utilizar para probar ciertos parámetros de las URL (*session, sort, price, etc.*). El tercer ejemplo, que utiliza sólo el comando “site:”, nos permite especificar un **bloque de contenido entre comillas**²³, útil para encontrar *near duplicates*.

Técnicas penalizadas

Hemos visto los errores que pueden causar la aparición de contenido duplicado de manera involuntaria, pero **también se puede crear a propósito con la intención de aumentar el volumen de páginas del sitio web**, lo cual está penalizado. Esto se puede llevar a cabo o bien duplicando páginas propias, o bien copiando sin permiso contenido de páginas web con una mejor reputación (***scraped content***). Ambas maneras de proceder están penalizadas y la segunda, además, puede estar incluso penada por ley.

En ocasiones puede ser interesante o necesario incluir contenido de otros sitios web. Esto se puede hacer si se tiene permiso del autor (***syndicated content***) y se recomienda añadir la etiqueta canonical indicando la página de origen del contenido. El uso de esta etiqueta hará que nuestra página no sea indexada, pero nos evitará penalizaciones.

2.5 OTRAS TÉCNICAS PENALIZADAS

A lo largo de este documento hemos ido viendo ciertas prácticas SEO penalizadas, como la duplicación de contenido, el *keyword stuffing* o el texto oculto con palabras clave. A continuación, vamos a ver brevemente alguna más:

1. ***Cloaking***: esta técnica tiene como base la técnica de texto oculto, que consistía en ocultar palabras clave a los usuarios, de manera que sólo las vieran los robots. *Cloaking* consiste básicamente en llevar dicha técnica al extremo, de manera que un buscador y un usuario vean páginas totalmente diferentes. La página que ven los buscadores está

²³ Como curiosidad, si quitamos el comando “site:” y dejamos el bloque de texto entre comillas, podemos comprobar si alguien está copiando nuestro contenido.

sobreoptimizada con el objetivo de mejorar el posicionamiento web y sería ininteligible para un usuario (en caso de que pudiera verla).

2. Páginas doorway: similar a cloaking, pero en este caso en lugar de ocultar el texto, se redireccionaba al usuario a la página normal mediante javascript, mientras que el buscador se quedaba en la página sobreoptimizada.
Esta técnica tenía su base en que los robots no eran capaces de entender javascript, o lo hacían de manera muy limitada, por lo que no se daban cuenta de que estábamos redireccionando a los usuarios. Esto ya no es cierto en la actualidad y la mayoría de los buscadores (incluyendo Google) son capaces de seguir una redirección javascript.
3. Granjas de enlaces: como hemos mencionado varias veces, uno de los factores que más influyen en el posicionamiento web, son los enlaces entre sitios web. Con el objetivo de proporcionar enlaces a sitios web para mejorar más rápidamente su posicionamiento web surgieron las granjas de enlaces. Una granja de enlaces es, por lo tanto, un sitio web cuyo único propósito es enlazar a otras páginas web. Con Google Panda, el robot de Google es capaz de identificar dichos sitios web y penaliza fuertemente a las páginas web enlazadas.
4. Ventas de enlaces: de igual manera, también está penalizado pagar a otros sitios web para que enlacen nuestra página.

2.6 FUTURO SEO

El mundo de SEO se encuentra constantemente en movimiento debido a los avances en las tecnologías web y móvil, al desarrollo de nuevos algoritmos por parte de los motores de búsqueda, etc. En marzo de 2016 tuvo lugar una nueva edición de la **Search Marketing Expo**, que es una de las conferencias más importantes en materia SEO y que se celebra varias veces al año en diferentes ciudades alrededor del mundo desde 2007. Profesionales del sector se reúnen para hablar del futuro de este mercado, entre ellos Google, por lo que es un buen lugar para extraer conclusiones.

Para terminar con esta introducción al mundo SEO y a partir de datos de la conferencia citada, así como de anuncios de los últimos años, presentamos algunas de las tendencias que tendrán más peso en el futuro próximo:

1. Adaptación a dispositivos móviles: no lo hemos tratado en este documento, pero Google también favorece los sitios web adaptados a dispositivos móviles. Ya en mayo de 2015, Google desveló que en 10 países (entre ellos EEUU y Japón), el número de búsquedas realizadas

por usuarios móviles superaba al realizado desde ordenadores [25]. Previsiblemente, este número continuará aumentando, por lo que las prácticas SEO orientadas a mejorar la experiencia móvil cobrarán más importancia.

2. Velocidad: ligado al punto anterior está la velocidad del sitio web, que también es un factor en el posicionamiento web que está cobrando cada vez más importancia. Existen varias prácticas SEO cuyo objetivo es reducir el tiempo de carga de la página, mejorando tanto la experiencia del usuario (y en particular, de usuarios móviles), como la de los robots al rastrear nuestras páginas. Recientemente, Google ha actualizado el proyecto de código abierto AMP²⁴ (*Accelerated Mobile Pages*), cuyo objetivo es proporcionar soluciones para mejorar la velocidad de las páginas web móvil.
3. HTTPS: Google ha expresado una preferencia clara por las páginas web que utilicen el protocolo HTTPS [26], indicando que dichas páginas reciben una pequeña ventaja con respecto a las que utilicen HTTP. Su intención es aumentar el peso de este factor en el futuro.
4. Indexación en tiempo real: hasta ahora, el contenido de nuestra página web no era rastreado e indexado en tiempo real, pero debido a la creciente demanda de información de última hora, Google lanzará a finales del verano 2016 un piloto de Google Indexing API [27], que nos permitirá solicitar el indexado de contenido en tiempo real.
5. Rich Cards: así se denomina un nuevo formato de visualización de resultados que utiliza marcado estructurado y que Google ya está implementando para recetas y películas [17].

²⁴ Más información en la página oficial de AMP: <https://www.ampproject.org/>

SEGUNDA PARTE

3 SISTEMAS Y GESTORES DE BASES DE DATOS

3.1 INTRODUCCIÓN A LOS SISTEMAS Y GESTORES DE BASES DE DATOS

3.1.1 Sistemas de información (SI)

En cualquier tipo de organización se dedican considerables recursos a la recolección, clasificación, procesamiento e intercambio de datos basados en procedimientos bien establecidos con vistas a alcanzar objetivos específicos. Esto es lo que se conoce como **sistema de información (SI)**. También podemos tomar la definición de Kenneth C. Laudon y Jane P. Laudon que lo expresa como “aquel conjunto de **componentes interrelacionados** que capturan, almacenan, procesan y distribuyen la información para apoyar la toma de decisiones, el control, análisis y visión de una organización” [1998].

Aunque por definición un sistema de información no tiene por qué ser **informático**, en la actualidad la mayoría de ellos lo son y es el tipo al que nos vamos a referir aquí. Los componentes de un SI son los siguientes:

1. Recursos humanos: personas que interactúan con el SI. Pueden ser los encargados de conseguir los datos con los que se rellenará el sistema, especialistas que analizan la información proporcionada por el SI o usuarios finales (trabajadores de oficina, clientes, etc.).
2. Hardware: dispositivos físicos utilizados en el procesamiento de los datos (CPUs, memorias, etc.).
3. Subsistema de comunicaciones: red que permite la conexión de diferentes dispositivos físicos.
4. Software: programas utilizados para la introducción, transformación y extracción de información.
5. Datos: los datos son el corazón de cualquier SI. Un dato por sí solo no contiene información. En combinación con otros datos y debidamente interpretados, permite obtener información.

En esta sección nos centraremos en el estudio de los **datos** y más en concreto, en el almacenamiento y la estructuración de los mismos.

3.1.2 De los ficheros a las bases de datos

En los **años sesenta**, los sistemas de almacenamiento de datos estaban basados en el uso de ficheros. Un **fichero** es una colección de información usada como unidad básica de almacenamiento en los sistemas informáticos. Cada aplicación, que constaba de uno o más programas, consultaba un fichero que era accedido de manera secuencial por los programas. Cuando se creaba una nueva aplicación, se creaba también un nuevo fichero con los datos que necesitaba dicha aplicación (una mezcla de datos nuevos y datos existentes en otros ficheros).

Como se puede inferir de esta explicación, los sistemas basados en ficheros presentan ciertos **problemas**:

1. Redundancia: este problema aparece cuando en los ficheros se tienen **datos que no aportan información**, ya que se pueden inferir de otros. El caso más sencillo consiste en tener el mismo dato almacenado en varios ficheros, lo que entre otras cosas, es un desperdicio de espacio de almacenamiento.
2. Dificultad de mantenimiento: tener redundancia se traduce en un **proceso de actualización de datos más costoso**. Si al tener que actualizar un determinado dato, no se actualiza en todos los lugares donde éste aparece, se producen **inconsistencias**.
3. Rigidez de búsqueda: al crear un fichero, **los datos se organizan de acuerdo con los tipos de accesos que se crean más frecuentes**. Más adelante, puede que se necesiten otros modos de acceso que no sean compatibles con la organización establecida.
4. Dependencia de los programas: a menudo, los programas que utilizan ficheros, son los que se encargan de definir la estructura y relaciones entre los datos almacenados en ellos (dónde comienza/acaba un campo, cuántos campos ahí, de qué tipo, etc.). Esto implica que cualquier **cambio en la estructura de un fichero, conlleva una modificación de los programas** que los manipulan.
5. Confidencialidad e integridad: en prácticamente cualquier organización resulta necesario **restringir la consulta de ciertos datos** a determinados usuarios (**confidencialidad**), lo cual no resulta fácil en los sistemas de ficheros. Lo mismo ocurre cuando queremos **evitar que personas no autorizadas modifique los datos (integridad)**.
6. Concurrencia: cuando un programa accede a un fichero, normalmente dicho fichero se bloquea, impidiendo el acceso simultáneo por parte de otros programas (conocido como **concurrencia**). En el peor de los casos, si no se bloquea y varios usuarios acceden a la vez, lo más probable es que se produzca pérdida de información.

La necesidad de solucionar estos problemas, así como otras exigencias (interrelación de datos), desembocó en la llegada de las **Bases de Datos (BBDD)** a principios de los años setenta.

3.1.3 Sistemas de Bases de Datos

Una **Base de Datos** podría definirse como un conjunto de datos y sus relaciones, almacenados con la mínima redundancia posible y de tal manera que aplicaciones y usuarios pueden acceder a ellos eficiente y simultáneamente.

Los **Sistemas de Bases de Datos** conllevan una fase de desarrollo más larga, pero una vez implementados tienen muchas **ventajas** con respecto a los sistemas basados en ficheros:

1. Redundancia: aunque no siempre es posible eliminar completamente la redundancia de datos, un objetivo claro de este tipo de sistemas es minimizarla.
2. Facilidad de mantenimiento: esto es un resultado directo de la minimización de redundancia. **A menor cantidad de datos redundantes, menor cantidad de problemas de actualización.** Además, algunos son capaces de detectar qué datos están duplicados y actualizarlos en consecuencia, lo que evita inconsistencias.
3. Flexibilidad de búsqueda: estos sistemas permiten realizar diferentes tipos de consulta, de manera que cada usuario puede acceder a los datos de la manera que mejor le convenga.
4. Independencia de los programas: en los sistemas de bases de datos, la definición y estructura de los mismos es **independiente** de los programas.
5. Confidencialidad e integridad: permiten el **establecimiento de restricciones a nivel de operación**. Esto quiere decir que se pueden definir usuarios o grupos de usuarios que tienen permiso para consultar o no los datos, para incluir o no nuevos datos, para modificar o no los datos existentes, etc.
6. Concurrencia: normalmente se permite el acceso concurrente a los datos y se evitan los problemas que hemos visto que ocurrían en los sistemas de ficheros.

El software de gestión de ficheros que se había utilizado hasta el momento para manejar datos, no estaba preparado para soportar estas características. Al igual que el almacenamiento de datos, este software evoluciona a medida que surgen nuevas necesidades y nuevas tecnologías. Se conoce como **Sistema Gestor de Bases de Datos (SGBD)**²⁵.

²⁵ En inglés se conocen como *Data Base Management Systems* (DBMS).

3.1.4 Sistemas Gestores de Bases de Datos

Un **Sistema Gestor de Bases de Datos (SGBD)** es un programa (o conjunto de programas) que permite a los usuarios realizar las siguientes operaciones sobre la base de datos:

1. **Definición:** utilizando un **lenguaje de definición de datos**, permite especificar la estructura, tipo y restricciones de los datos.
2. **Manipulación:** utilizando un **lenguaje de manipulación de datos**, posibilita la inserción, actualización, eliminación y consulta de los datos.

Además, el SGBD es el encargado de gestionar otros servicios, como la seguridad, integridad, concurrencia, recuperación de fallos, etc.

No hay que confundir, por lo tanto, Base de Datos o Sistema de Bases de Datos con Sistema Gestor de Bases de Datos. Así, si se ve la **Base de Datos** como un archivador de datos, el **Sistema Gestor de Bases de Datos** es el conjunto de programas mediante el cual se manipula y gestiona dicho archivador.

- Ejemplos de SGBD: Oracle, Ingres, Postgre SQL, Access, MySQL²⁶, etc.
- Ejemplos de BBDD: un listín telefónico, la lista de alumnos de clase, etc.

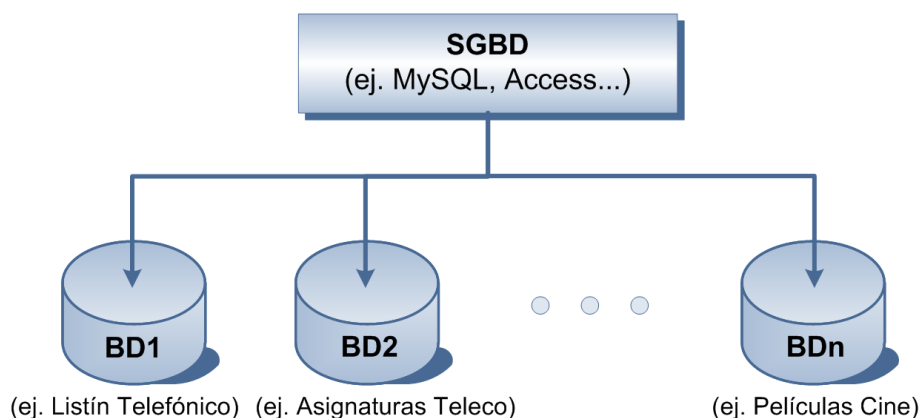


Figura 24: Sistemas Gestores de Bases de Datos y Bases de Datos.

Algunos **escenarios** en los que intervienen Sistemas Gestores de Bases de Datos y Bases de Datos son los que a continuación se presentan de manera esquematizada:

²⁶ Este será el Sistema Gestor de Bases de Datos usado en la presente asignatura.

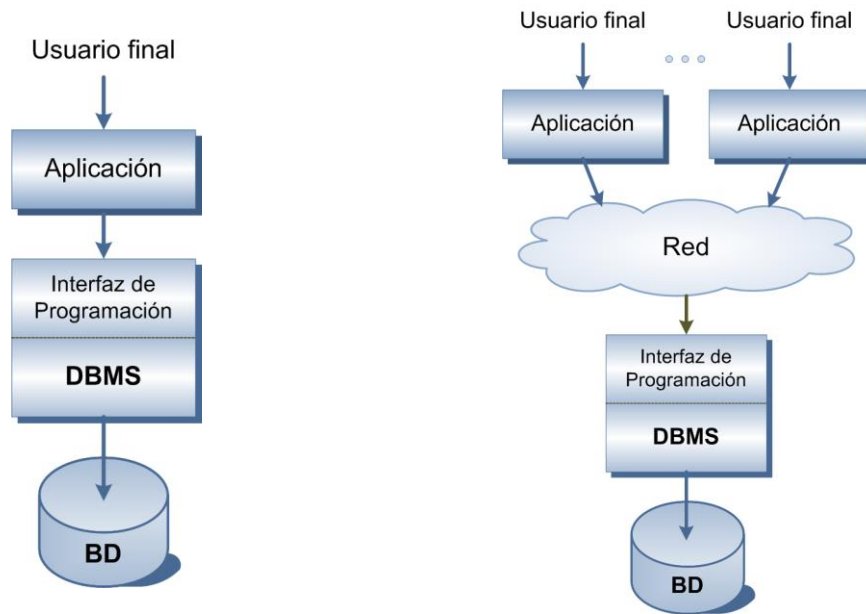


Figura 15: Escenarios en los que intervienen Sistemas Gestores de Bases de Datos y Bases de Datos.

3.1.5 Evolución de los modelos de BD y los SGBD

Partiendo de la base de que una BD es una representación de una parte de la realidad, los SGBD pueden utilizar diferentes **modelos de BD** para construir dicha representación de la realidad. El **modelo de BD** son las herramientas conceptuales que utiliza el SGBD: estructuras de datos, tipos de restricciones, operaciones para trabajar con los datos.

El primer modelo de BD, que apareció a principios de los años sesenta, fue el **modelo jerárquico**, cuyos datos son registros interrelacionados con estructura de árboles (un “registro padre” puede tener varios “registros hijos”). Más adelante, surgieron SGBD basados en el **modelo en red**, que permite a los “registros hijos” tener varios “registros padre”, lo que satisfacía la necesidad de representar interrelaciones más complejas. Los sistemas basados en estos dos modelos constituyen la **primera generación** de SGBD.

En 1970, E. F. Codd en el instituto de Investigación de IBM publicó el artículo "A Relational Model of Data For Large Shared Data Banks", en el que presentaba el **modelo relacional**, que se basa en la agrupación de datos en tablas bidimensionales interrelacionadas. Los primeros sistemas que implementaron este modelo aparecieron casi una década después. Destacan, entre otros, System R de IBM e Ingres de la Universidad de Berkley. Estos constituyen la **segunda generación** de SGBD.

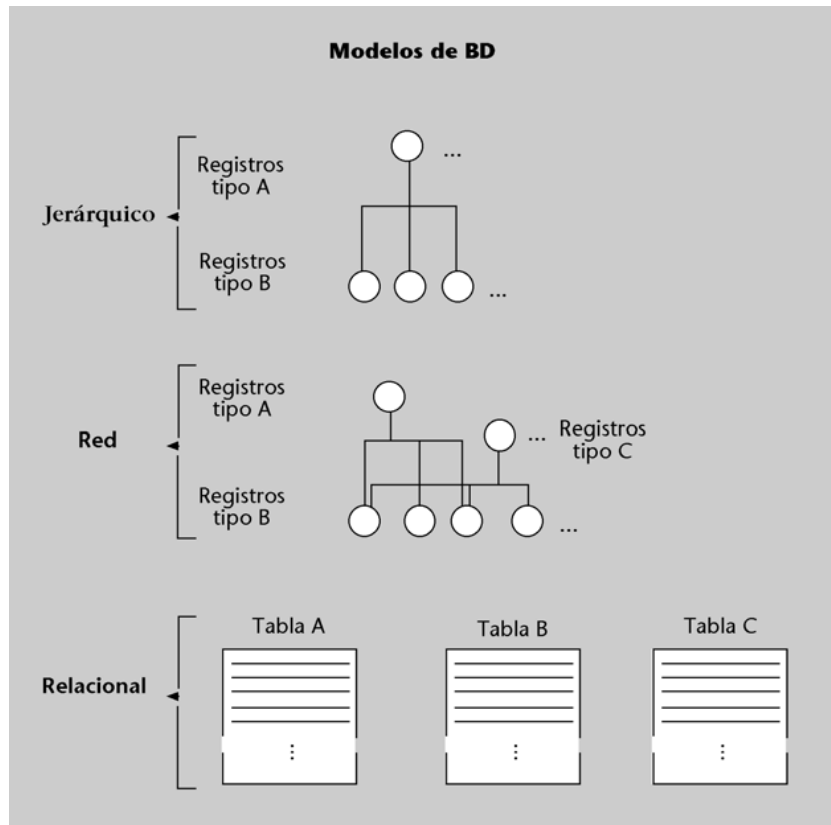


Figura 26: Comparación esquemática de los modelos de BD

La mayoría de los sistemas de bases de datos que se encuentran en funcionamiento en la actualidad son relacionales y son los que estudiaremos en este documento. Pero antes de centrarnos en ellos, vamos a ver brevemente las tendencias que están tomando fuerza en la actualidad.

Uno de las limitaciones de los SGBD relacionales es la variedad de tipos de datos que permiten definir, que se ha ido ampliando para permitir datos de tipo **multimedia** y recientemente, **tipos de datos abstractos** (TAD) que puede definir el desarrollador en función de sus necesidades. Este modelo de BD que incorpora el uso de TAD se denomina **modelo relacional con objetos** y los sistemas basados en él constituyen la **tercera generación** de SGBD. Los primeros SGBD que incluyeron esta posibilidad fueron Oracle (versión 8), Informix (versión 9) e IBM/DB2/UDB (versión 5). El rápido crecimiento de **Internet**, también ha influido en este tipo de SGBD, ya que muchas páginas web se basan en bases de datos, en las que los datos multimedia y la orientación a objetos juegan un papel fundamental.

Actualmente, los SGBD también se están adaptando a un tipo de aplicación de las bases de datos conocido como **almacén de datos**²⁷. Los almacenes de datos contienen datos extraídos de las BBDD de la organización correspondiente, así como

²⁷ En inglés, *Data Warehouse*.

de otras fuentes externas. Estos almacenes son utilizados por analistas, que llevan a cabo estudios, cuyo objetivo es descubrir conocimientos útiles para la organización. Algunos de los desafíos a los que se enfrentan los SGBD son la consolidación de datos procedentes de distintas fuentes o la creación y mantenimiento de réplicas.

3.2 SISTEMAS DE BASES DE DATOS RELACIONALES

3.2.1 Definiciones básicas

El **modelo relacional** se basa en el concepto matemático de relación, que gráficamente se representa como una tabla bidimensional.

Una **relación**²⁸ es, por lo tanto, una tabla con **registros o filas** y **campos o columnas** [ver Figura 27]. Aunque relación es el término formal, de ahora en adelante utilizaremos el término tabla para referirnos a una relación.

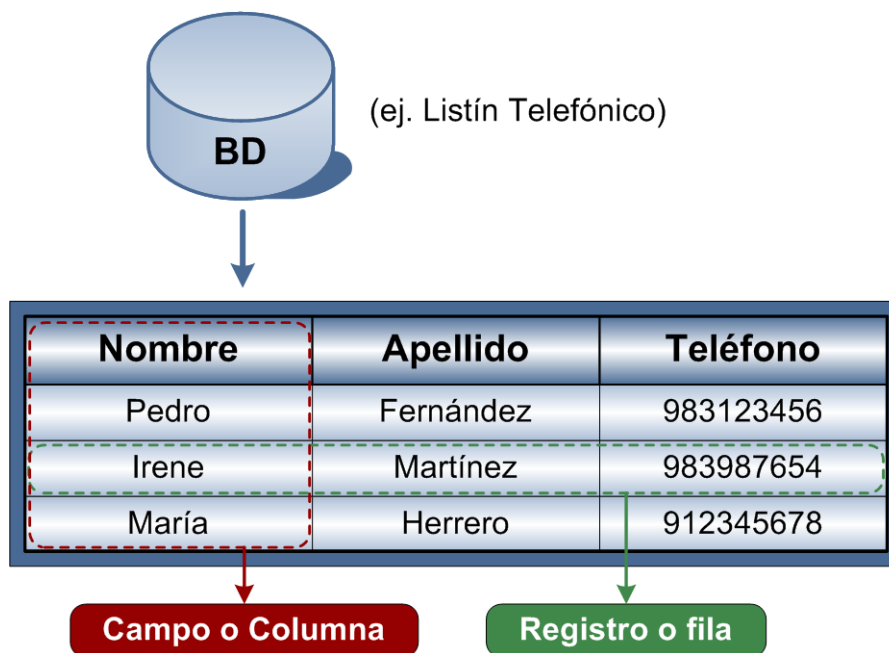


Figura 27: Ejemplo de una Tabla de una Base de Datos.

Un **atributo**²⁹ es el nombre de un campo de una tabla. En la tabla de la Figura 27, los atributos son nombre, apellido y teléfono.

²⁸ En inglés, *relations*.

²⁹ En inglés, *attributes*.

Un **dominio**³⁰ es el conjunto de valores legales de uno o varios atributos, es decir, un dominio **define el significado y la fuente de los valores** que pueden tomar los atributos. Por ejemplo, el dominio del campo teléfono es un número de teléfono de ocho o nueve dígitos.

Una **tupla**³¹ es un registro de una tabla. En la Figura 27, cada tupla tiene tres valores, uno para cada atributo.

En el resto del documento utilizaremos el término **campo** para referirnos a un atributo y el término **registro** para referirnos a una tupla.

Otros términos de interés son el **grado de una tabla**, que es el número de campos que contiene y que se suele mantener **constante** desde la definición de la tabla, y la **cardinalidad de una tabla**, que es el número de registros que contiene y que **varía** constantemente. La tabla de la Figura 27 es de grado 3 y de cardinalidad también 3.

Finalmente, una **base de datos relacional** se compone de varias tablas bidimensionales interrelacionadas normalizadas (más adelante estudiaremos en detalle el proceso de normalización de las tablas).

3.2.2 Claves

En una tabla **no pueden existir registros repetidos**, es decir, registros con valores idénticos para todos los campos. Por lo tanto, cada registro se puede identificar de modo único, para lo que se utilizan **claves**. Una **clave** es el campo o combinación de campos usados para identificar un registro. Existen diferentes tipos de claves y para que seamos capaces de entender la diferencia entre ellas, vamos a utilizar la tabla ciudades:

Tabla 5: Ciudades

cod	ciudad	num_habitantes	pais
89574	Palencia	80178	España
56381	Guadalajara	83720	España
95552	Guadalajara	1495000	México
52014	Nuremberg	495121	Alemania

Una **superclave** es un campo o conjunto de campos que identifican de modo inequívoco los registros de una tabla. Ejemplos de superclaves de ciudades: cod, combinación de cod y ciudad, combinación de ciudad y pais.

³⁰ En inglés, *domain*.

³¹ En inglés, *tuple row*.

Una **clave candidata** es una superclave en la que ninguno de sus subconjuntos es a su vez superclave de la tabla. Una clave candidata cumple las siguientes **propiedades**:

1. **Unicidad**: los valores del campo (o campos) son únicos para cada registro.
2. **Irreducibilidad**: no se pueden eliminar campos de la clave sin destruir la unicidad.

Las tres superclaves que hemos seleccionado (`cod`, `cod` y `ciudad`, `ciudad` y `pais`) cumplen la propiedad de unicidad, pero sólo dos cumplen la propiedad de irreducibilidad. Tomando la segunda superclave, `cod` y `ciudad`, es posible eliminar el campo `ciudad` y la clave seguirá cumpliendo la propiedad de unicidad. Sin embargo, si tomamos `ciudad` y `pais`, por sí solos no identifican inequívocamente a un registro, pero juntos sí, por lo que es clave candidata. Como está formada por más de un campo, se denomina **clave compuesta**. También `cod` es clave candidata, en este caso, **simple** (un solo campo).

De entre las claves candidatas se escoge la **clave primaria o principal**, que será la que utilizemos de forma oficial para identificar a los registros de una tabla. Las claves candidatas que no son escogidas se denominan **claves alternativas**. Por ejemplo, `cod` puede ser la clave primaria y la combinación de `ciudad` y `pais`, la clave alternativa.

Por último, existen las claves **externas, secundarias, ajenas o foráneas**. Una clave ajena es un campo (o campos) de una tabla cuyos valores coinciden con los valores de la clave primaria de otra tabla. Las claves ajenas **representan las relaciones entre datos de diferentes tablas**.

Por ejemplo, vamos a suponer que la base de datos en la que se encuentra la tabla `ciudades`, contiene también la tabla `empleados` [ver Tabla 6]. El campo `cod_ciudad` es una clave ajena, ya que sus valores constituyen la clave primaria en la tabla `ciudades`. Como se puede apreciar, los valores de la clave ajena pueden aparecer repetidos.

Tabla 6: Empleados

cod_employado	nombre	apellido	cod_ciudad
123	Pedro	Fernandez	89574
508	Irene	Martinez	89574
22	Maria	Herrero	56381
196	Juan	Aguado	52014

3.2.3 Reglas de integridad

Las **reglas de integridad** establecen ciertas pautas que los datos almacenados en una base de datos relacional deben cumplir, de manera que se garantice la integridad de los mismos:

1. Regla de integridad de entidades: esta regla hace referencia a las **claves primarias** y establece que ninguno de los campos que las componen puede tener el valor **nulo**³².
2. Regla de integridad referencial: la segunda regla se aplica a las **claves ajenas** y establece que si en una tabla aparece una clave ajenas, sus valores deben coincidir con los valores de la clave primaria a la que hace referencia o deben ser nulos.
3. Reglas o restricciones de negocio: también se conocen como restricciones semánticas y hacen referencia a las **necesidades específicas de una BD**. Por ejemplo, si la organización a la que pertenecen los empleados de la tabla `empleados` tiene como máximo 600, el SGBD admitir la definición de una restricción que no permita superar este número. Las reglas de negocio también determinan las relaciones entre tablas (que veremos a continuación). Por ejemplo, un empleado sólo puede tener asignado un código de ciudad, pero un código de ciudad puede estar asignado a varios empleados.

3.3 ¿CÓMO DISEÑAR UNA BASE DE DATOS?

3.3.1 Introducción

El diseño de una base de datos relacionales es un proceso complejo y que consta de varias fases:

1. Diseño conceptual: en esta fase se construye un **esquema conceptual** a partir de la información que incluya la BD. Este esquema es completamente independiente del SGBD que se vaya a usar, del modelo de datos utilizado, de los programa de aplicación, hardware, etc.

³² Donde nulo representa la ausencia de información. Es importante distinguirlo del valor cero o de una cadena vacía, que sí tienen significado. En MySQL (el SGBD utilizado en este documento) se representa mediante `NULL`.

2. Diseño lógico: en esta fase, se crea un **esquema lógico** a partir del esquema conceptual. El esquema lógico se adapta a un modelo de datos determinado (jerárquico, relacional, etc.). Sigue siendo independiente del resto de consideraciones físicas. En esta etapa se aplica la normalización, que veremos más adelante.
3. Diseño físico: por último, el diseño físico describe la implementación del esquema lógico, es decir, determinar las estructuras de almacenamiento de los datos y los métodos de acceso. Para el caso que nos ocupa (modelo de datos relacional), esta fase consiste en definir las tablas, las restricciones que se deben cumplir, el modelo de seguridad a seguir, etc.

En las siguientes secciones no vamos a detallar todos los procesos en profundidad, sino que vamos a dar una visión general centrándonos en la relación entre tablas y el proceso de formalización.

3.3.2 Relaciones entre entidades

El primer paso en el desarrollo de una base de datos es el **establecimiento de las entidades y las relaciones entre ellas**. Las entidades se deducen de las especificaciones que recibimos para la base de datos y son los **conceptos** más importantes en los que podemos agrupar la información, aquellos que existen por sí mismos y que en general, tendrán varias propiedades. Normalmente estas entidades, si han sido correctamente identificadas acabarán siendo las **tablas** de nuestra base de datos, y sus propiedades, los diferentes campos que las componen.

Una vez establecidas las entidades, debemos establecer la **relación** entre ellas. La mayoría de las relaciones son **binarias**, es decir, **entre dos entidades** y son las que vamos a ver en este documento. Otro factor que define una relación es la **cardinalidad**. Cuando establecemos una relación entre dos entidades, debemos definir también la cardinalidad mínima y máxima con la que participa cada una de ellas. Si la **cardinalidad mínima** de una entidad es **0**, quiere decir que su participación es **opcional** y si es **1**, que es **obligatoria**. La **cardinalidad máxima** indica si la ocurrencia³³ de una entidad sólo puede relacionarse con una ocurrencia de la otra entidad (**valor 1**) o si puede relacionarse con más de una (**valor 0**).

En función de la cardinalidad máxima, podemos hablar de:

1. Relación uno a uno: ambas entidades participan con cardinalidad máxima 1. **Traducido a tablas y registros**: cada registro de una tabla tiene uno, y solo un registro correspondiente en la tabla relacionada. Por este motivo, sería posible combinar las tablas que están así relacionadas en una sola tabla compuesta por todas las columnas de las tablas relacionadas.

³³ Considerando que la entidad se ha elegido correctamente y que por lo tanto, es el equivalente de una tabla, una **ocurrencia de la entidad**, se puede considerar como el equivalente de un **registro**.

Las relaciones de uno a uno se suelen utilizar para dividir tablas con muchas columnas en otras más manejables y reducir así el tiempo necesario para acceder a un determinado conjunto de datos.

Ejemplo de relación uno a uno:

- Cada ciudadano tiene un único número de DNI y cada número de DNI corresponde a un único ciudadano. Ningún ciudadano puede tener dos números de DNI, ni un número de DNI puede aplicarse a dos ciudadanos.

2. Relación uno a varios: una entidad participa con cardinalidad máxima 1 y la otra, con cardinalidad máxima n. **Traducido a tablas y registros**: vinculan un solo registro de una tabla, con dos o más registros de otra tabla, mediante una relación entre la clave principal de la tabla base y la clave externa correspondiente en la tabla relacionada.³⁴

Ejemplo de relación uno a varios:

- Un empleado sólo vive en una ciudad, pero una ciudad puede ser el lugar de residencia de varios empleados.
- Un vendedor puede gestionar varios clientes, pero un mismo cliente no puede ser gestionado por dos vendedores.

3. Relación varios a varios: las dos entidades participan con cardinalidad máxima n. **Traducido a tablas y registros**: para expresar esta relación es necesario crear una tabla extra, con la que las otras dos tablas tendrán una relación uno a varios.

Ejemplo de relación varios a varios:

- Un disco de música puede incluir canciones de varios artistas y una canción puede estar incluida en varios discos diferentes.
- Un profesor puede impartir varias asignaturas y una asignatura puede estar impartida por varios profesores.
- Un pedido puede incluir varios productos y un producto puede formar parte de varios pedidos.

3.3.3 Normalización

El proceso de **normalización**, consiste en la realización de una serie de pasos o normas de forma que, tras aplicar todas ellas, se obtiene la **descomposición de**

³⁴ Aunque la clave externa de la tabla relacionada pueda ser el componente de una clave principal compuesta, se considera una clave externa a efectos de relación.

los datos en diferentes tablas cuya estructura se considera óptima para su implementación, gestión y explotación desde diferentes aplicaciones. Los objetivos básicos son **evitar los problemas de integridad y redundancia en los datos**, además de intentar permitir un **máximo rendimiento** a las aplicaciones que deban utilizar dichos datos, ya que trata de evitar la dependencia entre las operaciones de inserción, actualización, etc., sobre los elementos de las tablas de la base de datos. Además, también reduce potencialmente las operaciones de reorganización que pueda ser necesario realizar para incorporar nuevos datos en el futuro.

La normalización se puede aplicar o bien **cuando ya tenemos el esquema lógico**, para eliminar las dependencias indeseadas entre campos o bien al comienzo, como **técnica de diseño de bases de datos** (en los ejemplos A, B, C y D la utilizaremos como técnica de diseño).

La normalización tiene tres etapas principales que transforman las tablas no normales en normalizadas: **primera, segunda y tercera forma normal**. Una tabla está en una forma normal, ya sea primera, segunda o tercera forma normal, cuando satisface el conjunto de restricciones impuestas por dicha norma. La única norma obligatoria es la primera, pero se recomienda llegar por lo menos hasta la tercera.

Una tabla está en:

- **Primera Forma Normal (1FN)**, si cada registro tiene valores **atómicos** (valor único) en todos sus campos.
- **Segunda Forma Normal (2FN)**, si está en 1FN y además, los campos que no forman parte de la clave primaria son completamente dependientes de ésta. Esta norma **sólo se aplica a tablas cuyas claves primarias sean compuestas**, ya que puede ocurrir que varios campos sean sólo dependientes de una parte de la clave. Esto no puede ocurrir si la clave es simple. Por lo tanto, las tablas cuya clave primaria sea simple y que estén en 1FN, automáticamente están también en 2FN.
- **Tercera Forma Normal (3FN)**, si está en 2FN y además, no contiene campos que no dependen únicamente de la clave primaria.

Una vez que los datos están en 3FN, están ya de manera automática en la 1FN y la 2FN. Por tanto, el proceso total se puede realizar de una forma más rápida y sencilla usando directamente la 3FN, que haciéndolo forma a forma.

3.4 EJEMPLOS DE DISEÑO

A continuación, vamos a ver varios ejemplos de diseño. En los ejemplos A, B, C y D utilizamos el proceso de normalización para el diseño de pequeñas bases de datos.

3.4.1 Ejemplo A

Se quiere diseñar una base de datos con la siguiente información: nombre y DNI de los profesores del departamento, asignaturas que imparten especificando su código, su nombre y el coste del material empleado en las mismas. Cada uno de los profesores puede impartir varias asignaturas, y cada asignatura puede ser impartida por varios profesores.

Versión 1

Base de Datos: ProfesoresDepartamento

Se define una tabla que consta de una serie de campos. Se escogen uno o más campos, en este caso solo un campo, en concreto el campo `dni` que se muestra subrayado, para conformar la clave primaria, teniendo en cuenta que constituya un identificador único para los registros de esa tabla.

profesores
<u>dni</u> nombre código asignatura coste

profesores (dni, nombre, código, asignatura, coste)

Y se realiza la inserción inicial de datos mostrada a continuación. Tal y como puede observarse, cada profesor puede impartir varias asignaturas y, además, es posible que la docencia de una asignatura sea compartida por varios profesores.

Tabla: profesores

<u>dni</u>	nombre	código	asignatura	coste
1234A	M ^a Ángeles	13054	STI	1
		13122	TTR	2
5678B	Nacho	13054	STI	1
		14984	TRIII	2
9012C	Belén	13055	TRI	3

El diseño propuesto para la tabla `profesores` no está en 1FN, ya que una relación está en 1FN cuando en la tabla no hay grupos de repetición o, dicho de otro modo, cuando todas y cada una de las celdas de las diferentes columnas y registros contienen valores atómicos. Así, la información asociada a **cada una de las celdas es un valor único** y no una colección de valores en un número variable. Esto equivale a decir que las **tablas son planas**. Una tabla plana tiene solo dos dimensiones: longitud (cardinalidad o número de registros) y anchura (grado o número de campos o columnas). Así, la tabla no puede contener celdas con más de un valor, porque para que una celda contenga más de un valor, la representación del contenido requeriría una tercera dimensión (profundidad) que permitiera mostrar los distintos valores de los datos.

La solución propuesta es expandir la tabla, añadiendo filas. También se va a expandir la clave primaria.

Versión 2

Base de Datos: ProfesoresDepartamento



`profesores (dni, código, nombre, asignatura, coste)`

Tabla: profesores

<u>dni</u>	nombre	<u>código</u>	asignatura	coste
1234A	M ^a Ángeles	13054	STI	1
1234A	M ^a Ángeles	13122	TTR	2
5678B	Nacho	13054	STI	1
5678B	Nacho	14984	TRIII	2
9012C	Belén	13055	TRI	3

La ventaja con respecto al diseño propuesto en la versión anterior de la tabla `profesores` es que no hay problemas si se imparten más o menos asignaturas, basta con añadir o eliminar filas o registros. Puede afirmarse que la tabla ya está en 1FN.

Sin embargo, el diseño propuesto para la tabla `profesores` aún presenta los siguientes problemas:

- Existe mucha redundancia. El nombre de las asignaturas o el de los profesores aparecen muchas veces, etc., por lo que es más probable cometer incoherencias que den lugar a inconsistencias en los datos almacenados.
- La asignatura depende de código, pero no de dni y el nombre del profesor depende de dni pero no de código, es decir, hay campos no clave que solo dependen de parte de la clave. Esto significa que el diseño propuesto para la tabla profesores no está en 2FN.

La solución propuesta es fragmentar la tabla, situando cada campo con su clave primaria adecuada, de forma que los campos no clave dependan totalmente de toda la clave primaria, bien sea esta una columna o una combinación de varias columnas, y no solo de una parte de ella.

Versión 3

Base de Datos: ProfesoresDepartamento

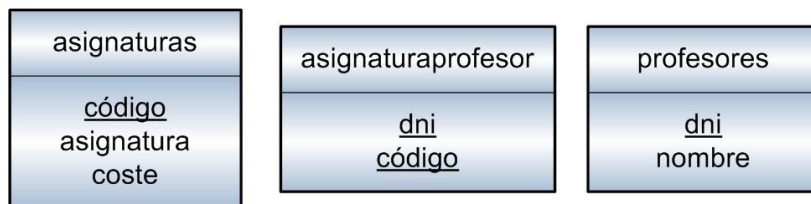


Tabla: asignaturas

<u>código</u>	asignatura	coste
13054	STI	1
13122	TTR	2
14984	TRIII	2
13055	TRI	3

Tabla: profesores

<u>dni</u>	nombre
1234A	Mª Ángeles
5678B	Nacho
9012C	Belén

Tabla: asignaturaprofesor

<u>código</u>	<u>dni</u>
13054	1234A
13054	5678B
14984	5678B
13122	1234A
13055	9012C

El diseño propuesto, formado por tres tablas diferentes, está en 3FN, ya que cada columna depende directamente de la clave primaria y no hay dependencias entre campos no pertenecientes a la clave. Nótese que dicho diseño hace uso de claves foráneas.

Nótese además que, tal y como sucede en el diseño propuesto, pueden existir **interrelaciones entre las tablas** que forman una base de datos relacional. Si esto es así, al codificar las diferentes acciones a realizar sobre las diferentes tablas de la base de datos, se hace necesario establecer **reglas** que eviten realizar acciones inadecuadas sobre la base de datos. Así, por ejemplo, si al modificar alguna tabla hay que realizar una modificación en otra, esta debe hacerse por el código de forma automática. En el ejemplo mostrado, si se borra un profesor de la tabla de profesores, también habría que eliminarle de la tabla asignaturaprofesor.

3.4.2 Ejemplo B

Diseñar una base de datos con la siguiente información: asignatura, indicando su código, nombre y el coste del material) y el profesor principal de la misma, indicando su DNI y su nombre. Por lo tanto, en este caso, cada profesor puede impartir varias asignaturas, pero de cada asignatura sólo se almacena el profesor principal.

Ejemplo de datos:

- M^a Ángeles (1234A) es la profesora principal de STI (13054) y TTR (13122), con coste 1 y 2 respectivamente.
- Belén (9012C) es la profesora principal de TRI (13055), con coste 3.
- Nacho (5678B) es el profesor principal de TRIII (14984), con coste 2.

Versión 1

Base de Datos: ProfesoresTelecoB



asignaturas (código, asignatura, coste, dni, nombre)

Tabla: asignaturas

<u>código</u>	asignatura	coste	dni	nombre
13054	STI	1	1234A	M ^a Ángeles
13055	TRI	3	9012C	Belén
13122	TTR	2	1234A	M ^a Ángeles
14984	TRIII	2	5678B	Nacho

Tal y como puede observarse en la inserción inicial de datos mostrada, en esta ocasión se considera que cada profesor puede impartir varias asignaturas, pero no se permite que la docencia de una asignatura sea compartida por varios profesores.

El diseño propuesto está en 1FN, ya que no hay celdas con valores múltiples. Igualmente también está en 2FN ya que los campos no clave dependen por completo de la clave. Sin embargo, no está en 3FN, ya que el campo no clave nombre depende del campo no clave dni.

Versión 2

Base de Datos: ProfesoresTelecoB

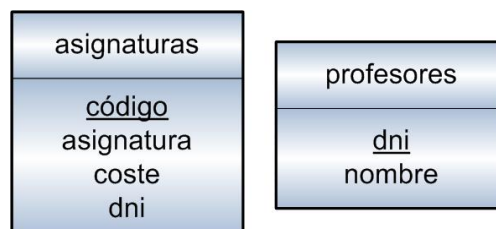


Tabla: profesores

<u>dni</u>	nombre
1234A	M ^a Ángeles
5678B	Nacho
9012C	Belén

Tabla: asignaturas

<u>código</u>	asignatura	coste	<u>dni</u>
13054	STI	1	1234A
13055	TRI	3	9012C
14984	TRIII	2	5678B
13122	TTR	2	1234A

El diseño propuesto, formado por dos tablas, sí está en 3FN, ya que cada campo no clave depende totalmente de la clave y no depende de ningún otro campo no clave.

Nótese además que el campo dni de la tabla asignaturas actúa como clave ajena puesto que representa en dicha tabla el campo que es la clave principal de la tabla profesores.

Finalmente señalar que, en este ejemplo, al ser la relación entre las tablas de uno – profesores – a varios – asignaturas – y no de varios – profesores – a varios – asignaturas – como en el caso ejemplo anterior, es suficiente con un diseño formado por dos tablas.

3.4.3 Ejemplo C

Se desea diseñar una base de datos con información sobre los pedidos realizados, en concreto, el número de pedido y la fecha del pedido y, para cada producto incluido en un pedido, el número de producto, la descripción del producto y el número de unidades pedidas. Cada pedido puede estar compuesto de diversos productos, mientras que estos productos pueden aparecer en diferentes pedidos.

Ejemplo de datos:

- El pedido #13054 se ha realizado el 1/11/2004, con los productos 101 (4 camisas) y 201 (5 bufandas).
- El pedido #13055 se ha realizado el 1/11/2004, con los productos 103 (6 abrigos) y 101 (7 camisas).
- El pedido #13122 se ha realizado el 3/11/2004, con los productos 101 (8 camisas) y 204 (5 pantalones).

Versión 1

Base de Datos: PedidosRealizados

pedidos
<u>numpedido</u> fechapedido numproducto descripproducto numunidades

Pedidos (numpedido, fechapedido, numproducto, descripproducto, numunidades)

Tabla: pedidos

<u>numpedido</u>	fechapedido	numproducto	descripproducto	numunidades
13054	01/11/2004	101 201	Camisa bufanda	4 5
13055	01/11/2004	103 101	Abrigo camisa	6 7
13122	03/11/2004	101 204	Camisa pantalón	8 5

Tal y como puede observarse en la inserción inicial de datos mostrada, se considera que en un pedido se pueden solicitar varios productos y que cada producto puede solicitarse en varios pedidos.

El diseño propuesto para la tabla `pedidos` no está en 1FN, ya que no todas las celdas contienen valores atómicos.

Para solucionarlo, se propone expandir la tabla, añadiendo filas o registros. Además también se va a expandir la clave primaria.

Versión 2

Base de Datos: PedidosRealizados

pedidos
<u>numpedido</u>
fechapedido
<u>numproducto</u>
descripproducto
numunidades

Pedidos (numpedido, fechapedido, numproducto, descripproducto, numunidades)

<u>numpedido</u>	fechapedido	<u>numproducto</u>	descripproducto	numunidades
13054	01/11/2004	101	Camisa	4
13054	01/11/2004	201	bufanda	5
13055	01/11/2004	103	abrigo	6
13055	01/11/2004	101	Camisa	7
13122	03/11/2004	101	Camisa	8
13122	03/11/2004	204	pantalón	5

El diseño propuesto para la tabla `pedidos` ya está en 1FN ya que todas las celdas contienen valores atómicos.

Sin embargo, el diseño propuesto para la tabla `pedidos` aún presenta los siguientes problemas:

- Existe mucha redundancia, por lo que es más probable cometer incoherencias que den lugar a inconsistencias en los datos almacenados.
- La fecha del pedido depende del número del pedido pero no del número de producto y la descripción del producto depende del número del producto, pero no del número del pedido, es decir, hay campos no clave que solo dependen de parte de la clave. Esto significa que el diseño propuesto para la tabla `pedidos` no está en 2FN.

La solución propuesta es fragmentar la tabla, situando cada campo con su clave primaria adecuada, de forma que los campos no clave dependan totalmente de toda la clave primaria, bien sea esta una columna o una combinación de varias columnas, y no solo de una parte de ella.

Versión 3

Base de Datos: PedidoRealizados

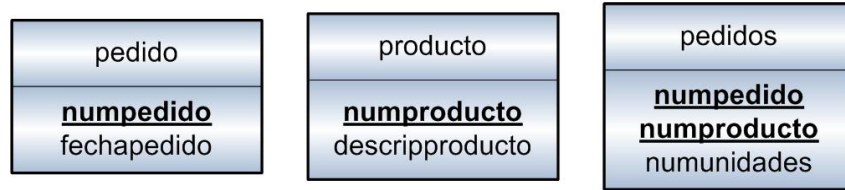


Tabla: pedido

<u>numpedido</u>	Fechapedido
13054	01/11/2004
13055	01/11/2004
13122	03/11/2004

Tabla: producto

<u>numproducto</u>	descripproducto
101	Camisa
201	bufanda
103	abrigo
204	pantalón

Tabla: pedidos

<u>numpedido</u>	<u>numproducto</u>	numunidades
13054	101	4
13054	201	5
13055	103	6
13055	101	7
13122	101	8
13122	204	5

El diseño propuesto, formado por tres tablas diferentes, está en 3FN, ya que cada columna depende directamente de la clave primaria y, por otra parte, no hay dependencias entre campos no pertenecientes a la clave. Nótese además que dicho diseño hace uso de claves foráneas.

3.4.4 Ejemplo D

Se quiere diseñar una base de datos con la siguiente información en relación a clientes y vendedores: número de cliente, nombre del cliente, dirección del cliente, número de vendedor y nombre del

vendedor. Cada vendedor lógicamente podrá atender a diversos clientes, pero cada cliente sólo podrá ser atendido por un vendedor.

Versión 1

Base de Datos: ClientesyVendedores

clientes
<u>numcliente</u> nombrecliente direccion numvendedor nombrevendedor

clientes (numcliente, nombrecliente, direccion, numvendedor, nombrevendedor)

Tabla: clientes

<u>numcliente</u>	nombrecliente	direccion	numvendedor	nombrevendedor
102	López, María	Alameda, 7	40	Rodríguez, Marcos
145	Pérez Carlos	Huelgas, 15	10	Gallego, Berta
230	García, Diego	Recoletas, 8	12	López, Juan
340	López, Javier	Vendimia, 8	2	Alonso, Victoria
57	Pérez, José	Atocha, 15	12	López, Juan
34	Mora, Marta	Valencia, 20	2	Alonso, Victoria

Tal y como puede observarse en la inserción inicial de datos mostrada, se considera que en un vendedor puede gestionar varios clientes y que cada cliente solo puede ser atendido por un vendedor.

El diseño propuesto para la tabla clientes está en 1FN porque los valores de todas las celdas son atómicos. Además también está en 2FN, ya que como la clave primaria se compone solamente de un campo, no puede haber problemas de que haya campos que dependan de una parte de la clave.

Sin embargo, no está en 3FN, ya que hay campos no clave que dependen de otro campo no clave, en concreto el nombre del vendedor depende del número del vendedor.

La solución propuesta es apartar todos los campos no clave que dependen de otro campo no clave y situarlos en otra tabla que tenga ese campo como clave.

Versión 2

Base de Datos: ClientesyVendedores

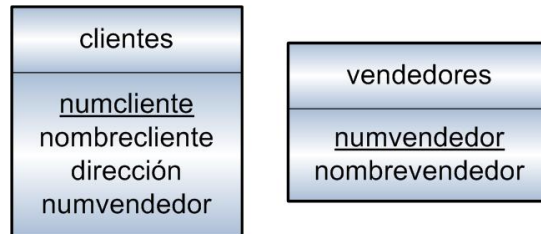


Tabla: vendedores

<u>numvendedor</u>	nombrevendedor
40	Rodríguez, Marcos
10	Gallego, Berta
12	López, Juan
2	Alonso, Victoria

Tabla: clientes

<u>numcliente</u>	nombrecliente	direccion	numvendedor
102	López, María	Alameda, 7	40
145	Pérez Carlos	Huelgas, 15	10
230	García, Diego	Recoletas, 8	12
340	López, Javier	Vendimia, 8	2
57	Pérez, José	Atocha, 15	12
34	Mora, Marta	Valencia, 20	2

El diseño propuesto, formado por dos tablas, sí está en 3NF, ya que cada columna depende directamente de la clave primaria y no hay campos no clave que dependan de otros campos no clave. Nótese que el campo `numvendedor` usado en la tabla `clientes`, actúa como clave ajena puesto que representa en dicha tabla el campo que es la clave principal de la tabla `vendedores`.

Finalmente señalar que, en este ejemplo, al ser la relación entre las tablas de uno – vendedor – a varios – clientes –, es suficiente con un diseño formado por dos tablas.

3.4.5 Ejemplo líneas de metro

Por último, vamos a ver un ejemplo más extenso y complejo y en consecuencia, también más cercano a la realidad. El objetivo es realizar el diseño de una BD para gestionar las líneas de metro de una ciudad. Las especificaciones son las siguientes:

- Cada **línea** tiene dos identificadores: un número (línea 1, línea 2, etc.) y un color (no hay dos líneas con el mismo número, ni dos con el mismo color). De cada línea se quiere almacenar la hora de salida del primer tren y la del último. Además, cada línea está compuesta por un conjunto de estaciones en un orden determinado.
- Las **estaciones** tienen un código que las distingue de las demás. De ellas se almacena su nombre y los horarios de apertura y cierre. Una estación pertenece al menos a una línea, aunque puede pertenecer a varias.
- Cada estación tiene asociado uno o varios **accesos** al exterior. Cada uno de dichos accesos pertenece a una única estación. Los accesos tienen un código asociado. Además, de cada acceso se almacena la calle y el número de portal más cercano al mismo.
- Cada línea tiene asignado un conjunto de **trenes**. No puede suceder que un tren esté asignado a varias líneas, pero sí que no esté asignado a ninguna (cuando está en reparación). Cada tren tiene un número diferente al resto de trenes. Se necesita guardar la fecha de compra, el modelo y la capacidad en número de vagones.
- En algunas estaciones hay **cocheras** para aparcar los trenes cuando no están de servicio. Cada tren tiene asignada una de estas cocheras y cada cochera está situada en una estación (en alguna de las estaciones de final de línea hay más de una cochera). De cada cochera se almacena el número de máquinas y el número de vagones que pueden alojar.
- Existen varios **abonos** disponibles para los viajeros. Un abono es válido para varias líneas, siendo también posible que una línea esté asignada a más de un abono. Cada abono tendrá un nombre diferente y la siguiente información: duración, el precio normal, el precio para estudiantes y el precio para jubilados.

Diseño de BD lineasmetro:

El primer paso es **identificar las entidades (y sus atributos)**. También podemos elegir directamente las claves primarias. A primera vista las entidades son: líneas (número, color, hora de salida del primer tren, hora de llegada del último tren), estaciones (código, nombre, hora de apertura, hora de cierre), accesos

(código, calle, número de portal), *trenes* (número, fecha de compra, modelo, número de vagones), *cocheras* (código, número de máquinas, número de vagones) y *abonos* (nombre, duración, precio normal, precio para estudiantes, precio para jubilados). Cada entidad dará lugar a una tabla, siendo sus atributos los campos de la misma.

Una vez hecho esto, **establecemos las relaciones entre entidades**. Siempre es útil identificar primero las relaciones de varios a varios, que como hemos dicho, dan lugar a una tabla extra. En el caso que nos ocupa, hay **dos relaciones varios a varios**: entre las entidades *líneas* y *estaciones*, y entre las entidades *líneas* y *abonos*. Crearemos por lo tanto dos tablas más para satisfacer dichas relaciones: *lineas_estaciones* y *abonos_lineas*.

La tabla *lineas_estaciones* está compuesta por las claves primarias de las tablas *lineas* y *estaciones*, siendo la combinación de ambas la clave primaria de *lineas_estaciones* y además tiene un campo extra, el orden, que no se puede poner en ninguna otra tabla, ya que depende de la clave primaria de ambas tablas.

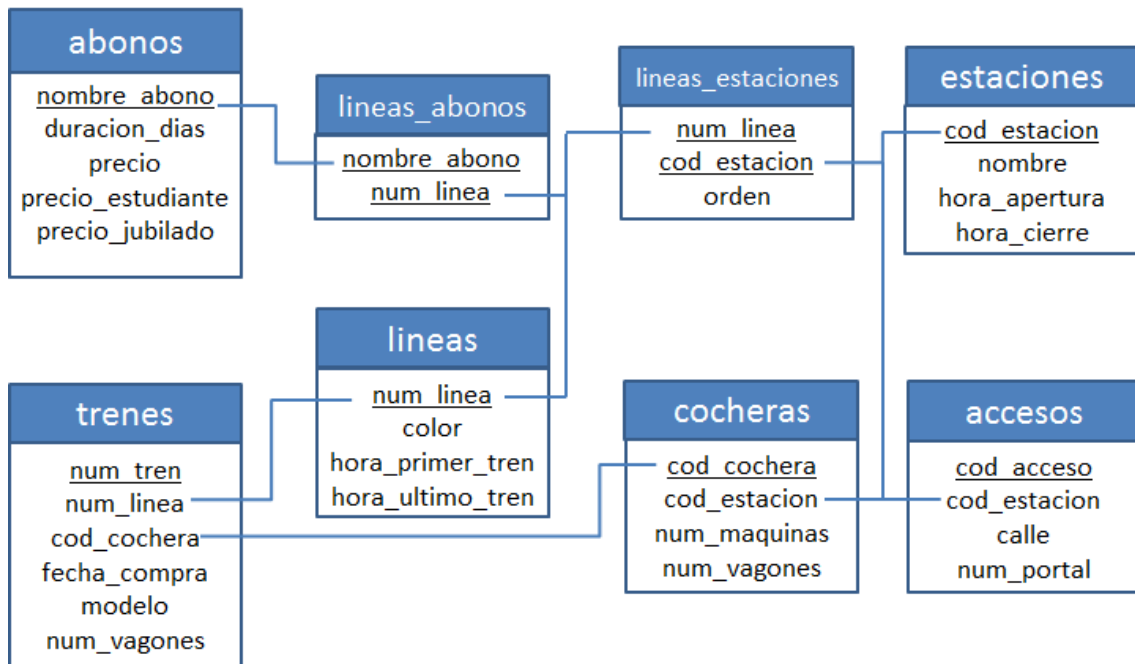
Seguimos identificando el resto de relaciones. La tabla *lineas*, además de las dos relaciones con las tablas *abonos* y *estaciones*, también está relacionada con la tabla *trenes* mediante una relación uno a varios (un tren puede estar asignado a cero o a una línea, pero una línea es recorrida por varios trenes), por lo que la tabla *trenes* debe añadir un campo (la clave primaria de la tabla *lineas*).

Siguiendo con la tabla *trenes*, ésta también está relacionada con la tabla *cocheras* mediante una relación uno a varios (una cochera contiene varios trenes, pero un tren sólo puede estar asignado a una cochera), por lo que debemos añadir un campo más a *trenes* (la clave primaria de la tabla *cocheras*).

La tabla *cocheras* está, a su vez, relacionada con la tabla *estaciones* mediante una relación uno a varios (en una estación puede haber más de una cochera, pero una cochera sólo puede estar situada en una estación), por lo que también añadiremos un campo a *cocheras* (la clave primaria de la tabla *estaciones*).

Por último, la tabla *accesos* sólo está relacionada con la tabla *estaciones* mediante una relación uno a varios (en una estación puede haber más de un acceso, pero un acceso sólo puede estar situado en una estación). Añadimos un campo a *accesos* (la clave primaria de la tabla *estaciones*).

A continuación, se puede observar el esquema final de la base de datos:



4 LENGUAJE SQL

4.1 INTRODUCCIÓN

SQL (Structured Query Language) es un lenguaje de programación diseñado para almacenar, manipular y recuperar datos almacenados en bases de datos relacionales. Es un **lenguaje declarativo**, sólo hay que indicar qué se quiere hacer. SQL apareció por primera vez en 1974, cuando Codd (IBM) desarrolló el primer prototipo de una base de datos relacional.

Existe un SQL estándar, pero el SQL que implementan las principales SGBDR³⁵ no son exactamente iguales. Esto se debe a dos motivos:

1. el estándar SQL es bastante complejo, y no es práctico implementarlo de forma completa, y
2. cada proveedor de base de datos quiere diferenciar su producto del resto.

Los lenguajes de consulta tienen dos partes fundamentales:

- **DDL (Data Definition Language)**: usado para creación, modificación, destrucción de tablas, etc. de la base de datos.

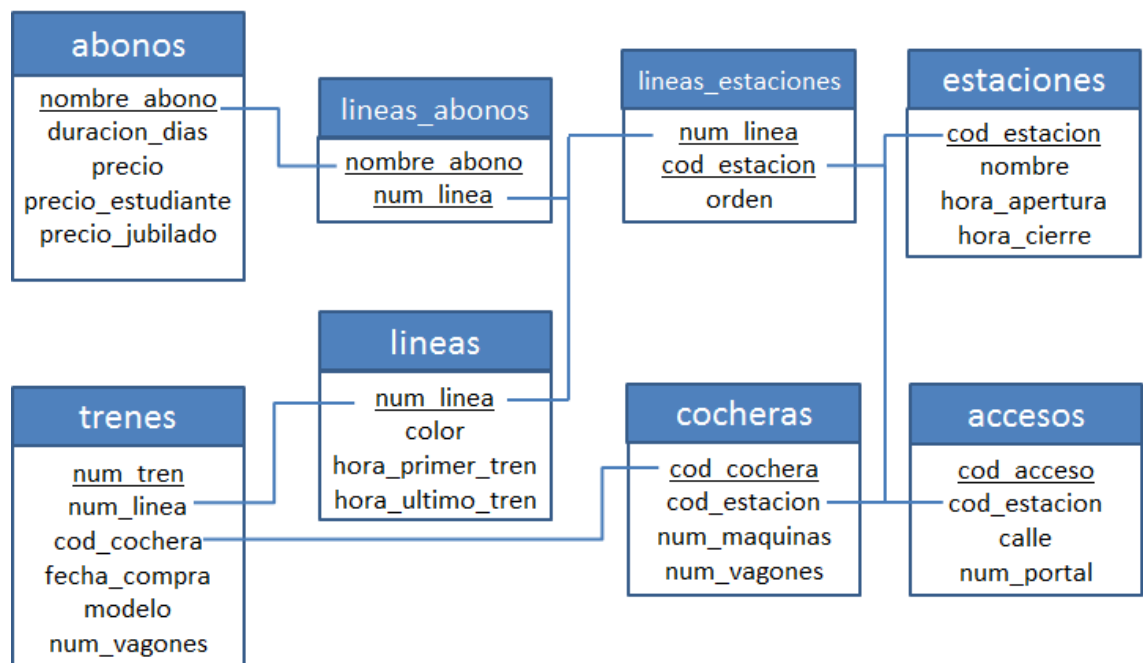
³⁵ En inglés, RDBMS (*Relational Database Management System*).

- **DML (Data Manipulation Language):** usado para obtener, borrar, actualizar, añadir, etc., datos de la base de datos.

4.2 BASE DE DATOS LINEASMETRO

En las siguientes secciones, utilizaremos la base de datos `lineasmetro`, cuyo diseño se ha realizado en la Sección 3.4.5. Basaremos todos los ejemplos en ella: la creación, modificación y destrucción de tablas, el relleno con datos, realizaremos consultas sobre dichos datos, etc.

El esquema se puede ver de nuevo a continuación:



4.3 SQL DATA DEFINITION LANGUAGE (DDL)

El primer paso para obtener una base de datos es su definición: cómo se denomina, qué tablas tiene y cuáles son sus propiedades, etc. Como ya hemos visto, la parte de SQL responsable de dichas tareas es el DDL.

Las sentencias que componen el DDL son las siguientes:

CREATE	Permite crear bases de datos y tablas
ALTER	Permite modificar bases de datos y tablas

DROP Permite borrar bases de datos y tablas

4.3.1 Sentencia CREATE

Se comienza definiendo la base de datos con la **sentencia CREATE DATABASE**:

```
CREATE DATABASE basedatos;
```

Una vez creada la base de datos se puede comenzar a crear tablas. Para ello se utiliza la **sentencia CREATE TABLE**. A continuación se puede ver la estructura que debe tener esta sentencia:

```
CREATE TABLE tabla  
(definición de la lista de campos);
```

```
CREATE TABLE nombre_tabla  
(nombre_campo1      tipo_datos_1      [restricciones columna1],  
 nombre_campo2      tipo_datos_2      [restricciones columna2],  
 nombre_campo3      tipo_datos_3      [restricciones columna3],  
 nombre_campo4      tipo_datos_4      [restricciones columna4],  
 ...  
);
```

4.3.1.1 Definición de campos

Algunas consideraciones a tener en cuenta:

El **nombre** elegido para la tabla y para sus campos debe ser un nombre válido para el SGBD utilizado. En general, dichos nombres sólo pueden contener caracteres alfanuméricos, deben comenzar con un carácter alfabético o con el carácter de subrayado (_) y su longitud no debe sobrepasar 24 caracteres.

Cada campo debe llevar un **tipo de datos** asociado. Diferentes gestores de bases de datos soportan diferentes tipos de datos. Dado que MySQL es el gestor que se utiliza en esta asignatura, nos centraremos en los tipos que soporta éste.

En MySQL existen tres tipos de datos básicos: **numérico, fecha y hora y cadena**. Cada categoría incluye un gran número de tipos y dispone de varios

tamaños de almacenamiento. A la hora de determinar qué tipo es el más adecuado para un campo, se debe escoger el tipo más pequeño en el que encajen los datos, es decir, siempre considerando el tamaño máximo que pueden tomar los valores de dicho campo para evitar que se trunquen o que se produzcan errores.

1. **Tipos numéricos:** números enteros, en coma flotante o en coma fija. Algunos tipos numéricos pueden llevar asociados los siguientes **atributos:**

- UNSIGNED: Normalmente el rango de un valor de tipo numérico va desde un número negativo hasta un número positivo. Al añadir la opción UNSIGNED a un **entero**, **ambos extremos del rango se desplazan** comenzando en cero y acabando en un número positivo mayor. Por ejemplo, tomando el rango -127...128, su versión UNSIGNED sería 0...255. Sin embargo, si se añade la opción a un número de **coma flotante**, aunque el extremo inferior del rango también se desplaza hasta el cero, **el extremo superior no varía**.
- ZEROFILL: rellena con ceros a la izquierda. Si se especifica en una columna, se le asigna automáticamente el atributo UNSIGNED.
- AUTO_INCREMENT: al añadir este atributo a un campo de tipo entero, los **valores del mismo se generan automáticamente de manera secuencial**. Hay que tener en cuenta, que si se eliminan registros, la cuenta no recupera los valores que han desaparecido, sino que continúa como si no hubiera pasado nada. El único modo de reiniciar los valores consiste en eliminar la tabla al completo y volver a crearla de nuevo.

Tabla 7. Tipos de números enteros

NÚMEROS ENTEROS			
Tipo	Rango (con signo / sin signo)	Espacio de almacenamiento (bytes)	¿Qué permite almacenar?
TINYINT [(M) ³⁶]	-127..128 / 0..255	1	Enteros muy pequeños
SMALLINT [(M)]	-32768..32767 / 0..65535	2	Enteros pequeños
MEDIUMINT [(M)]	-8388608..8388607 / 0..16777215	3	Enteros de tamaño medio
INT/INTEGER [(M)]	$-2^{31}..2^{31}-1$ / 0..16777215	4	Enteros normales
BIGINT [(M)]	$-2^{63}..2^{63}-1$ / 0.. $2^{64}-1$	8	Enteros grandes
BIT [(M) ³⁷]	0.. 2^M-1	(M+7)/8 (aprox.)	Para representar campos de bits ³⁸

³⁶ M en relación a enteros especifica el ancho de visualización, que es independiente del rango de valores permitido. Por ejemplo, el rango de valores de TINYINT(2) sigue siendo de -127 a 128. Además, si se añade un valor de más de dos dígitos, se visualizará de forma correcta. Especificar un ancho de visualización sólo tiene sentido usado en conjunción con el atributo ZEROFILL, ya que por defecto el ancho de visualización se rellena con espacios.

³⁷ En combinación con BIT, M puede tomar valores del 1 al 64. En este caso, M está directamente relacionado con el número de valores permitido. Por ejemplo, BIT(8) permite almacenar valores del 0 al 255. Si se introducen valores mayores, estos no se visualizarán adecuadamente.

³⁸ Los valores se pueden introducir en decimal o en binario (ej.: b'0101'), pero se devuelven como valores binarios. Para visualizarlos correctamente se debe añadir el 0 (nombre_campo+0) si se quieren ver en decimal o utilizar a mayores alguna función de conversión (BIN(nombre_campo+0), HEX(nombre_campo+0), OCT(nombre_campo+0)) si se quieren ver en binario, hexadecimal, etc.

Por ejemplo, utilizando una tabla t con un único campo b de tipo BIT(8), en la que se han insertado tres registros con los valores binarios 11111111, 1010 y 0101, visualizamos estos valores de la siguiente manera:

BOOL/BOOLEAN (equivale a TINYINT(1))	0 ó 1	1	0 se interpreta como falso y 1 como verdadero
---	-------	---	---

Antes de ver los tipos asociados a números en coma flotante o en coma fija vamos a recordar la diferencia entre ambos. Los dos hacen referencia a números con decimales, pero **los tipos de coma flotante guardan valores aproximados, mientras que los de coma fija guardan valores exactos**. Así, cuando la exactitud sea importante (ej.: en bases de datos bancarias), se debe utilizar con tipos -de coma fija.

Tabla 8. Tipos de números en coma flotante

NÚMEROS EN COMA FLOTANTE			
Tipo	Rango	Espacio de almacen. (bytes)	¿Qué permite almacenar?
FLOAT (PRECISION)	Depende de la precisión	<u>PRECISIÓN SIMPLE</u> Si $0 \leq \text{precisión} \leq 24$, 4 bytes <u>PRECISIÓN DOBLE</u> Si $24 < \text{precisión} \leq 53$, 8 bytes	Número en coma flotante de precisión simple o doble.
FLOAT [(M, D) ³⁹]	$\pm 1.175494351E-38$ $\pm 3.402823466E+38$	4	Número en coma flotante de precisión simple. Equivale a FLOAT (4) .

```
mysql> SELECT b+0, BIN(b+0), OCT(b+0), HEX(b+0) FROM t;
+-----+-----+-----+-----+
| b+0 | BIN(b+0) | OCT(b+0) | HEX(b+0) |
+-----+-----+-----+-----+
| 255 | 11111111 | 377      | FF       |
| 10  | 1010     | 12       | A        |
| 5   | 101      | 5        | 5        |
+-----+-----+-----+-----+
```

[Fuente: Manual de Referencia MySQL 5.5, Sección 9.1.6 Bit-Field Literals]

³⁹ Para números en coma flotante la M especifica el ancho de visualización (incluyendo el número decimales) y la D, el número de decimales concreto que se desea visualizar.

DOUBLE [PRECISION] / REAL [(M, D)]	±1.7976931348623157E+308 ±2.2250738585072014E+308	8	Número en coma flotante de precisión doble. Equivale a FLOAT (8) .
--	--	---	--

Tabla 9. Tipos de números en coma fija

NÚMEROS EN COMA FIJA			
Tipo	Rango	Espacio de almacenamiento (bytes)	¿Qué permite almacenar?
DECIMAL/NUMERIC [(M, D)]	Varía	Varía	Número de precisión doble como cadena. -Se usa para valores en los que la exactitud es necesaria.

2. **Tipos de fecha y hora:** como se verá a continuación, el formato estándar de las fechas es AAAA-MM-DD. Sin embargo, MySQL también permite introducir el año con formato de dos dígitos, que convertirá de la siguiente manera: dígitos en el rango 70-99 se almacena como 1970-1999 y en el rango 00-69, como 2000-2069.

Tabla 10. Tipos de fecha y hora

FECHA/HORA			
Tipo	Rango	Espacio de almacen. (bytes)	¿Qué permite almacenar?
DATE	1000-01-01 9999-12-31	3	Una fecha. Se mostrará con formato AAAA-MM-DD.
TIME	-838:59:59 838:59:59 ⁴⁰	3	Una hora. Se mostrará con formato (H) HH:MM:SS.
DATETIME	1000-01-01 00:00:00 9999-12-31 23:59:59	8	Una fecha y una hora. Se mostrarán con el formato YYYY-MM-DD HH:MM:SS

⁴⁰ El rango es mayor del necesario para representar una hora del día, porque también se puede usar para representar intervalos de tiempo entre eventos.

TIMESTAMP	1970-01-01 00:00:00 2038-01-19 03:14:07	4	Una marca de tiempo, que resulta útil para la generación de informes.
YEAR [(M) ⁴¹]	70-69 (2 díg.) (1970-2069) 1901-2155 (4 díg.)	1	Un año. Permite visualizar el año con 2 ó 4 dígitos. Esto no tiene nada que ver con cómo se deben introducir. ⁴²

3. **Tipos de cadena:** una cadena puede contener letras, números y caracteres especiales. Estos tipos se pueden clasificar en **tres subgrupos:** los tipos que se tratan como **cadena no binarias** [Tabla 11], los tipos **SET y ENUM** [Tabla 12], que siguen unas reglas especiales y los tipos **BLOB (Binary Large Object)** [Tabla 13], que se tratan como cadenas binarias y pueden contener imágenes, sonidos, etc.

Tabla 11. Tipos de cadenas

CADENAS			
Tipo	Rango	Espacio de almacenamiento (bytes)	¿Qué permite almacenar?
CHAR (M)	0..255 caracteres	M × w ⁴³	Cadena de caracteres de longitud fija M . Los campos de tipo CHAR se rellenarán con espacios hasta cubrir la anchura máxima, independientemente del tamaño de los datos, por lo que el espacio de almacenamiento depende solamente de la longitud establecida.

⁴¹ En este caso, M sólo puede tomar los valores 2 ó 4.

⁴² Para aclarar este punto: considerando un campo de tipo YEAR(2), se puede introducir el año como 2016. MySQL lo almacena correctamente y lo muestra como 16. De igual manera, para un campo de tipo YEAR(4), se puede introducir el año como 16 y se mostrará 2016.

⁴³ Donde w es el número de bytes de almacenamiento requeridos por el carácter de máxima longitud, que depende del formato de codificación de caracteres (ej. en UTF-8 un carácter puede necesitar hasta 3 bytes).

VARCHAR (M)	0..65535 caracteres	Si $0 \leq L \leq 255$, L+1 Si $256 \leq L \leq 65,535$ L+2	Cadena de caracteres de longitud variable L . El espacio de almacenamiento depende de cada cadena introducida ⁴⁴ . Necesita un byte por carácter más un prefijo de uno o dos bytes ⁴⁵ . La desventaja es que resulta más lento.
TINYTEXT	0..2 ⁸ -1 bytes	L+1	Cadena de caracteres de longitud variable L (donde $L < 2^8$).
TEXT	0..2 ¹⁶ -1 bytes	L+2	Cadena de caracteres de longitud variable L (donde $L < 2^{16}$).
MEDIUMTEXT	0..2 ²⁴ -1 bytes	L+3	Cadena de caracteres de longitud variable L (donde $L < 2^{24}$).
LONGTEXT	0..2 ³² -1 bytes	L+4	Cadena de caracteres de longitud variable L (donde $L < 2^{32}$).

⁴⁴ En la siguiente tabla podemos apreciar la diferencia de espacio de almacenamiento entre CHAR y VARCHAR:

Value	CHAR (4)	Storage Required	VARCHAR (4)	Storage Required
' '	' '	4 bytes	' '	1 byte
'ab'	'ab '	4 bytes	'ab'	3 bytes
'abcd'	'abcd'	4 bytes	'abcd'	5 bytes
'abcdefgh'	'abcd'	4 bytes	'abcd'	5 bytes

[Fuente: Manual de Referencia MySQL 5.5, Sección 9.1.6 Bit-Field Literals]

⁴⁵ Aunque se pueda utilizar VARCHAR para cadenas de más de 255 caracteres, su uso está desaconsejado. Para esos casos se recomienda utilizar TEXT y sus variantes.

Tabla 12. Tipos SET y ENUM

SET y ENUM			
Tipo	Rango (máx. nº de valores permitidos)	Espacio de almacenamiento (bytes)	¿Qué permite almacenar?
ENUM	2 ¹⁶	<p>Si $1 \leq n \leq 255$ valores permitidos 1 byte</p> <p>Si $255 < n \leq 65535$ valores permitidos 2 bytes</p>	<p>Cadena de caracteres elegida de entre un conjunto de valores permitidos, que se definen al crear la tabla.⁴⁶ Dichas cadenas se codifican internamente como números (denominados índice, comienzan en el 1), lo que permite ahorrar espacio.</p> <p>Los campos de este tipo sólo pueden tomar uno de los valores o el valor NULL (su índice también es NULL).</p>
SET	64	(nº valores permitidos + 7)/8 (aprox.)	<p>Cero o más cadenas de caracteres elegidas de entre un conjunto de valores permitidos, que se definen al crear la tabla.⁴⁷ Dichas cadenas se codifican internamente como números.</p> <p>Los campos de este tipo pueden tomar uno o varios⁴⁸ de los valores o el valor NULL.</p>

⁴⁶ nombre_columna ENUM('valor1', 'valor2', 'valor3', 'valor4', ...)

⁴⁷ nombre_columna SET('valor1', 'valor2', 'valor3', 'valor4', ...)

⁴⁸ Por ejemplo, una columna definida como SET('uno', 'dos', 'tres') puede tomar los siguientes valores: NULL, '', 'uno', 'dos', 'tres', 'uno,dos', 'uno,tres', 'dos,tres', 'uno,dos,tres'. Como se puede apreciar, aquellos valores compuestos por varios miembros del SET se separan mediante comas, por lo que al definir los valores individuales no se deben incluir comas.

Tabla 13. Tipos BLOB

BLOBs			
Tipo	Rango	Espacio de almacenamiento (bytes)	¿Qué permite almacenar?
TINYBLOB	0..2 ⁸ -1	L+1	Cadena de caracteres de longitud variable L (donde L < 2 ⁸).
BLOB	0..2 ¹⁶ -1	L+2	Cadena de caracteres de longitud variable L (donde L < 2 ¹⁶).
MEDIUMBLOB	0..2 ²⁴ -1	L+3	Cadena de caracteres de longitud variable L (donde L < 2 ²⁴).
LOB	0..2 ³² -1	L+4	Cadena de caracteres de longitud variable L (donde L < 2 ³²).

Una vez especificado el nombre del campo y el tipo de datos que soporta, se pueden imponer ciertas **restricciones**. Las más comunes son las siguientes:

- NOT NULL: el campo no puede tener valores nulos. Se usa para los campos obligatorios, aquellos que siempre deben contener un valor. El cero o un espacio también se consideran valores.
- PRIMARY KEY: el campo es la clave primaria. Si la clave primaria es simple se coloca al final de la línea de definición del campo. Si es múltiple, en la última sentencia seguida de unos paréntesis que contendrán aquellos campos que formen parte de la clave primaria⁴⁹.
- UNIQUE: el campo no puede tener valores repetidos. Es una clave alternativa.

4.3.1.2 Creación de tablas a partir de registros existentes

Además de crear una tabla desde cero, también **se pueden crear tablas en las que se introduzcan registros de otras tablas ya existentes** utilizando **AS SELECT**.

CREATE TABLE tabla (definición de la lista de campos)

⁴⁹ PRIMARY KEY(nombre_campo1, nombre_campo2)

```
AS SELECT lista de campos  
FROM lista de tablas  
[WHERE expresión condicional]50;
```

Considerando una tabla denominada `tabla1` que constara solo de 3 campos (`campo1`, `campo2` y `campo3`), la siguiente sentencia permitiría crear una tabla `tabla2` que fuese una copia exacta de la tabla `tabla1` (incluyendo los valores de `tabla1`):

```
CREATE TABLE tabla2 (campo1, campo2, campo3)  
AS SELECT campo1, campo2, campo3 FROM tabla1;
```

Como `tabla2` tiene las mismas columnas y en el mismo orden que `tabla1`, la lista de campos de la primera línea podría haberse omitido:

```
CREATE TABLE tabla2  
AS SELECT campo1, campo2, campo3  
FROM tabla1;
```

Si se hubiese querido que `tabla2` sólo tuviese un subconjunto de los registros de `tabla1` entonces se hubiese tenido que usar la cláusula `WHERE` para especificar las condiciones apropiadas.

4.3.1.3 Creación `lineasmetro`

Una vez establecidas las reglas de la sentencia `CREATE`, podemos comenzar a desarrollar la base de datos `lineasmetro`. En nuestro caso, queremos definirla desde cero:

```
CREATE DATABASE lineasmetro;
```

Como ya se ha visto, las tablas que componen nuestra base de datos son `lineas`⁵¹, `estaciones`, `lineas_estaciones`, `accesos`, `cocheras`, `trenes`,

⁵⁰ Para conocer en más detalle el uso y la estructura de la cláusula `WHERE`, Sección 4.4.2

⁵¹ Es recomendable no introducir tildes, ni en los nombres de tablas y campos, ni en los datos que se introducirán después, ya que ciertas configuraciones no los soportan y las palabras aparecen entrecortadas.

abonos y abonos_lineas. A continuación vamos a ver la definición de algunas de ellas⁵²:

```
CREATE TABLE lineas
(
  num_linea int NOT NULL AUTO_INCREMENT PRIMARY KEY,
  color char(10) NOT NULL,
  hora_primer_tren time NOT NULL,
  hora_ultimo_tren time NOT NULL
);
```

```
CREATE TABLE lineas_estaciones
(
  num_linea int NOT NULL,
  cod_estacion char(6) NOT NULL,
  orden int NOT NULL,
  PRIMARY KEY(num_linea, cod_estacion)
);
```

En la tabla trenes, aparece el único campo que no tiene asociado la opción `NOT NULL`, `num_linea`. Esto permite dejar los trenes en reparación sin asociar a una línea determinada. En esta sentencia también se puede ver que el número de tren, a pesar de ser un número, es de tipo `char`, ya que no se opera con él.

```
CREATE TABLE trenes
(
  num_tren char(4) NOT NULL PRIMARY KEY,
  num_linea int,
  cod_cochera char(4) NOT NULL,
  fecha_compra date NOT NULL,
  modelo char(10) NOT NULL,
  num_vagones int NOT NULL
);
```

⁵² Las sentencias `CREATE` para todas las tablas se encuentran en el Anexo 1.

```
CREATE TABLE abonos
(
  nombre_abono char(10) NOT NULL PRIMARY KEY,
  duracion_dias int NOT NULL,
  precio float(8,2) NOT NULL,
  precio_estudiante float(8,2) NOT NULL,
  precio_jubilado float(8,2) NOT NULL,
);
```

Tras la ejecución de todas las sentencias de creación de tablas, podemos comprobar si han sido añadidas (utilizando el comando `SHOW TABLES`):

```
mysql> show tables;
+-----+
| Tables_in_lineasmetro |
+-----+
| abonos                 |
| abonos_lineas         |
| accesos               |
| cocheras              |
| estaciones            |
| lineas                |
| lineas_estaciones     |
| trenes                |
+-----+
8 rows in set (0.00 sec)
```

Para comprobar si han sido creadas correctamente con los atributos especificados, también es posible obtener la definición completa de cada tabla. Por ejemplo, para la tabla `trenes` (utilizando el comando `DESCRIBE`):

```
mysql> describe trenes;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| num_tren   | int(11)   | NO   | PRI | NULL    |       |
| num_linea  | int(11)   | YES  |     | NULL    |       |
| cod_cochera | char(4)   | NO   |     | NULL    |       |
| fecha_compra | date      | NO   |     | NULL    |       |
| modelo     | char(10)  | NO   |     | NULL    |       |
| num_vagones | int(11)   | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
6 rows in set (0.10 sec)
```

O la tabla `abonos_lineas` (utilizando el comando `SHOW COLUMNS`):

```
mysql> show columns from abonos_lineas;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nombre_abono | char(10)  | NO   | PRI | NULL    |      |
| num_linea    | int(11)   | NO   | PRI | NULL    |      |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.39 sec)
```

4.3.2 Sentencia ALTER

Una vez definidas las tablas, puede que queramos añadir/modificar/eliminar algún campo, para lo que se usa la **sentencia ALTER TABLE**. En función de la tarea que se quiera realizar, se necesita añadir los siguientes términos:

- ADD: para añadir un campo.
- MODIFY: para modificar la definición de un campo.
- DROP: para borrar un campo.

Vamos a realizar dichas modificaciones sobre la tabla abonos_lineas para ver cómo se usa la sentencia. Primero añadimos un campo:

```
ALTER TABLE abonos_lineas
ADD [COLUMN] prueba int;
```

```
mysql> describe abonos_lineas;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nombre_abono | char(10)  | NO   | PRI | NULL    |      |
| num_linea    | int(11)   | NO   | PRI | NULL    |      |
| prueba       | int(11)   | YES  |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

Una vez hecho esto, procedemos a modificarlo:

```
ALTER TABLE abonos_lineas
MODIFY [COLUMN] prueba char(2) NOT NULL;
```

```
mysql> describe abonos_lineas;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nombre_abono | char(10)  | NO   | PRI | NULL    |      |
| num_linea    | int(11)   | NO   | PRI | NULL    |      |
| prueba       | char(2)   | NO   |     | NULL    |      |
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```

Y por último, lo borramos. De esta manera, la tabla `abonos_lineas` vuelve a su estado inicial:

```
ALTER TABLE abonos_lineas
DROP [COLUMN] prueba;
```

```
mysql> describe abonos_lineas;
+-----+-----+-----+-----+-----+-----+
| Field      | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| nombre_abono | char(10)  | NO   | PRI | NULL    |      |
| num_linea   | int(11)   | NO   | PRI | NULL    |      |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.01 sec)
```

4.3.3 Sentencia DROP

La última sentencia del DDL, **DROP TABLE**, permite eliminar una tabla:

```
DROP TABLE tabla;
```

Utilizando **DROP DATABASE**, se elimina la base de datos al completo:

```
DROP DATABASE basedatos;
```

4.4 SQL DATA MANIPULATION LANGUAGE (DML)

El componente DML de SQL comprende cuatro sentencias básicas:

SELECT	obtener registros de tablas
INSERT	añadir registros nuevos a las tablas
UPDATE	modificar registros de tablas
DELETE	eliminar registros de tablas

4.4.1 Consultas de Modificación: INSERT, UPDATE y DELETE

Una vez definida la base de datos y las tablas que la componen, el siguiente paso es insertar nuevos registros. Una vez insertados, puede que haya que

modificarlos o borrarlos. Para realizar estas tareas SQL proporciona tres sentencias, las denominadas **consultas de modificación**:

INSERT	Inserta nuevos registros en una tabla
UPDATE	Modifica datos en los registros que ya existan en una tabla
DELETE	Borra los registros que existen de una tabla

Si sólo se tienen derechos para ejecutar `SELECT`, el SGBD informará de que no se puede ejecutar ni `INSERT`, ni `UPDATE`, ni `DELETE`.

Si se es el propietario de las tablas, se tienen automáticamente privilegios para realizar todas las operaciones.

4.4.1.1 Sentencia `INSERT`

Ya que nuestra base de datos está vacía, comenzamos con la **sentencia `INSERT`**, que nos va a permitir insertar datos.

El formato estándar de esta sentencia es el siguiente:

```
INSERT INTO tabla [(lista de campos)]  
VALUES (lista de valores);
```

Si los valores proporcionados, concuerdan en orden y número con los campos de la tabla, la lista de campos puede omitirse. Esto se ha hecho en el siguiente ejemplo, en el que la tabla solo tiene tres campos; los dos primeros de tipo cadena y el último de tipo numérico:

```
INSERT INTO tabla  
VALUES ('valor1', 'valor2', 40);
```

Teniendo en cuenta estas indicaciones, rellenamos las diferentes tablas de `lineasmetro` con los datos con los que se trabajará más adelante⁵³, siempre teniendo en cuenta la definición de los campos que se llevó a cabo en la sección anterior.

⁵³ Los datos se han escogido utilizando como modelo el metro de la Comunidad de Madrid para intentar ajustarse lo más posible a una línea de metro real. Como es obvio, no todos los datos son exactos y la cantidad de líneas, estaciones, etc. se ha reducido para que las tablas sean lo más manejable posibles.

La primera tabla que vamos a rellenar es la correspondiente a las líneas de la red metro, que tiene la peculiaridad de poseer un campo (`num_linea`) con el atributo `AUTO_INCREMENT`. Como queremos que la numeración empiece en 1, que es el comportamiento por defecto⁵⁴, sólo tenemos que añadir explícitamente los valores de los otros tres campos.

```
INSERT INTO lineas (color, hora_primer_tren, hora_ultimo_tren)
VALUES ('azul', '06:05:00', '02:00:00'),
      ('rojo', '06:05:00', '02:00:00'),
      ('amarillo', '06:05:00', '02:00:00'),
      ('marron', '07:00:00', '01:00:00'),
      ('verde', '07:00:00', '01:00:00');
```

⁵⁴ Para entender mejor el funcionamiento del atributo `AUTO_INCREMENT`, vamos a utilizar un pequeño ejemplo. Tomamos una tabla denominada `tabla1`, que consta de dos campos, `secuencia` y `numero`, donde `secuencia` tiene asociado el atributo `AUTO_INCREMENT`. Podemos añadir registros a la tabla especificando sólo los valores de `numero`, `secuencia` tomará entonces valores empezando en 1 (comportamiento por defecto):

```
INSERT INTO tabla1 (numero)
VALUES ('uno'), ('dos'), ('tres');
```

```
+-----+-----+
| secuencia | numero |
+-----+-----+
|          | 1      | uno    |
|          | 2      | dos    |
|          | 3      | tres   |
+-----+-----+
3 rows in set <0.00 sec>
```

Por el contrario, si queremos que los valores de `secuencia` empiecen a partir de otro número, debemos indicarlo explícitamente para el primer registro:

```
INSERT INTO tabla1
VALUES (5, 'cinco');

INSERT INTO tabla1
VALUES ('seis'), ('siete');
```

```
+-----+-----+
| secuencia | numero |
+-----+-----+
|          | 5      | cinco  |
|          | 6      | seis   |
|          | 7      | siete  |
+-----+-----+
3 rows in set <0.00 sec>
```


A la hora de rellenar el resto de tablas, se omiten los campos, ya que los valores proporcionados concuerdan en orden y número con ellos. A continuación se puede ver el comienzo de algunas de estas sentencias⁵⁵:

```
INSERT INTO abonos
VALUES ('Mensual_A', 30, 54.60, 20, 12.30),
        ('Mensual_B', 30, 63.70, 20, 12.30),
        ('Mensual_C', 30, 72, 20, 12.30),
        ('Anual_A', 365, 546, 200, 123),
        ...;
```

```
INSERT INTO accesos
VALUES ('A000', 'ES1001', 'Arturo Soria', '330'),
        ('A001', 'ES1002', 'Bambu', '14'),
        ('A002', 'ES1003', 'Agustin de Foxa', '1'),
        ('A003', 'ES1003', 'Vestibulo Cercanias', '1'),
        ...;
```

```
INSERT INTO trenes
VALUES (100, 1, 'C01', '1994-02-01', 'T2000', 4),
        (101, 1, 'C01', '1994-02-01', 'T2000', 4),
        (102, 1, 'C01', '1996-02-15', 'T2000', 4),
        ...;
```

Aunque no se haya utilizado para `lineasmetro`, la sentencia `INSERT` también puede usarse junto con una búsqueda realizada mediante la sentencia `SELECT` para copiar registros ya existentes de una tabla a otra. La sentencia `SELECT`⁵⁶ reemplaza a la cláusula `VALUES`:

```
INSERT INTO tabla [(lista de campos)]
SELECT lista de campos
FROM lista de tablas
```

⁵⁵ Las sentencias `INSERT` para todas las tablas se encuentran en el Anexo 1.

⁵⁶ Para conocer en más detalle el uso y la estructura de la sentencia `SELECT`, pág.__(Sección____)

```
[WHERE expresión condicional];
```

Sólo los campos especificados de los registros seleccionados por la búsqueda se insertan en la tabla, esto es, aquellos que cumplen la condición especificada en la cláusula `WHERE`. Los campos de la tabla origen y los de la tabla destino deben ser compatibles.

Por ejemplo, tomando una tabla vacía a la que hemos llamado `lineas_reducida`, que consta de los mismos campos que la tabla `linea`, insertamos los registros de las líneas 1 y 2 tomados de dicha tabla.

```
INSERT INTO lineas_reducida
SELECT *
FROM lineas
WHERE num_linea < 3;
```

La tabla resultante es la siguiente:

```
+-----+-----+-----+-----+
| num_linea | color | hora_primer_tren | hora_ultimo_tren |
+-----+-----+-----+-----+
|          1 | azul  | 06:05:00         | 02:00:00         |
|          2 | rojo  | 06:05:00         | 02:00:00         |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

En el siguiente ejemplo utilizamos otra tabla vacía denominada `lineas_reducida2`, que sólo consta de dos campos (`num_linea` y `color`). Esta vez añadimos todos los registros de la tabla `lineas` (por lo que se elimina la cláusula `WHERE`), pero sólo tomamos sus dos primeros campos:

```
INSERT INTO lineas_reducida2
SELECT num_linea, color
FROM lineas;
```

A continuación se puede ver la tabla `lineas_reducida2`:

```

+-----+-----+
| num_linea | color |
+-----+-----+
|          1 | azul |
|          2 | rojo |
|          3 | amarillo |
|          4 | marron |
|          5 | verde |
+-----+-----+
5 rows in set (0.00 sec)

```

4.4.1.2 Sentencia UPDATE

Una vez rellena la base de datos, puede que se necesite modificar ciertos registros. La **sentencia UPDATE** nos permite modificar los valores que nos interesen. Está formada por tres cláusulas (una de ellas opcional):

```

UPDATE tabla
SET lista de columnas y valores
[WHERE expresión condicional];

```

La lista de campos y valores especificada por SET contiene los campos que se desea actualizar con su valor correspondiente de la siguiente manera:

```

campo1 = valor1, campo2 = valor2,...

```

donde `valor1`, `valor2`... pueden ser constantes o expresiones.

Para ver más claramente el caso en el que dicho valores son expresiones complejas utilizaremos la tabla abonos:

```

+-----+-----+-----+-----+-----+
| nombre_abono | duracion_dias | precio | precio_estudiante | precio_jubilado |
+-----+-----+-----+-----+-----+
| Mensual_A    | 30 | 54.60 | 20.00 | 12.30 |
| Mensual_B    | 30 | 63.70 | 20.00 | 12.30 |
| Mensual_C    | 30 | 72.00 | 20.00 | 12.30 |
| Anual_A      | 365 | 546.00 | 200.00 | 123.00 |
| Anual_B      | 365 | 637.00 | 200.00 | 123.00 |
| Anual_C      | 365 | 720.00 | 200.00 | 123.00 |
| Turista_1    | 1 | 8.40 | 8.40 | 8.40 |
| Turista_2    | 2 | 14.20 | 14.20 | 14.20 |
| Turista_3    | 3 | 18.40 | 18.40 | 18.40 |
| Turista_5    | 5 | 26.80 | 26.80 | 26.80 |
| Turista_7    | 7 | 35.40 | 35.40 | 35.40 |
+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)

```

Es posible, e incluso muy probable, que en el futuro los precios aumenten. Suponiendo que la subida es del 5% y que afecta tanto al precio normal, como al precio de estudiantes y jubilados⁵⁷, se actualiza la tabla utilizando UPDATE:

```
UPDATE abonos
SET precio = precio*1.05, precio_estudiante = precio_estudiante*1.05, precio_jubilado =
precio_jubilado*1.05;
```

La tabla queda de la siguiente manera:

nombre_abono	duracion_dias	precio	precio_estudiante	precio_jubilado
Mensual_A	30	57.33	21.00	12.92
Mensual_B	30	66.89	21.00	12.92
Mensual_C	30	75.60	21.00	12.92
Anual_A	365	573.30	210.00	129.15
Anual_B	365	668.85	210.00	129.15
Anual_C	365	756.00	210.00	129.15
Turista_1	1	8.82	8.82	8.82
Turista_2	2	14.91	14.91	14.91
Turista_3	3	19.32	19.32	19.32
Turista_5	5	28.14	28.14	28.14
Turista_7	7	37.17	37.17	37.17

11 rows in set (0.00 sec)

Como se puede observar, no hemos utilizado la cláusula WHERE, porque queríamos actualizar todos los registros de la tabla. En el caso de que la subida sólo afectara a los abonos para turistas, la sentencia se modificaría de la siguiente manera:

```
UPDATE abonos
SET precio = precio + (precio*5)/100, precio_estudiante = precio_estudiante + (precio_estudiante*5)/100,
    precio_jubilado = precio_jubilado + (precio_jubilado*5)/100
WHERE nombre_abono = 'Turista_1' OR nombre_abono = 'Turista_2' OR nombre_abono = 'Turista_3'
OR nombre_abono = 'Turista_5' OR nombre_abono = 'Turista_7';
```

Así, el cambio sólo afecta a los últimos cinco registros de la tabla:

⁵⁷ Para más información sobre las operaciones aritméticas permitidas y su uso, Sección 4.4.2

```

+-----+-----+-----+-----+-----+
| nombre_abono | duracion_dias | precio | precio_estudiante | precio_jubilado |
+-----+-----+-----+-----+-----+
| Mensual_A    | 30            | 54.60  | 20.00             | 12.30           |
| Mensual_B    | 30            | 63.70  | 20.00             | 12.30           |
| Mensual_C    | 30            | 72.00  | 20.00             | 12.30           |
| Anual_A      | 365           | 546.00 | 200.00            | 123.00          |
| Anual_B      | 365           | 637.00 | 200.00            | 123.00          |
| Anual_C      | 365           | 720.00 | 200.00            | 123.00          |
| Turista_1    | 1             | 8.82   | 8.82              | 8.82            |
| Turista_2    | 2             | 14.91  | 14.91             | 14.91           |
| Turista_3    | 3             | 19.32  | 19.32             | 19.32           |
| Turista_5    | 5             | 28.14  | 28.14             | 28.14           |
| Turista_7    | 7             | 37.17  | 37.17             | 37.17           |
+-----+-----+-----+-----+-----+
11 rows in set (0.00 sec)

```

4.4.1.3 Sentencia DELETE

Por último, la **sentencia DELETE** se usa para borrar registros de una tabla. Su estructura es la siguiente:

```

DELETE
FROM tabla
[WHERE expresión condicional];

```

Al igual que en la sentencia UPDATE, la cláusula WHERE es opcional y si se omite, se borran todos los registros de la tabla seleccionada.

Si lo que nos interesa es borrar únicamente ciertos registros, en la cláusula WHERE especificaremos que condiciones deben cumplir éstos. Tomando la tabla cocheras:

```

+-----+-----+-----+-----+
| cod_cochera  | cod_estacion | num_maquinas | num_vagones |
+-----+-----+-----+-----+
| C01          | ES1001       | 50           | 300          |
| C02          | ES1001       | 20           | 120          |
| C03          | ES1005       | 20           | 120          |
| C04          | ES1011       | 35           | 210          |
| C05          | ES1012       | 15           | 90           |
| C06          | ES1019       | 15           | 90           |
| C07          | ES1021       | 50           | 300          |
| C08          | ES1021       | 30           | 180          |
| C09          | ES1033       | 40           | 240          |
| C10         | ES1040       | 70           | 420          |
+-----+-----+-----+-----+
10 rows in set (0.14 sec)

```

Para borrar aquellos registros, cuyas cocheras tengan una capacidad (en número de máquinas) menor de 30 ejecutamos la siguiente sentencia:

```
DELETE
FROM cocheras
WHERE num_maquinas < 30;
```

Se eliminan cuatro registros, dando lugar a la siguiente tabla:

```
+-----+-----+-----+-----+
| cod_cochera | cod_estacion | num_maquinas | num_vagones |
+-----+-----+-----+-----+
| C01         | ES1001       | 50           | 300         |
| C04         | ES1011       | 35           | 210         |
| C07         | ES1021       | 50           | 300         |
| C08         | ES1021       | 30           | 180         |
| C09         | ES1033       | 40           | 240         |
| C10         | ES1040       | 70           | 420         |
+-----+-----+-----+-----+
6 rows in set (0.00 sec)
```

4.4.2 Consulta de Selección: SELECT

Llegados a este punto, lo único que nos queda por hacer es realizar consultas a nuestra base de datos, que cómo ya hemos visto se lleva a cabo mediante la **sentencia SELECT**.

Se puede, por ejemplo, seleccionar **un campo** de una tabla:

```
SELECT color
FROM lineas;
```

```
+-----+
| color |
+-----+
| azul  |
| rojo  |
| amarillo |
| marr  |
| verde |
| gris  |
| naranja |
| rosa  |
| morado |
| negro |
+-----+
10 rows in set (0.00 sec)
```

Seleccionar **varios campos** de una tabla:

```
SELECT num_linea, color
FROM lineas;
```

```
+-----+-----+
| num_linea | color |
+-----+-----+
|          1 | azul |
|          2 | rojo |
|          3 | amarillo |
|          4 | marr |
|          5 | verde |
|          6 | gris |
|          7 | naranja |
|          8 | rosa |
|          9 | morado |
|         10 | negro |
+-----+-----+
10 rows in set (0.00 sec)
```

O seleccionar una tabla **al completo**:

```
SELECT *
FROM lineas;
```

```
+-----+-----+-----+-----+
| num_linea | color | hora_primer_tren | hora_ultimo_tren |
+-----+-----+-----+-----+
|          1 | azul | 06:05:00 | 02:00:00 |
|          2 | rojo | 06:05:00 | 02:00:00 |
|          3 | amarillo | 06:05:00 | 02:00:00 |
|          4 | marr | 07:00:00 | 01:00:00 |
|          5 | verde | 07:00:00 | 01:00:00 |
|          6 | gris | 06:05:00 | 02:00:00 |
|          7 | naranja | 06:05:00 | 02:00:00 |
|          8 | rosa | 06:05:00 | 02:00:00 |
|          9 | morado | 06:05:00 | 02:00:00 |
|         10 | negro | 06:05:00 | 02:00:00 |
+-----+-----+-----+-----+
10 rows in set (0.05 sec)
```

En una tabla, puede haber registros relativos a un campo repetidos. La opción por defecto al utilizar la sentencia `SELECT` muestra todos los registros, lo cual no siempre es deseable. Por ejemplo, si queremos obtener las distintas opciones en términos de duración que ofrecen los abonos, ciertos valores aparecerán repetidos.

```
SELECT duración_dias
FROM abonos;
```

```
+-----+
| duracion_dias |
+-----+
|          30   |
|          30   |
|          30   |
|         365   |
|         365   |
|         365   |
|            1   |
|            2   |
|            3   |
|            5   |
|            7   |
+-----+
11 rows in set (0.00 sec)
```

En este caso, sería preferible visualizar solamente los valores que son distintos. Para ello, se añade la palabra clave `DISTINCT` a la sentencia `SELECT`:

```
SELECT DISTINCT duracion_dias
FROM abonos;
```

```
+-----+
| duracion_dias |
+-----+
|          30   |
|         365   |
|            1   |
|            2   |
|            3   |
|            5   |
|            7   |
+-----+
7 rows in set (0.00 sec)
```

La sentencia `SELECT` permite realizar operaciones aritméticas sobre los campos seleccionados. Los operadores admitidos son los siguientes:

+	Suma
-	Resta
*	Producto
/	División

La multiplicación y la división tienen preferencia sobre la suma y la resta. Los operadores que tienen igual preferencia se evalúan por orden de aparición. El orden de evaluación puede alterarse mediante paréntesis.

Para ejemplificar el uso de estas expresiones aritméticas, vamos a utilizar la tabla `abonos`. Podría ser útil para saber obtener el coste conjunto de los distintos abonos para un grupo de cuatro estudiantes que están planeando un viaje.


```
SELECT 4 * precio_estudiante
FROM abonos;
```

```
+-----+
| 4*precio_estudiante |
+-----+
|          80.00      |
|          80.00      |
|          80.00      |
|         800.00      |
|         800.00      |
|         800.00      |
|          33.60      |
|          56.80      |
|          73.60      |
|         107.20      |
|         141.60      |
+-----+
11 rows in set (0.00 sec)
```

También se puede mostrar uno o varios campos tal como se encuentran en la tabla y aplicar la expresión aritmética a otro/s.

```
SELECT nombre_abono, duracion_dias, 4 * precio_estudiante
FROM abonos;
```

```
+-----+-----+-----+
| nombre_abono | duracion_dias | 4*precio_estudiante |
+-----+-----+-----+
| Mensual_A    |          30   |          80.00      |
| Mensual_B    |          30   |          80.00      |
| Mensual_C    |          30   |          80.00      |
| Anual_A      |         365   |         800.00      |
| Anual_B      |         365   |         800.00      |
| Anual_C      |         365   |         800.00      |
| Turista_1    |            1   |          33.60      |
| Turista_2    |            2   |          56.80      |
| Turista_3    |            3   |          73.60      |
| Turista_5    |            5   |         107.20      |
| Turista_7    |            7   |         141.60      |
+-----+-----+-----+
11 rows in set (0.00 sec)
```

Otra opción es combinar varios campos mediante una operación aritmética. Por ejemplo, para visualizar el precio de los abonos para tres personas (dos jubilados y un adulto).

```
SELECT nombre_abono, duracion_dias, precio + 2*precio_jubilado
FROM abonos;
```

```

+-----+-----+-----+
| nombre_abono | duracion_dias | precio + 2*precio_jubilado |
+-----+-----+-----+
| Mensual_A    | 30            | 79.20                       |
| Mensual_B    | 30            | 88.30                       |
| Mensual_C    | 30            | 96.60                       |
| Anual_A      | 365           | 792.00                      |
| Anual_B      | 365           | 883.00                      |
| Anual_C      | 365           | 966.00                      |
| Turista_1    | 1             | 25.20                       |
| Turista_2    | 2             | 42.60                       |
| Turista_3    | 3             | 55.20                       |
| Turista_5    | 5             | 80.40                       |
| Turista_7    | 7             | 106.20                      |
+-----+-----+-----+
11 rows in set (0.00 sec)

```

El nombre que aparece en el campo resultante, por defecto, es la expresión aritmética tal como se ha definido en la sentencia `SELECT`. Es posible asignar un alias a la nueva columna virtual (éste no debe contener espacios ni estar rodeado por comillas), o bien mediante la palabra clave `AS`, o bien dejando un espacio.

```

SELECT nombre_abono, duracion_dias, precio + 2*precio_jubilado [AS]58
1adulto2jubilados
FROM abonos;

```

```

+-----+-----+-----+
| nombre_abono | duracion_dias | 1adulto2jubilados |
+-----+-----+-----+
| Mensual_A    | 30            | 79.20              |
| Mensual_B    | 30            | 88.30              |
| Mensual_C    | 30            | 96.60              |
| Anual_A      | 365           | 792.00             |
| Anual_B      | 365           | 883.00             |
| Anual_C      | 365           | 966.00             |
| Turista_1    | 1             | 25.20              |
| Turista_2    | 2             | 42.60              |
| Turista_3    | 3             | 55.20              |
| Turista_5    | 5             | 80.40              |
| Turista_7    | 7             | 106.20             |
+-----+-----+-----+
11 rows in set (0.00 sec)

```

SQL proporciona una serie de funciones de agregación que también pueden incluirse en la sentencia `SELECT`. Cada una de estas funciones opera sobre un campo y devuelve un valor único calculado sobre todos los registros seleccionados de dicho campo.

AVG (campo)	Devuelve el valor medio
COUNT (campo)	Devuelve el número de valores no nulos que existen
COUNT (DISTINCT campo)	Devuelve el número de valores distintos

⁵⁸ Los corchetes se utilizan para indicar que lo que se encuentra entre ellos es opcional.

COUNT (*)	Devuelve el número de filas
MAX (campo)	Devuelve el valor máximo
MIN (campo)	Devuelve el valor mínimo
SUM (campo)	Devuelve la suma de todos los valores

Como se puede deducir a partir de las definiciones, las funciones `AVG` y `SUM`, sólo se pueden aplicar a campos cuyos registros contengan valores numéricos. `MAX` y `MIN`, se pueden aplicar tanto a registros de tipo numérico como a registros de tipo fecha.

A continuación se presentan algunos ejemplos utilizando la tabla `trenes`. Las cuatro primeras consultas devuelven respectivamente el número medio, mínimo, máximo y total de vagones de los trenes.

```
SELECT AVG(num_vagones)
FROM trenes;
```

```
+-----+
| avg(num_vagones) |
+-----+
|                5.0714 |
+-----+
1 row in set (0.06 sec)
```

```
SELECT MIN(num_vagones)
FROM trenes;
```

```
+-----+
| min(num_vagones) |
+-----+
|                4 |
+-----+
1 row in set (0.05 sec)
```

```
SELECT MAX(num_vagones)
FROM trenes;
```

```
+-----+
| max(num_vagones) |
+-----+
|                6 |
+-----+
1 row in set (0.00 sec)
```

```
SELECT SUM(num_vagones)
FROM trenes;
```

```
+-----+
| sum(num_vagones) |
+-----+
|                284 |
+-----+
1 row in set (0.00 sec)
```

Para obtener el número de trenes de los que se compone la base de datos, se puede utilizar la función `COUNT`.

```
SELECT COUNT(*)
FROM trenes;
```

```
+-----+
| count(*) |
+-----+
|         56 |
+-----+
1 row in set (0.00 sec)
```

Mediante la siguiente consulta obtenemos el número de modelos de tren distintos.

```
SELECT COUNT(DISTINCT modelo)
FROM trenes;
```

```
+-----+
| count(distinct modelo) |
+-----+
|                        6 |
+-----+
1 row in set (0.00 sec)
```

Según como se definió la base de datos, los trenes que se encuentran averiados, no tienen asignada ninguna línea, sino el valor `null`. Por lo tanto, para obtener el número de trenes en activo, basta con aplicar la función `COUNT` al campo `num_linea`.

```
SELECT COUNT(num_linea) [AS] trenes_activos
FROM trenes;
```

```
+-----+
| trenes_activos |
+-----+
|              54 |
+-----+
1 row in set (0.00 sec)
```

Como estas funciones devuelven un único valor, no pueden incluirse en la sentencia `SELECT` con campos que contengan varios valores procedentes de diferentes registros:

```
SELECT num_linea, AVG(num_vagones)
FROM trenes;
```

```
+-----+-----+
| num_linea | avg(num_vagones) |
+-----+-----+
|          1 |          5.0714   |
+-----+-----+
1 row in set (0.05 sec)
```

El resultado de la consulta es incorrecto, ya que selecciona el primer número de línea y la media de todas las líneas. Más adelante veremos cómo arreglar esto.

La cláusula `WHERE`, como ya hemos visto brevemente en las secciones anteriores, se utiliza para introducir condiciones referentes a los registros. Cuando se utiliza, no se seleccionan una serie de campos de todos los registros, únicamente de aquellos registros que verifiquen las condiciones introducidas por la citada cláusula.

```
SELECT lista de campos
FROM lista de tablas
WHERE expresión condicional;
```

Para crear una expresión condicional se necesitan ciertos operadores. SQL permite utilizar los siguientes **operadores de comparación y operadores lógicos**:

Tabla 14. Operadores lógicos y de comparación

OPERADORES LÓGICOS		OPERADORES DE COMPARACIÓN	
NOT	Negación de condiciones	=	Igual
AND	Conjunción de condiciones	<> (!=)	Distinto
OR	Disyunción de condiciones	>	Mayor que
BETWEEN	Compara con un rango de valores	<	Menor que
IN	Compara con una lista de valores	>=	Mayor o igual que
LIKE	Compara con un patrón de caracteres	<=	Menor o igual que
ANY/SOME	Compara con una lista de valores		

ALL

(Aparecen precediendo a una subconsulta)⁵⁹

Las expresiones en las que intervienen los operadores mencionados se conocen como **expresiones lógicas** porque al evaluarlas, se obtiene como resultado verdadero o falso. Además, los operadores booleanos **AND** y **OR** permiten combinar dichas expresiones lógicas.

Se pueden realizar comparaciones o bien con valores numéricos o bien con valores que contengan uno o más caracteres alfabéticos (**constant**). En este último caso, el valor debe ir entre comillas simples.

A continuación, vamos a realizar varios ejemplos utilizando los operadores mencionados. En el primer ejemplo se obtienen las estaciones correspondientes a la línea 1. Como el valor de comparación es numérico, no se utilizan comillas simples.

```
SELECT cod_estacion
FROM lineas_estaciones
WHERE num_linea = 1;
```

```
+-----+
| cod_estacion |
+-----+
| ES1001       |
| ES1002       |
| ES1003       |
| ES1004       |
| ES1005       |
| ES1006       |
| ES1007       |
| ES1008       |
| ES1009       |
| ES1010       |
| ES1011       |
+-----+
11 rows in set (0.50 sec)
```

Si lo que queremos saber son las líneas a las que pertenece una determinada estación, el valor de comparación será una constante (el código de la estación deseada), por lo tanto se entrecomilla.

```
SELECT num_linea
FROM lineas_estaciones
WHERE cod_estacion = 'ES1009';
```

⁵⁹ Estos operadores no aparecen en los siguientes ejemplos, ya que las subconsultas todavía no se han introducido. Para más información sobre su uso, Sección 4.4.2

```
+-----+
| num_linea |
+-----+
|          1 |
|          2 |
|          3 |
+-----+
3 rows in set (0.00 sec)
```

Los operadores mayor, mayor o igual, menor y menor o igual funcionan todos de manera similar.

```
SELECT nombre_abono, duracion_dias, precio
FROM abonos
WHERE precio < 60;
```

```
+-----+-----+-----+
| nombre_abono | duracion_dias | precio |
+-----+-----+-----+
| Mensual_A    |          30   |  54.60 |
| Turista_1    |           1   |   8.40 |
| Turista_2    |           2   |  14.20 |
| Turista_3    |           3   |  18.40 |
| Turista_5    |           5   |  26.80 |
| Turista_7    |           7   |  35.40 |
+-----+-----+-----+
6 rows in set (0.00 sec)
```

En los siguientes dos ejemplos se utiliza el operador `BETWEEN` para obtener las cocheras con una capacidad de máquinas determinada.

```
SELECT cod_cochera, num_maquinas
FROM cocheras
WHERE num_maquinas BETWEEN 10 AND 30;
```

```
+-----+-----+
| cod_cochera | num_maquinas |
+-----+-----+
| C02        |          20   |
| C03        |          20   |
| C05        |          15   |
| C06        |          15   |
| C08        |          30   |
+-----+-----+
5 rows in set (0.15 sec)
```

```
SELECT cod_cochera, num_maquinas
FROM cocheras
WHERE num_maquinas NOT BETWEEN 10 AND 30;
```

```
+-----+-----+
| cod_cochera | num_maquinas |
+-----+-----+
| C01         |             50 |
| C04         |             35 |
| C07         |             50 |
| C09         |             40 |
| C10         |             70 |
+-----+-----+
5 rows in set (0.00 sec)
```

Además de con rangos de valores numéricos, `BETWEEN` también funciona con rangos de constantes y fechas. A continuación se presentan algunos ejemplos utilizando las tablas estaciones y trenes.

```
SELECT cod_estacion, nombre
FROM estaciones
WHERE nombre BETWEEN 'A' AND 'G';
```

```
+-----+-----+
| cod_estacion | nombre          |
+-----+-----+
| ES1002       | Bambu           |
| ES1003       | Chamartin       |
| ES1005       | Cuatro Caminos |
| ES1007       | Bilbao          |
| ES1010       | Atocha          |
| ES1016       | Banco de Espa  |
| ES1020       | Canal           |
| ES1022       | Delicias        |
| ES1023       | Embajadores    |
| ES1025       | Callao          |
| ES1028       | Alonso Martinez|
| ES1029       | Colon           |
| ES1031       | Diego de Leon   |
| ES1035       | El Carmen      |
| ES1040       | Carabanchel    |
+-----+-----+
15 rows in set (0.13 sec)
```

```
SELECT num_tren, fecha_compra, modelo
FROM trenes
WHERE modelo BETWEEN 'T2' AND 'T4';
```



```
+-----+-----+-----+
| num_tren | fecha_compra | modelo |
+-----+-----+-----+
|      100 | 1994-02-01   | T2000  |
|      101 | 1994-02-01   | T2000  |
|      102 | 1996-02-15   | T2000  |
|      103 | 1996-02-15   | T2000  |
|      104 | 2006-02-04   | T3000  |
|      105 | 2006-02-04   | T3000  |
|      106 | 2006-02-04   | T3000  |
|      107 | 2006-02-04   | T3000  |
|      108 | 2006-02-04   | T3000  |
|      109 | 2006-02-04   | T3000  |
|      110 | 2006-02-04   | T3000  |
|      114 | 1994-02-01   | T2000  |
|      143 | 2006-02-04   | T3000  |
|      144 | 2006-02-04   | T3000  |
|      145 | 2008-02-01   | T3000  |
|      146 | 2008-02-01   | T3000  |
|      147 | 2008-02-01   | T3000  |
|      148 | 2008-02-01   | T3000  |
|      149 | 2008-02-01   | T3000  |
|      150 | 2008-02-01   | T3000  |
|      154 | 2006-02-01   | T3000  |
|      155 | 2006-02-01   | T3000  |
+-----+-----+-----+
22 rows in set (0.00 sec)
```

```
SELECT num_tren, fecha_compra
FROM trenes
WHERE fecha_compra BETWEEN '2004-01-01' AND '2008-01-01';
```

```
+-----+-----+
| num_tren | fecha_compra |
+-----+-----+
|      104 | 2006-02-04   |
|      105 | 2006-02-04   |
|      106 | 2006-02-04   |
|      107 | 2006-02-04   |
|      108 | 2006-02-04   |
|      109 | 2006-02-04   |
|      110 | 2006-02-04   |
|      121 | 2004-02-10   |
|      122 | 2004-02-10   |
|      143 | 2006-02-04   |
|      144 | 2006-02-04   |
|      154 | 2006-02-01   |
|      155 | 2006-02-01   |
+-----+-----+
13 rows in set (0.05 sec)
```

Al contrario que BETWEEN, que permite filtrar en función de un rango de valores, IN permite comparar con unos valores determinados.

```
SELECT cod_cochera, num_maquinas
FROM cocheras
WHERE num_maquinas IN(10,15,30);
```

```

+-----+-----+
| cod_cochera | num_maquinas |
+-----+-----+
| C05         | 15           |
| C06         | 15           |
| C08         | 30           |
+-----+-----+
3 rows in set (0.06 sec)

```

```

SELECT cod_cochera, num_maquinas
FROM cocheras
WHERE num_maquinas NOT IN(10,15,30);

```

```

+-----+-----+
| cod_cochera | num_maquinas |
+-----+-----+
| C01         | 50           |
| C02         | 20           |
| C03         | 20           |
| C04         | 35           |
| C07         | 50           |
| C09         | 40           |
| C10         | 70           |
+-----+-----+
7 rows in set (0.00 sec)

```

Hay que tener en cuenta que los valores con los que se comparan deben coincidir con el contenido de la celda al completo. Con el operador `BETWEEN`, como se ha visto, se pueden definir los extremos (por ejemplo, 'A' y 'G') y la consulta devuelve todos los registros cuyo contenido se encuentre entre ellos. Al hacerlo con el operador `IN` el resultado es el siguiente:

```

SELECT cod_estacion, nombre
FROM estaciones
WHERE nombre IN('A','G');

```

ERROR 1054 (42S22): Unknown column 'A' in 'where clause'

En ocasiones no se conoce el contenido completo de una celda o lo que interesa es una parte común a todos. Para ello, se utiliza el operador `LIKE`, que permite comparar el contenido de un registro con un patrón de caracteres. Este operador reconoce dos caracteres especiales:

Tabla 15. Caracteres especiales para uso en combinación con `LIKE`

%	Representa cualquier secuencia de cero o más caracteres
_	Representa un único carácter cualquiera

El siguiente ejemplo selecciona los accesos que se encuentran en una plaza utilizando este operador.

```
SELECT *
FROM accesos
WHERE calle LIKE 'Plaza%';
```

cod_acceso	cod_estacion	calle	num_portal
A006	ES1004	Plaza de Castilla	1
A031	ES1017	Plaza de Isabel II	1
A036	ES1021	Plaza de Legazpi	1
A043	ES1025	Plaza de Callao	1
A047	ES1026	Plaza de Espa	18
A048	ES1026	Plaza Conde de Toreno	1
A051	ES1028	Plaza Santa Barbara	5
A067	ES1036	Plaza Marques de Salamanca	10

8 rows in set (0.12 sec)

```
SELECT cod_estacion, nombre
FROM estaciones
WHERE cod_estacion LIKE '%1_';
```

cod_estacion	nombre
ES1010	Atocha
ES1011	Menendez Pelayo
ES1012	Ventas
ES1013	Goya
ES1014	Principe de Vergara
ES1015	Retiro
ES1016	Banco de Espa
ES1017	Opera
ES1018	Noviciado
ES1019	Quevedo

10 rows in set (0.08 sec)

Los últimos operadores que quedan por ver son AND y OR, que como ya adelantamos permiten combinar expresiones lógicas. Los operadores menor que, igual, IN, LIKE, etc. se evalúan primero, después se evalúan los operadores AND y después los operadores OR. Se pueden usar paréntesis para alterar el orden de evaluación.

Al ejecutar la siguiente sentencia se obtienen las tres primeras estaciones de las líneas 1 y 2:

```
SELECT *
FROM líneas_estaciones
WHERE num_linea <= 2 AND orden < 4;
```

num_linea	cod_estacion	orden
1	ES1001	1
1	ES1002	2
1	ES1003	3
2	ES1012	1
2	ES1013	2
2	ES1014	3

6 rows in set (0.16 sec)

En el siguiente ejemplo el operador OR nos permite obtener los registros que cumplan alguna de las tres condiciones que se especifican:

```
SELECT num_tren, num_linea, cod_cochera, modelo, num_vagones
FROM trenes
WHERE num_linea = 1
OR cod_cochera = 'C01'
OR modelo LIKE 'T5%';
```

num_tren	num_linea	cod_cochera	modelo	num_vagones
100	1	C01	T2000	4
101	1	C01	T2000	4
102	1	C01	T2000	4
103	1	C01	T2000	4
104	1	C01	T3000	4
105	1	C01	T3000	4
106	1	C01	T3000	4
107	1	C02	T3000	4
108	1	C02	T3000	4
109	1	C03	T3000	4
110	1	C03	T3000	4
111	1	C04	T9000	6
112	1	C04	T9000	6
113	2	C05	T5000	6
131	4	C09	T5000	6
132	4	C09	T5000	6
133	4	C09	T5000	6
134	4	C09	T5000	6
154	NULL	C01	T3000	4

19 rows in set (0.00 sec)

A la hora de combinar AND y OR se debe prestar especial atención al orden de evaluación. Tomando el ejemplo anterior, si quisiéramos que además de cumplir alguna de las tres condiciones especificadas (línea 1, cochera C01, modelo T5000), se cumpliera que el número de vagones fuera menor que 5, añadiríamos dicha condición con el operador AND:

```
SELECT num_tren, num_linea, cod_cochera, modelo, num_vagones
FROM trenes
WHERE num_linea = 1
OR cod_cochera = 'C01'
OR modelo LIKE 'T5%'
AND num_vagones < 5;
```

num_tren	num_linea	cod_cochera	modelo	num_vagones
100	1	C01	T2000	4
101	1	C01	T2000	4
102	1	C01	T2000	4
103	1	C01	T2000	4
104	1	C01	T3000	4
105	1	C01	T3000	4
106	1	C01	T3000	4
107	1	C02	T3000	4
108	1	C02	T3000	4
109	1	C03	T3000	4
110	1	C03	T3000	4
111	1	C04	T9000	6
112	1	C04	T9000	6
154	NULL	C01	T3000	4

14 rows in set (0.00 sec)

¿Por qué no desaparecen los registros relativos a los trenes 111 y 112?

Esto se debe a un error en la formulación de la consulta. Al no utilizar paréntesis, AND sólo combina la condición relativa al número de vagones con la condición relativa al modelo. Por eso sólo desaparecen los registros cuyo modelo es T5000 (éste modelo tiene 6 vagones).

La sintaxis correcta es la siguiente:

```
SELECT num_tren, num_linea, cod_cochera, modelo, num_vagones
FROM trenes
WHERE num_vagones < 5
AND (num_linea = 1 OR cod_cochera = 'C01' OR modelo LIKE 'T5%');
```

num_tren	num_linea	cod_cochera	modelo	num_vagones
100	1	C01	T2000	4
101	1	C01	T2000	4
102	1	C01	T2000	4
103	1	C01	T2000	4
104	1	C01	T3000	4
105	1	C01	T3000	4
106	1	C01	T3000	4
107	1	C02	T3000	4
108	1	C02	T3000	4
109	1	C03	T3000	4
110	1	C03	T3000	4
154	NULL	C01	T3000	4

12 rows in set (0.05 sec)

En la cláusula `WHERE` también se pueden incluir expresiones aritméticas. A continuación, se obtienen aquellos abonos mensuales cuyo precio para tres adultos se encuentra por debajo de los 200€:

```
SELECT nombre_abono, duracion_dias, precio*3
FROM abonos
WHERE precio*3 <= 200
AND nombre_abono LIKE 'Mensual%';
```

nombre_abono	duracion_dias	precio*3
Mensual_A	30	163.80
Mensual_B	30	191.10

2 rows in set (0.09 sec)

El modelo de bases de datos relacional, no presta atención al orden en que los registros aparecen en la tabla. En ocasiones puede resultar útil ordenarlos, para lo que SQL proporciona la **cláusula ORDER BY**.

```
SELECT cod_estacion, nombre
FROM estaciones
WHERE nombre BETWEEN 'A' AND 'G'
ORDER BY nombre;
```

En la captura de la izquierda se encuentra el resultado de la consulta sin la cláusula `ORDER BY` y en la de la derecha con ella, con los registros ordenados de forma ascendente según el campo nombre.

cod_estacion	nombre	cod_estacion	nombre
ES1002	Bambu	ES1028	Alonso Martinez
ES1003	Chamartin	ES1010	Atocha
ES1005	Cuatro Caminos	ES1002	Bambu
ES1007	Bilbao	ES1016	Banco de Espa
ES1010	Atocha	ES1007	Bilbao
ES1016	Banco de Espa	ES1025	Callao
ES1020	Canal	ES1020	Canal
ES1022	Delicias	ES1040	Carabanchel
ES1023	Embajadores	ES1003	Chamartin
ES1025	Callao	ES1029	Colon
ES1028	Alonso Martinez	ES1005	Cuatro Caminos
ES1029	Colon	ES1022	Delicias
ES1031	Diego de Leon	ES1031	Diego de Leon
ES1035	El Carmen	ES1035	El Carmen
ES1040	Carabanchel	ES1023	Embajadores

15 rows in set (0.13 sec) 15 rows in set (0.10 sec)

Sin ORDER BY Con ORDER BY nombre

Como acabamos de ver, la cláusula ORDER BY por defecto ordena los registros de manera ascendente. Si se quieren de manera descendente, basta con añadir la partícula DESC detrás del nombre del campo.

```
SELECT cod_estacion, nombre
FROM estaciones
WHERE nombre BETWEEN 'A' AND 'G'
ORDER BY nombre DESC;
```

cod_estacion	nombre
ES1023	Embajadores
ES1035	El Carmen
ES1031	Diego de Leon
ES1022	Delicias
ES1005	Cuatro Caminos
ES1029	Colon
ES1003	Chamartin
ES1040	Carabanchel
ES1020	Canal
ES1025	Callao
ES1007	Bilbao
ES1016	Banco de Espa
ES1002	Bambu
ES1010	Atocha
ES1028	Alonso Martinez

15 rows in set (0.00 sec)

Se pueden tomar varios campos como criterios para ordenar: el primero proporciona el orden primario, el segundo el orden secundario, etc. En el siguiente ejemplo, los registros se ordenan ascendente en función del campo fecha y para aquellos que coincidan, en función del número de tren.

```
SELECT num_tren, fecha_compra, modelo
FROM trenes
WHERE modelo BETWEEN 'T2' AND 'T4'
ORDER BY fecha_compra, num_tren;
```

num_tren	cod_cochera	fecha_compra	modelo
100	C01	1994-02-01	T2000
101	C01	1994-02-01	T2000
114	C05	1994-02-01	T2000
102	C01	1996-02-15	T2000
103	C01	1996-02-15	T2000
154	C01	2006-02-01	T3000
155	C10	2006-02-01	T3000
104	C01	2006-02-04	T3000
105	C01	2006-02-04	T3000
106	C01	2006-02-04	T3000
107	C02	2006-02-04	T3000
108	C02	2006-02-04	T3000
109	C03	2006-02-04	T3000
110	C03	2006-02-04	T3000
143	C10	2006-02-04	T3000
144	C10	2006-02-04	T3000
145	C10	2008-02-01	T3000
146	C10	2008-02-01	T3000
147	C10	2008-02-01	T3000
148	C10	2008-02-01	T3000
149	C10	2008-02-01	T3000
150	C10	2008-02-01	T3000

22 rows in set (0.00 sec)

En este caso hay que tener cuidado al utilizar la palabra clave `DESC`, ya que si la colocamos al final de la sentencia, sólo afectará al último campo.

```
SELECT num_tren, fecha_compra, modelo
FROM trenes
WHERE modelo BETWEEN 'T2' AND 'T4'
ORDER BY fecha_compra, num_tren DESC;
```


num_tren	cod_cochera	fecha_compra	modelo
114	C05	1994-02-01	T2000
101	C01	1994-02-01	T2000
100	C01	1994-02-01	T2000
103	C01	1996-02-15	T2000
102	C01	1996-02-15	T2000
155	C10	2006-02-01	T3000
154	C01	2006-02-01	T3000
144	C10	2006-02-04	T3000
143	C10	2006-02-04	T3000
110	C03	2006-02-04	T3000
109	C03	2006-02-04	T3000
108	C02	2006-02-04	T3000
107	C02	2006-02-04	T3000
106	C01	2006-02-04	T3000
105	C01	2006-02-04	T3000
104	C01	2006-02-04	T3000
150	C10	2008-02-01	T3000
149	C10	2008-02-01	T3000
148	C10	2008-02-01	T3000
147	C10	2008-02-01	T3000
146	C10	2008-02-01	T3000
145	C10	2008-02-01	T3000

22 rows in set (0.00 sec)

```

SELECT num_tren, fecha_compra, modelo
FROM trenes
WHERE modelo BETWEEN 'T2' AND 'T4'
ORDER BY fecha_compra DESC, num_tren;

```

num_tren	cod_cochera	fecha_compra	modelo
145	C10	2008-02-01	T3000
146	C10	2008-02-01	T3000
147	C10	2008-02-01	T3000
148	C10	2008-02-01	T3000
149	C10	2008-02-01	T3000
150	C10	2008-02-01	T3000
104	C01	2006-02-04	T3000
105	C01	2006-02-04	T3000
106	C01	2006-02-04	T3000
107	C02	2006-02-04	T3000
108	C02	2006-02-04	T3000
109	C03	2006-02-04	T3000
110	C03	2006-02-04	T3000
143	C10	2006-02-04	T3000
144	C10	2006-02-04	T3000
154	C01	2006-02-01	T3000
155	C10	2006-02-01	T3000
102	C01	1996-02-15	T2000
103	C01	1996-02-15	T2000
100	C01	1994-02-01	T2000
101	C01	1994-02-01	T2000
114	C05	1994-02-01	T2000

22 rows in set (0.00 sec)

Los registros también se pueden ordenar en base a un campo calculado. A continuación tomamos como ejemplo el caso de los abonos para un adulto y dos jubilados. Al igual que la primera vez que se usó, se da un alias al campo calculado.

Es importante apuntar que **los alias se pueden utilizar en la cláusula ORDER BY, pero no en la cláusula WHERE.**

```
SELECT nombre_abono, duracion_dias, precio + 2*precio_jubilado [AS]
1adulto2jubilados
FROM abonos
WHERE (precio + 2*precio_jubilado) <= 100
ORDER BY 1adulto2jubilados;
```

nombre_abono	duracion_dias	1adulto2jubilados
Turista_1	1	25.20
Turista_2	2	42.60
Turista_3	3	55.20
Mensual_A	30	79.20
Turista_5	5	80.40
Mensual_B	30	88.30
Mensual_C	30	96.60

7 rows in set (0.25 sec)

En la sección correspondiente a las funciones de agregación, vimos un ejemplo incorrecto y adelantamos que se podía corregir. Para ello se usa La cláusula GROUP BY. Ésta se utiliza en conjunción con las funciones de agregación para agrupar registros en función de los campos utilizados en la sentencia SELECT.

Volviendo al ejemplo incorrecto, al añadir GROUP BY se da sentido a la sentencia y se obtiene la media de número de vagones de tren por cada línea:

```
SELECT num_linea, AVG(num_vagones)
FROM trenes
GROUP BY num_linea;
```

num_linea	AVG(num_vagones)
NULL	4.0000
1	4.3077
2	5.8000
3	6.0000
4	5.5000
5	4.6667

6 rows in set (0.20 sec)

También se puede utilizar en combinación con la función COUNT. Por ejemplo, para saber el número de trenes asignados a cada línea:

```
SELECT num_linea, COUNT(*)
FROM trenes
GROUP BY num_linea;
```

```
+-----+-----+
| num_linea | count(*) |
+-----+-----+
|      NULL |        2 |
|         1 |       13 |
|         2 |       10 |
|         3 |         8 |
|         4 |         8 |
|         5 |       15 |
+-----+-----+
6 rows in set (0.08 sec)
```

En el siguiente ejemplo se obtiene el número de distintos modelos de tren que están asociados a cada línea:

```
SELECT num_linea, COUNT(DISTINCT modelo)
FROM trenes
GROUP BY num_linea;
```

```
+-----+-----+
| num_linea | count(distinct modelo) |
+-----+-----+
|      NULL |            1 |
|         1 |            3 |
|         2 |            3 |
|         3 |            1 |
|         4 |            3 |
|         5 |            4 |
+-----+-----+
6 rows in set (0.04 sec)
```

Utilizando la función SUM se puede obtener el número de vagones asociados a cada línea:

```
SELECT num_linea, SUM(num_vagones)
FROM trenes
GROUP BY num_linea;
```

```
+-----+-----+
| num_linea | sum(num_vagones) |
+-----+-----+
|      NULL |          8 |
|         1 |         56 |
|         2 |         58 |
|         3 |         48 |
|         4 |         44 |
|         5 |         70 |
+-----+-----+
6 rows in set (0.00 sec)
```

GROUP BY no tiene porqué limitarse a un único campo. Para conocer el número de trenes de cada modelo asociados a cada línea, se utiliza esta cláusula con dos campos:

```
SELECT num_linea, modelo, COUNT(*)
FROM trenes
GROUP BY num_linea, modelo;
```

num_linea	modelo	count(*)
NULL	T3000	2
1	T2000	4
1	T3000	7
1	T9000	2
2	T2000	1
2	T5000	1
2	T7000	8
3	T9000	8
4	T5000	4
4	T7000	2
4	T8000	2
5	T3000	8
5	T7000	2
5	T8000	2
5	T9000	3

15 rows in set (0.00 sec)

También se puede utilizar en combinación con WHERE:

```
SELECT num_linea, modelo, COUNT(*)
FROM trenes
WHERE modelo = 'T2000'
OR modelo = 'T7000'
OR modelo = 'T8000'
GROUP BY num_linea, modelo;
```

num_linea	modelo	count(*)
1	T2000	4
2	T2000	1
2	T7000	8
4	T7000	2
4	T8000	2
5	T7000	2
5	T8000	2

7 rows in set (0.09 sec)

De igual modo que la cláusula `WHERE` permite seleccionar registros específicos, la **cláusula `HAVING`** permite seleccionar **grupos** específicos, por lo que sólo se debe utilizar en combinación con `GROUP BY`:

```
SELECT num_linea, AVG(num_vagones)
FROM trenes
GROUP BY num_linea
HAVING AVG(num_vagones) > 5;
```

```
+-----+-----+
| num_linea | AVG(num_vagones) |
+-----+-----+
|          2 |          5.8000 |
|          3 |          6.0000 |
|          4 |          5.5000 |
+-----+-----+
3 rows in set (0.06 sec)
```

`WHERE` y `HAVING` pueden aparecer en la misma sentencia:

```
SELECT num_linea, modelo, COUNT(*)
FROM trenes
WHERE modelo = 'T2000'
OR modelo = 'T7000'
OR modelo = 'T8000'
GROUP BY num_linea, modelo
HAVING COUNT(*) > 2;
```

```
+-----+-----+-----+
| num_linea | modelo | count(*) |
+-----+-----+-----+
|          1 | T2000 |          4 |
|          2 | T7000 |          8 |
+-----+-----+-----+
2 rows in set (0.00 sec)
```

A menudo es necesario realizar consultas que involucran campos de varias tablas a la vez. Como resultado se **combinan** varias tablas. Dichas tablas se especifican mediante la cláusula `FROM` y se unen entre sí mediante uno o más campos comunes. Por lo tanto, es indispensable que entre las tablas involucradas exista alguna **relación** (uno a uno, uno a varios o varios a varios).

Para mostrar la utilidad de este procedimiento, vamos a utilizar un ejemplo que aparece al comienzo de la sección en el que se obtienen las estaciones correspondientes a la línea 1. Lo que la consulta devuelve es el código de las

estaciones. Para saber a qué estaciones se refieren esos códigos habría que realizar otra consulta para obtener sus nombres. Utilizando la combinación de tablas, sólo hace falta una consulta:

```
SELECT nombre
FROM estaciones, lineas_estaciones
WHERE num_linea = 1
AND estaciones.cod_estacion = lineas_estaciones.cod_estacion;
```

```
+-----+
| nombre |
+-----+
| Pinar de Chamartin |
| Bambu |
| Chamartin |
| Plaza de Castilla |
| Cuatro Caminos |
| Iglesia |
| Bilbao |
| Gran Via |
| Sol |
| Atocha |
| Menendez Pelayo |
+-----+
11 rows in set (0.24 sec)
```

Este método de resolución de consultas se denomina **por igualación**.

Es necesario especificar a qué tabla pertenecen los campos comunes a varias. Por eso, en la consulta anterior, `cod_estacion` aparece prefijado con el nombre de la tabla. Si los campos no son comunes, no hace falta prefijarlos. Puede darse el caso de que haya campos con el mismo nombre, pero que guarden datos diferentes en diferentes tablas (no son campos comunes). También es necesario añadir un prefijo a estos campos.

No hay por qué restringirse a uniones que se refieran únicamente a dos tablas. Por ejemplo, para obtener las calles de los accesos de las tres primeras estaciones de las líneas 1 y 2 se combinan tres tablas:

```
SELECT num_linea, nombre [AS] estación, calle
FROM lineas_estaciones, estaciones, accesos
WHERE num_linea <= 2
AND orden <=3
AND accesos.cod_estacion = estaciones.cod_estacion
AND estaciones.cod_estacion = lineas_estaciones.cod_estacion;
```

```

+-----+-----+-----+
| num_linea | estacion          | calle                |
+-----+-----+-----+
|          1 | Pinar de Chamartin | Arturo Soria         |
|          1 | Bambu              | Bambu                |
|          1 | Chamartin          | Agustin de Foxa     |
|          1 | Chamartin          | Vestibulo Cercacnias |
|          2 | Ventas             | Alcala               |
|          2 | Goya               | Conde de Pe         |
|          2 | Goya               | Goya                 |
|          2 | Goya               | Goya                 |
|          2 | Goya               | General Diaz Porlier |
|          2 | Principe de Vergara | Alcala               |
+-----+-----+-----+
10 rows in set (0.05 sec)

```

El uso del prefijo con el nombre de las tablas, puede conducir a nombres bastante largos. Para evitarlo, se define un nombre abreviado o alias en la cláusula `FROM`, el cual se puede usar en cualquier parte dentro de la sentencia `SELECT`. En la cláusula `FROM`, el alias aparece inmediatamente después del nombre de la tabla separado por un espacio:

```
FROM tabla1 alias1, ..., tablan aliasn
```

Aplicando alias al último ejemplo, la consulta quedaría de la siguiente manera:

```

SELECT num_linea, nombre [AS] estación, calle
FROM lineas_estaciones t1, estaciones t2, accesos t3
WHERE num_linea <= 2
AND orden <=3
AND t3.cod_estacion = t2.cod_estacion
AND t2.cod_estacion = t1.cod_estacion;

```

La expresión condicional usada en la unión de búsquedas, puede usar otro operador que no sea el igual (=), tal y como ocurre en el siguiente ejemplo:

```

SELECT t2.campo1, t2.campo2, t2.campo3
FROM tabla1 t1, tabla2 t2
WHERE t1.campo2 = 'valor'
AND t1.campo1 = t2.campo1
AND t1.campo4 = t2.campo4
AND t1.campo3 > t2.campo3;

```

SQL también permite especificar **subconsultas**⁶⁰, que son consultas incluidas dentro de las cláusulas `WHERE` o `HAVING`. El resultado de la subconsulta, se utiliza en la búsqueda principal, por eso se ejecuta antes que ésta. Utilizar subconsultas es otro método de resolución de consultas complejas (**resolución mediante consultas anidadas**).

Existen varios operadores que se usan para introducir subconsultas y según cual usemos, el resultado será diferente:

- `IN`: como ya hemos visto al estudiar los operadores lógicos, `IN` se utiliza para comparar con una lista de valores. En este caso, dicha lista de valores la obtiene de la subconsulta que introduce.
- `EXISTS/NOT EXISTS`: este operador, también conocido como test de existencia, comprueba si la subconsulta produce algún resultado (`EXISTS`) o si no produce ningún resultado (`NOT EXISTS`). Si utilizamos `EXISTS` y la subconsulta produce algún resultado (también cuentan como resultado registros con el valor `NULL`), se devuelve el resultado `TRUE` y si no produce ninguno, el resultado `FALSE` (con `NOT EXISTS` funciona a la inversa). La subconsulta introducida por este operador debe comenzar⁶¹ con `SELECT *`.
- `ANY/SOME`: estos dos operadores son equivalentes, podemos utilizar cualquiera de los dos y obtendremos el mismo resultado. Deben ir precedidos por un operador de comparación (`=`⁶², `<`, `>`, `,`, etc.) y devuelven `TRUE` si la comparación con **al menos uno** de los valores devueltos por la subconsulta devuelve `TRUE`.

⁶⁰ A continuación vamos a ver su uso en combinación con la sentencia `SELECT`, pero las subconsultas también se pueden utilizar junto con las consultas de modificación:

```
DELETE
FROM tabla1
WHERE columna IN
(SELECT columna
FROM tabla2);
```

⁶¹ En la práctica se pueden especificar campos en la sentencia `SELECT`, pero MySQL los ignora cuando la subconsulta va precedida por `EXISTS`.

⁶² `IN` (usado en subconsultas) y `=ANY` son equivalentes. Pero `NOT IN` no es equivalente a `!= ANY`, sino a `!=ALL`.

- ALL: también debe ir precedido por un operador de comparación (=, <, >, etc.) y devuelve TRUE si la comparación con **todos** los valores devueltos por la subconsulta devuelve TRUE.

En el siguiente ejemplo utilizamos una subconsulta para obtener los accesos que están asociados a estaciones con cochera. Primero se realiza la subbúsqueda en la tabla `cocheras`, que selecciona los códigos de las estaciones que se utilizarán en la búsqueda principal de la tabla `accesos`:

```
SELECT *
FROM accesos
WHERE cod_estacion IN (SELECT cod_estacion
                       FROM cocheras);
```

cod_acceso	cod_estacion	calle	num_portal
A000	ES1001	Arturo Soria	330
A007	ES1005	Reina Victoria	2
A008	ES1005	Bravo Murillo	101
A019	ES1011	Menendez Pelayo	137
A020	ES1011	Gutenberg	4
A021	ES1012	Alcala	233
A034	ES1019	Gta. Quevedo	2
A036	ES1021	Plaza de Legazpi	1
A037	ES1021	Paseo de las Delicias	132
A060	ES1033	Capitan Cortes	4
A061	ES1033	Costa del Sol	20
A072	ES1040	Gta. del Ej	1

12 rows in set (0.00 sec)

Esta consulta se puede resolver utilizando `EXISTS` y se llega al mismo resultado:

```
SELECT *
FROM accesos
WHERE EXISTS (SELECT *
             FROM cocheras
             WHERE accesos.cod_estacion = cocheras.cod_estacion);
```

Otra opción equivalente, consiste en utilizar el método por igualdad:

```
SELECT DISTINCT cod_acceso, t1.cod_estacion, calle, num_portal
FROM accesos t1, cocheras t2
WHERE t1.cod_estacion = t2.cod_estacion;
```

También es posible anidar subconsultas dentro de otras subconsultas. Partiendo del ejemplo anterior, añadimos una subbúsqueda que nos permite quedarnos únicamente con los accesos de las estaciones cuyas cocheras tengan la máxima capacidad de máquinas:

```
SELECT *
FROM accesos
WHERE cod_estacion IN (SELECT cod_estacion
                        FROM cocheras
                        WHERE num_maquinas = (SELECT MAX(num_maquinas)
                                             FROM cocheras)
                        );
```

```
+-----+-----+-----+-----+
| cod_acceso | cod_estacion | calle       | num_portal |
+-----+-----+-----+-----+
| A072       | ES1040       | Gta. del Ej | 1          |
+-----+-----+-----+-----+
1 row in set (0.06 sec)
```

En vez de utilizar el operador de igualdad, también se pueden utilizar los operadores mayor/menor que, como en el siguiente ejemplo, en el que nos deshacemos de las cocheras con la menor capacidad:

```
SELECT *
FROM accesos
WHERE cod_estacion IN (SELECT cod_estacion
                        FROM cocheras
                        WHERE num_maquinas > (SELECT MIN(num_maquinas)
                                              FROM cocheras)
                        );
```

cod_acceso	cod_estacion	calle	num_portal
A000	ES1001	Arturo Soria	330
A007	ES1005	Reina Victoria	2
A008	ES1005	Bravo Murillo	101
A019	ES1011	Menendez Pelayo	137
A020	ES1011	Gutenberg	4
A036	ES1021	Plaza de Legazpi	1
A037	ES1021	Paseo de las Delicias	132
A060	ES1033	Capitan Cortes	4
A061	ES1033	Costa del Sol	20
A072	ES1040	Gta. del Ej	1

10 rows in set (0.00 sec)

A continuación presentamos cuatro ejemplos cuyo objetivo es establecer claramente la diferencia entre ANY y ALL. La subconsulta que se utiliza es algo más compleja que las que hemos visto hasta ahora, por lo que primero vamos a ver el resultado que se obtiene al ejecutarla independientemente:

```
SELECT num_linea, 2*COUNT(num_linea)
FROM trenes
GROUP BY num_linea
HAVING num_linea IS NOT NULL;
```

num_linea	2*COUNT(num_linea)
1	26
2	20
3	16
4	16
5	30

5 rows in set (0.00 sec)

Lo que se obtiene, es el número de trenes asignados a cada línea multiplicado por dos⁶³. En los siguientes ejemplos vamos a comparar la capacidad de las cocheras con estos valores.

Primero nos quedamos con las cocheras que **tienen capacidad suficiente** para almacenar el doble de las máquinas asignadas a **cualquiera** de las líneas:

```
SELECT cod_cochera, num_maquina
FROM cocheras
WHERE num_maquinas >= ALL (SELECT 2*COUNT(num_linea)
                            FROM trenes
                            GROUP BY num_linea)
```

⁶³ Obviando la columna num_linea, que no va a aparecer en la subconsulta y que se ha puesto en el ejemplo para aportar claridad.

```
HAVING num_linea IS NOT NULL
```

```
);
```

```
+-----+-----+
| cod_cochera | num_maquinas |
+-----+-----+
| C01         |          50  |
| C04         |          35  |
| C07         |          50  |
| C08         |          30  |
| C09         |          40  |
| C10         |          70  |
+-----+-----+
6 rows in set (0.00 sec)
```

En el segundo ejemplo, con las cocheras que **tienen capacidad suficiente** para almacenar el doble de las máquinas asignadas a **por lo menos una** de las líneas:

```
SELECT cod_cochera, num_maquina
FROM cocheras
WHERE num_maquinas >= ANY (SELECT 2*COUNT(num_linea)
                            FROM trenes
                            GROUP BY num_linea
                            HAVING num_linea IS NOT NULL
                           );
```

```
+-----+-----+
| cod_cochera | num_maquinas |
+-----+-----+
| C01         |          50  |
| C02         |          20  |
| C03         |          20  |
| C04         |          35  |
| C07         |          50  |
| C08         |          30  |
| C09         |          40  |
| C10         |          70  |
+-----+-----+
8 rows in set (0.00 sec)
```

En el siguiente, con las cocheras que **no tienen capacidad suficiente** para almacenar el doble de las máquinas asignadas a **ninguna** de las líneas:

```
SELECT cod_cochera, num_maquina
FROM cocheras
WHERE num_maquinas < ALL (SELECT 2*COUNT(num_linea)
                          FROM trenes
```

```
GROUP BY num_linea
HAVING num_linea IS NOT NULL
);
```

cod_cochera	num_maquinas
C05	15
C06	15

2 rows in set (0.00 sec)

Y por último, con las cocheras que **no tienen capacidad suficiente** para almacenar el doble de las máquinas asignadas a **alguna** de las líneas:

```
SELECT cod_cochera, num_maquina
FROM cocheras
WHERE num_maquinas < ANY (SELECT 2*COUNT(num_linea)
FROM trenes
GROUP BY num_linea
HAVING num_linea IS NOT NULL
);
```

cod_cochera	num_maquinas
C02	20
C03	20
C05	15
C06	15

4 rows in set (0.00 sec)

Se pueden comparar los resultados de búsquedas y/o sub-búsquedas mediante los siguientes operadores⁶⁴:

- UNION: permite unir varias consultas. Aparecen los registros distintos devueltos por todas las búsquedas implicadas. Si se quiere que también aparezcan los registros duplicados, hay que añadir la palabra clave ALL.
- INTERSECTION: calcula la intersección entre dos o más búsquedas, es decir, devuelve los registros comunes a todas las búsquedas⁶⁵.

⁶⁴ De los operadores que se van a presentar, el único que funciona en MySQL es UNION.

- **MINUS/EXCEPT:** calcula la diferencia entre dos o más consultas, es decir, devuelve los registros que aparecen en la primera búsqueda, pero no en la segunda. Si hay una tercera búsqueda, los registros que aparezcan en ella también se eliminarán y así sucesivamente⁶⁶.

Es importante señalar que las búsquedas con las que operen estos operadores deben ser compatibles, es decir, las cláusulas `SELECT` de ambas búsquedas deben contener el mismo número y tipo de campos.

4.4.2.1 Ejercicios

Una vez que hemos visto todas las herramientas que podemos utilizar a la hora de realizar búsquedas, vamos a ver cómo resolver consultas más elaboradas. A continuación se presentan varios ejercicios con su solución correspondiente:

- 1) Crea una consulta SQL con la que se obtenga el total de vagones que se pueden almacenar en cada estación (muestra el nombre de la estación, no el código) de las líneas 1, 3 y verde en las que se encuentren trenes que se hayan comprado a lo largo de 2002. Ordena el resultado de mayor a menor cantidad de vagones.

```
SELECT nombre, SUM(cocheras.num_vagones)
FROM cocheras, estaciones, lineas_estaciones, lineas, trenes
WHERE cocheras.cod_estacion = estaciones.cod_estacion
      AND estaciones.cod_estacion = lineas_estaciones.cod_estacion
      AND lineas_estaciones.num_linea = lineas.num_linea
      AND (lineas.num_linea IN (1,3) OR color = 'verde')
      AND lineas.num_linea = trenes.num_linea
      AND fecha_compra LIKE '2002%'
GROUP BY nombre
ORDER BY 2 DESC;
```

```
SELECT DISTINCT nombre, cocheras.num_vagones
FROM cocheras, estaciones, lineas_estaciones, lineas, trenes
WHERE cocheras.cod_estacion = estaciones.cod_estacion
```

⁶⁵ La intersección entre dos consultas también se puede conseguir utilizando `IN+subconsulta` o `EXISTS+subconsulta`.

⁶⁶ La diferencia entre dos consultas también se puede conseguir utilizando `NOT IN+subconsulta` o `NOT EXISTS+subconsulta`.

```
AND estaciones.cod_estacion = lineas_estaciones.cod_estacion
AND lineas_estaciones.num_linea = lineas.num_linea
AND (lineas.num_linea IN (1,3) OR color = 'verde')
AND lineas.num_linea = trenes.num_linea
AND fecha_compra LIKE '2002%' // fecha_compra BETWEEN '2002-01-01' AND '2002-12-31'
ORDER BY 2 DESC;
```

```
+-----+-----+
| nombre | num_vagones |
+-----+-----+
| Carabanchel | 420 |
| Ventas | 90 |
+-----+-----+
2 rows in set, 1 warning (0.00 sec)
```

- 2) Obtenga el nombre de las estaciones junto con su hora de cierre de entre aquellas con accesos a la calle "Gran Vía" entre el número 24 y el 40 que pertenezcan a una línea cuyo primer tren salga antes de las 7 de la mañana. Utiliza el método de resolución mediante consultas anidadas:

```
SELECT nombre, hora_cierre
FROM estaciones
WHERE cod_estacion IN (SELECT cod_estacion
                        FROM accesos
                        WHERE calle = "Gran Vía"
                        AND num_portal BETWEEN 24 AND 30)
AND cod_estacion IN (SELECT cod_estacion
                     FROM lineas_estaciones
                     WHERE num_linea IN (SELECT num_linea
                                          FROM lineas
                                          WHERE hora_primer_tren < "07:00:00")
                     );
```

```
+-----+-----+
| nombre | hora_cierre |
+-----+-----+
| Gran Vía | 02:00:00 |
+-----+-----+
1 row in set (0.06 sec)
```

- 3) Elimina los trenes con fecha de compra anterior al año 2000 y que estén asignados a líneas con cochera en la primera de sus estaciones con espacio para menos de 30 máquinas.

```
DELETE
FROM trenes
WHERE fecha_compra LIKE '1%'
AND num_linea IN (SELECT num_linea
                  FROM lineas_estaciones
                  WHERE orden = 1
                  AND cod_estacion IN (SELECT cod_estacion
                                       FROM cocheras
                                       WHERE num_maquinas < 30)
                  );
```


5 CONCLUSIONES

El objetivo de este trabajo era el estudio de diferentes tecnologías relacionadas con el campo de desarrollo web. En concreto, el posicionamiento web, las bases de datos y el lenguaje SQL.

Antes de recibir la asignación del proyecto, no estaba familiarizada en absoluto con el SEO, ni siquiera sabía de su existencia. En retrospectiva, me parece increíble que no hubiera oído hablar de ello antes. Debido al estado del panorama web, siempre creciendo y siempre cambiando, la necesidad de aplicar SEO a nuestras páginas web es clara. Cuánto más competitivo es un mercado, más difícil resulta hacerse un hueco y diferenciarse del resto de la oferta. En el mercado particular de las páginas web, esto se traduce en una mayor necesidad de entender cómo funcionan los motores de búsqueda, ya que éstos constituyen nuestra plataforma de acceso a potenciales usuarios, con el objetivo de poder utilizar esta información en nuestro beneficio y aquí, es donde entra en juego el SEO.

Cantidad de profesionales del sector se están especializando en este campo, ya que la demanda no para de crecer. Además, como las necesidades y tecnologías web, así como los algoritmos utilizados por los motores de búsqueda, están en constante evolución, las técnicas SEO también. Lo cual quiere decir, que la demanda de profesionales SEO no es algo puntual, es necesario actualizarse continuamente. Por lo tanto, creo que un acercamiento al SEO puede resultar muy interesante para los alumnos a los que está dirigida esta documentación, ya que les introduce a un nuevo sector profesional, que quizás desconocen.

El tema de diseño de bases de datos y del lenguaje SQL lo enfoqué de manera diferente, ya que yo misma he sido alumna de TAW y por lo tanto, sí que estaba familiarizada con ellos, así como con la propia documentación. En un principio, el tema de diseño de bases de datos fue lo que más me atrajo del proyecto, ya que en las prácticas que realicé el año pasado en Alemania en la empresa Daimler AG, las tareas que debía llevar a cabo estaban directamente relacionadas con bases de datos. Éste fue mi primer contacto con el mundo laboral y, en consecuencia, mi primer contacto con la importancia de la organización y el análisis de datos.

Una gran parte de los recursos de las empresas, está dedicada a la recolección y almacenamiento de datos, con el objetivo de analizarlos y utilizarlos en la toma de decisiones. Una buena estructuración de los datos, o lo que es lo mismo, un buen diseño de base de datos, que se ajuste a las necesidades de la empresa ahorra mucho tiempo y dinero. Directamente relacionado, está el uso del lenguaje SQL, que es el estándar de facto de la mayoría de los SGBD relacionales que se usan actualmente y por lo tanto, es importante conocer su manejo en profundidad.

La principal ventaja de haber trabajado previamente como alumna con la documentación, tanto de diseño de base de datos, como de SQL, ha sido la facilidad

para reconocer algunos de los puntos débiles de la misma, explicaciones que me habían resultado confusas o difíciles de entender al estudiar la materia, así como otras que me habían parecido incompletas.

Mientras que las asignaturas del Grado me han preparado para el trabajo en grupo, la realización de este proyecto ha supuesto mi primer trabajo a gran escala realizado de manera prácticamente autónoma, que creo que es muy importante para el futuro laboral. Además, me ha gustado especialmente que el objetivo fuera el desarrollo de documentación desde una perspectiva docente. Siempre me ha gustado enseñar y la transformación de información más técnica u obtusa a información más entendible me llama especialmente la atención.

6 FUTURAS LÍNEAS DE DESARROLLO

Siempre hay cabida para la superación, y aunque estoy contenta con el resultado del proyecto, hay ciertos puntos que se podrían extender y algunos temas nuevos que se podrían incluir.

Con respecto al tema de **posicionamiento web**, hay tanta información al respecto y las tendencias cambian con tanta rapidez, que resulta difícil discernir qué tiene más importancia o qué aspectos van a seguir teniendo importancia en unos meses. Además, teniendo en cuenta que el documento tiene como objetivo ser la guía de un único seminario, las limitaciones aumentan. Aun así, se podría dedicar una sección a la importancia que los buscadores dan a los enlaces, (qué es el *link building*, los *backlinks*, etc.), ya que es un factor fundamental de SEO. Especialmente interesante resultaría explorar las tendencias que vienen con más fuerza y que se han especificado en la sección dedicada al futuro del SEO [Sección 2.6]: adaptación a móviles, indexación en tiempo real, etc.

Por otra parte, la sección dedicada al **diseño de BBDD**, contiene los fundamentos teóricos más básicos del diseño y no creo que se puedan ampliar más debido a las limitaciones de tiempo (esta sección ha sido desarrollada teniendo en cuenta que la asignatura en la que se va a utilizar sólo dispone de una clase para exponer dicho material). Lo que sí puede ser interesante, por ejemplo, es incluir un ejemplo de diseño (podría ser la base de datos *lineasmetro*) que pase por todas las fases (diseño conceptual, lógico y físico), de manera que se vea de manera más intuitiva cómo va cambiando el esquema en cada fase, sin necesidad de meter más teoría.

La principal mejora que se podría hacer a la parte relativa **SQL** sería añadir más ejemplos. En particular, más ejemplos complejos. De este último tipo se podrían añadir varias consultas que incluyeran la resolución por igualación y por consultas anidadas, para que queden claros los dos enfoques.

Como complemento a los temas tratados, y también pertenecientes al campo del desarrollo web, sería especialmente interesante estudiar el **paradigma de diseño adaptativo** y, del lado del servidor, realizar una revisión de **PHP** (nuevas librerías, expresiones regulares, uso junto a Ajax, etc.) que no se ha hecho por falta de tiempo.

7 BLIOGRAFÍA

- [1] *SQL Tutorial* (2016). Consultado por última vez en julio de 2016
<http://www.w3schools.com/sql/>
- [2] Cuenca, Carlos Luis (2013). *Tipos de Datos de Mysql*. Consultado por última vez en julio de 2016
<http://www.desarrolloweb.com/articulos/1054.php>
- [3] *MySQL 5.5 Reference Manual* (2008). Consultado por última vez en julio de 2016
<https://dev.mysql.com/doc/refman/5.5/en/>
- [4] Maldonado, Javier (2002). *El posicionamiento en los buscadores de Internet*. Consultado por última vez en julio de 2016
<http://www.desarrolloweb.com/contacta/41.html>
- [5] *¿Qué es el SEO? ¿Puedo hacer SEO?* (2009). Consultado por última vez en julio de 2016
<http://www.blogtecnologico.net/que-es-el-seo-puedo-hacer-seo/>
- [6] *URL* (2016). Consultado por última vez en julio de 2016
<https://de.onpage.org/wiki/URL>
- [7] *What Is SEO / Search Engine Optimization?* (2016). Consultado por última vez en julio de 2016
<http://searchengineland.com/guide/what-is-seo>
- [8] *Empresas con página web*(2016). Consultado por última vez en junio de 2016
<http://www.ontsi.red.es/ontsi/es/indicador/empresas-con-página-web>
- [9] *Individuos que usan regularmente Internet* (2015). Consultado por última vez en junio de 2016
<http://www.ontsi.red.es/ontsi/es/indicador/individuos-que-usan-regularmente-internet>
- [10] Fishkin, Rand (2015). *The Beginners Guide to SEO*. Consultado por última vez en julio de 2016
<https://moz.com/beginners-guide-to-seo>
- [11] *What is Robots.txt?* (2016). Consultado el día de 2016
<https://moz.com/learn/seo/robotstxt>

- [12] Dr. Meyers, Peter J. (2011). *Duplicate Content in a Post-Panda World*. Consultado por última vez en julio de 2016
<https://moz.com/blog/duplicate-content-in-a-post-panda-world>
- [13] *Market Share Report* (2016). Consultado por última vez en junio de 2016
<https://www.netmarketshare.com>
- [14] *Cómo funciona Búsqueda de Google* (2016). Consultado por última vez en junio de 2016
<https://support.google.com/webmasters/answer/70897/?hl=es>
- [15] *Cómo usar rel="nofollow" para enlaces específicos* (2016). Consultado por última vez en julio de 2016
<https://support.google.com/webmasters/answer/96569?hl=es>
- [16] Kupke, Joachim, Ohye, Maile (2009). *Specify your canonical*. Consultado por última vez en julio de 2016
<https://webmasters.googleblog.com/2009/02/specify-your-canonical.html>
- [17] Zohary, Na'ama, Ng, Elliott (2016). *Introducing rich cards*. Consultado por última vez en julio de 2016
<https://webmasters.googleblog.com/2016/05/introducing-rich-cards.html>
- [18] *The Periodic Table Of SEO Success Factors* (2016). Consultado por última vez en junio de 2016
<http://searchengineland.com/seotable>
- [19] *Suchmaschinenoptimierung* (2016). Consultado por última vez en julio de 2016
<https://de.onpage.org/wiki/Kategorie:Suchmaschinenoptimierung>
- [20] Yao, Spencer. *How Your Domain Name Will Impact SEO & Social Media Marketing*. Consultado por última vez en julio de 2016
<https://www.searchenginejournal.com/how-your-domain-name-will-impact-seo-social-media-marketing/>
- [21] Audette, Adam (2015). *We Tested How Googlebot Crawls Javascript And Here's What We Learned*. Consultado por última vez en julio de 2016
<http://searchengineland.com/tested-googlebot-crawls-javascript-heres-learned-220157>
- [22] *Sitemaps XML format* (2008). Consultado por última vez en julio de 2016
<http://www.sitemaps.org/protocol.html>
- [23] *Bing XML Sitemap Plugin*. Consultado por última vez en julio de 2016
<https://www.bing.com/webmaster/help/bing-xml-sitemap-plugin-f50bebf5>

- [24] *What's coming for SEO in 2016?* (2016). Consultado por última vez en julio de 2016
<http://www.link-assistant.com/news/7-seo-trends-for-2016.html>

- [25] *Building for the Next Moment* (mayo, 2012). Consultado por última vez en julio de 2016
<https://adwords.googleblog.com/2015/05/building-for-next-moment.html>

- [26] *HTTPS as a ranking signal* (agosto, 2014). Consultado por última vez en julio de 2016
<https://webmasters.googleblog.com/2014/08/https-as-ranking-signal.html>

- [27] *Search at I/O 16 Recap: Eight things you don't want to miss* (junio, 2016). Consultado por última vez en julio de 2016
<https://webmasters.googleblog.com/2016/06/search-at-io-16-recap-eight-things-you.html>

- [28] Marqués, Mercedes (2010). *Bases de Datos*.
<https://openlibra.com/es/book/bases-de-datos-2>

- [29] Watt , Adrienne, Eng, Nelson (2015). *Database Design - 2nd Edition*.
<https://openlibra.com/es/book/database-design-2nd-edition>

- [30] Paré, Rafael Camps (2007). *Introducción a las bases de datos*. Universitat Oberta de Catalunya

- [31] Costa, Dolors Costal (2007). *El modelo relacional y el álgebra relacional*. Universitat Oberta de Catalunya

- [32] Escofet, Carme Martín (2007). *El lenguaje SQL*. Universitat Oberta de Catalunya

8 ANEXO 1

CREATE TABLE lineas

```
(  
  num_linea int NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  color char(10) NOT NULL,  
  hora_primer_tren time NOT NULL,  
  hora_ultimo_tren time NOT NULL  
);
```

CREATE TABLE estaciones

```
(  
  cod_estacion char(6) NOT NULL PRIMARY KEY,  
  nombre char(30) NOT NULL,  
  hora_apertura time NOT NULL,  
  hora_cierre time NOT NULL  
);
```

CREATE TABLE lineas_estaciones

```
(  
  num_linea int NOT NULL,  
  cod_estacion char(6) NOT NULL,  
  orden int NOT NULL,  
  PRIMARY KEY(num_linea, cod_estacion)  
);
```

CREATE TABLE accesos

```
(  
  cod_estacion char(4) NOT NULL PRIMARY KEY,  
  cod_estacion char(6) NOT NULL,  
  calle char(50) NOT NULL,
```

```
num_portal char(4) NOT NULL
```

```
);
```

```
CREATE TABLE cocheras
```

```
(
```

```
cod_cochera char(4) NOT NULL PRIMARY KEY,
```

```
cod_estacion char(6) NOT NULL,
```

```
num_estacion int NOT NULL,
```

```
num_vagones int NOT NULL
```

```
);
```

```
CREATE TABLE trenes
```

```
(
```

```
num_tren int NOT NULL PRIMARY KEY,
```

```
num_linea int,
```

```
cod_cochera char(4) NOT NULL,
```

```
fecha_compra date NOT NULL,
```

```
modelo char(10) NOT NULL,
```

```
num_vagones int NOT NULL
```

```
);
```

```
CREATE TABLE abonos
```

```
(
```

```
nombre_abono char(10) NOT NULL PRIMARY KEY,
```

```
duracion_dias int NOT NULL,
```

```
precio float(8,2) NOT NULL,
```

```
precio_estudiante float(8,2) NOT NULL,
```

```
precio_jubilado float(8,2) NOT NULL,
```

```
);
```

```
CREATE TABLE abonos_lineas
```



```
(  
nombre_abono char(10) NOT NULL,  
num_linea int NOT NULL,  
PRIMARY KEY(nombre_abono, num_linea)  
);
```

INSERT

```
INSERT INTO lineas (color, hora_primer_tren, hora_ultimo_tren)
```

```
VALUES ('azul', '06:05:00', '02:00:00'),  
        ('rojo', '06:05:00', '02:00:00'),  
        ('amarillo', '06:05:00', '02:00:00'),  
        ('marron', '07:00:00', '01:00:00'),  
        ('verde', '07:00:00', '01:00:00');
```

```
INSERT INTO estaciones
```

```
VALUES ('ES1001', 'Pinar de Chamartin', '06:05:00', '02:00:00'),  
        ('ES1002', 'Bambu', '06:05:00', '02:00:00'),  
        ('ES1003', 'Chamartin', '06:05:00', '02:00:00'),  
        ('ES1004', 'Plaza de Castilla', '06:05:00', '02:00:00'),  
        ('ES1005', 'Cuatro Caminos', '06:05:00', '02:00:00'),  
        ('ES1006', 'Iglesia', '06:05:00', '02:00:00'),  
        ('ES1007', 'Bilbao', '06:05:00', '02:00:00'),  
        ('ES1008', 'Gran Via', '06:05:00', '02:00:00'),  
        ('ES1009', 'Sol', '06:05:00', '02:00:00'),  
        ('ES1010', 'Atocha', '06:05:00', '02:00:00'),  
        ('ES1011', 'Menendez Pelayo', '06:05:00', '02:00:00'),  
        ('ES1012', 'Ventas', '06:05:00', '02:00:00'),  
        ('ES1013', 'Goya', '06:05:00', '02:00:00'),  
        ('ES1014', 'Principe de Vergara', '06:05:00', '02:00:00'),  
        ('ES1015', 'Retiro', '06:05:00', '02:00:00'),  
        ('ES1016', 'Banco de España', '06:05:00', '02:00:00'),  
        ('ES1017', 'Opera', '06:05:00', '02:00:00');
```

```
('ES1018', 'Noviciado', '06:05:00', '02:00:00'),  
( 'ES1019', 'Quevedo', '06:05:00', '02:00:00'),  
( 'ES1020', 'Canal', '06:05:00', '02:00:00'),  
( 'ES1021', 'Legazpi', '06:05:00', '02:00:00'),  
( 'ES1022', 'Delicias', '06:05:00', '02:00:00'),  
( 'ES1023', 'Embajadores', '06:05:00', '02:00:00'),  
( 'ES1024', 'Lavapies', '06:05:00', '02:00:00'),  
( 'ES1025', 'Callao', '06:05:00', '02:00:00'),  
( 'ES1026', 'Plaza de España', '06:05:00', '02:00:00'),  
( 'ES1027', 'Moncloa', '06:05:00', '02:00:00'),  
( 'ES1028', 'Alonso Martinez', '07:00:00', '01:00:00'),  
( 'ES1029', 'Colon', '07:00:00', '01:00:00'),  
( 'ES1030', 'Serrano', '07:00:00', '01:00:00'),  
( 'ES1031', 'Diego de Leon', '07:00:00', '01:00:00'),  
( 'ES1032', 'San Lorenzo', '07:00:00', '01:00:00'),  
( 'ES1033', 'Hortaleza', '07:00:00', '01:00:00'),  
( 'ES1034', 'Suanzes', '07:00:00', '01:00:00'),  
( 'ES1035', 'El Carmen', '07:00:00', '01:00:00'),  
( 'ES1036', 'Nuñez de Balboa', '07:00:00', '01:00:00'),  
( 'ES1037', 'Ruben Dario', '07:00:00', '01:00:00'),  
( 'ES1038', 'La Latina', '07:00:00', '01:00:00'),  
( 'ES1039', 'Piramides', '07:00:00', '01:00:00'),  
( 'ES1040', 'Carabanchel', '07:00:00', '01:00:00');
```

INSERT INTO líneas_estaciones

VALUES (1, 'ES1001', 1),
(1, 'ES1002', 2),
(1, 'ES1003', 3),
(1, 'ES1004', 4),
(1, 'ES1005', 5),
(1, 'ES1006', 6),

(1, 'ES1007', 7),
(1, 'ES1008', 8),
(1, 'ES1009', 9),
(1, 'ES1010', 10),
(1, 'ES1011', 11),

(2, 'ES1012', 1),
(2, 'ES1013', 2),
(2, 'ES1014', 3),
(2, 'ES1015', 4),
(2, 'ES1016', 5),
(2, 'ES1009', 6),
(2, 'ES1017', 7),
(2, 'ES1018', 8),
(2, 'ES1019', 9),
(2, 'ES1020', 10),
(2, 'ES1005', 11),

(3, 'ES1021', 1),
(3, 'ES1022', 2),
(3, 'ES1023', 3),
(3, 'ES1024', 4),
(3, 'ES1009', 5),
(3, 'ES1025', 6),
(3, 'ES1026', 7),
(3, 'ES1027', 8),
(3, 'ES1019', 9),

(4, 'ES1007', 1),
(4, 'ES1028', 2),
(4, 'ES1029', 3),

```
(4, 'ES1030', 4),  
(4, 'ES1013', 5),  
(4, 'ES1031', 6),  
(4, 'ES1032', 7),  
(4, 'ES1033', 8),  
  
(5, 'ES1034', 1),  
(5, 'ES1035', 2),  
(5, 'ES1012', 3),  
(5, 'ES1031', 4),  
(5, 'ES1036', 5),  
(5, 'ES1037', 6),  
(5, 'ES1028', 7),  
(5, 'ES1008', 8),  
(5, 'ES1025', 9),  
(5, 'ES1017', 10),  
(5, 'ES1038', 11),  
(5, 'ES1039', 12),  
(5, 'ES1040', 13);
```

INSERT INTO abonos

```
VALUES ('Mensual_A', 30, 54.60, 20, 12.30),  
( 'Mensual_B', 30, 63.70, 20, 12.30),  
( 'Mensual_C', 30, 72, 20, 12.30),  
( 'Anual_A', 365, 546, 200, 123),  
( 'Anual_B', 365, 637, 200, 123),  
( 'Anual_C', 365, 720, 200, 123),  
( 'Turista_1', 1, 8.40, 8.40, 8.40),  
( 'Turista_2', 2, 14.20, 14.20, 14.20),  
( 'Turista_3', 3, 18.40, 18.40, 18.40),  
( 'Turista_5', 5, 26.80, 26.80, 26.80),
```

```
('Turista_7', 7, 35.40, 35.40, 35.40);
```

```
INSERT INTO abonos_lineas
```

```
VALUES ('Mensual_A', 1),
```

```
        ('Mensual_A', 2),
```

```
        ('Mensual_A', 3),
```

```
        ('Mensual_B', 1),
```

```
        ('Mensual_B', 2),
```

```
        ('Mensual_B', 3),
```

```
        ('Mensual_B', 4),
```

```
        ('Mensual_C', 1),
```

```
        ('Mensual_C', 2),
```

```
        ('Mensual_C', 3),
```

```
        ('Mensual_C', 4),
```

```
        ('Mensual_C', 5),
```

```
        ('Anual_A', 1),
```

```
        ('Anual_A', 2),
```

```
        ('Anual_A', 3),
```

```
        ('Anual_B', 1),
```

```
        ('Anual_B', 2),
```

```
        ('Anual_B', 3),
```

```
        ('Anual_B', 4),
```

```
        ('Anual_C', 1),
```

```
        ('Anual_C', 2),
```

```
        ('Anual_C', 3),
```

```
        ('Anual_C', 4),
```

```
        ('Anual_C', 5),
```

```
        ('Turista_1', 1),
```

```
        ('Turista_1', 2),
```

```
('Turista_1', 3),  
( 'Turista_2', 1),  
( 'Turista_2', 2),  
( 'Turista_2', 3),  
( 'Turista_3', 1),  
( 'Turista_3', 2),  
( 'Turista_3', 3),  
( 'Turista_5', 1),  
( 'Turista_5', 2),  
( 'Turista_5', 3),  
( 'Turista_5', 4),  
( 'Turista_5', 5),  
( 'Turista_7', 1),  
( 'Turista_7', 2),  
( 'Turista_7', 3),  
( 'Turista_7', 4),  
( 'Turista_7', 5);
```

INSERT INTO accesos

```
VALUES ('A000', 'ES1001', 'Arturo Soria', '330'),  
( 'A001', 'ES1002', 'Bambu', '14'),  
( 'A002', 'ES1003', 'Agustin de Foxa', '1'),  
( 'A003', 'ES1003', 'Vestibulo Cercacnias', '1'),  
( 'A004', 'ES1004', 'Paseo de la Castellana', '189'),  
( 'A005', 'ES1004', 'Bravo Murillo', '377'),  
( 'A006', 'ES1004', 'Plaza de Castilla', '1'),  
( 'A007', 'ES1005', 'Reina Victoria', '2'),  
( 'A008', 'ES1005', 'Bravo Murillo', '101'),  
( 'A009', 'ES1006', 'General Martinez Campos', '1'),  
( 'A010', 'ES1006', 'Santa Engracia', '58'),  
( 'A011', 'ES1006', 'Eloy Gonzalo', '38'),  
( 'A012', 'ES1007', 'Gta. Bilbao', '5'),  
( 'A013', 'ES1007', 'Luchana', '22'),
```

('A014', 'ES1008', 'Gran Via', '25'),
('A015', 'ES1008', 'Montera', '44'),
('A016', 'ES1009', 'Pta. del Sol', '7'),
('A017', 'ES1009', 'Preciados', '3'),
('A018', 'ES1010', 'Gta. Carlos V', '11'),
('A019', 'ES1011', 'Menendez Pelayo', '137'),
('A020', 'ES1011', 'Gutenberg', '4'),
('A021', 'ES1012', 'Alcala', '233'),
('A022', 'ES1013', 'Conde de Peñalver', '2'),
('A023', 'ES1013', 'Goya', '89'),
('A024', 'ES1013', 'Goya', '62'),
('A025', 'ES1013', 'General Diaz Porlier', '3'),
('A026', 'ES1014', 'Alcala', '123'),
('A027', 'ES1015', 'Lagasca', '4'),
('A028', 'ES1015', 'Alcala', '75'),
('A029', 'ES1016', 'Alcala', '51'),
('A030', 'ES1016', 'Alcala', '47'),
('A031', 'ES1017', 'Plaza de Isabel II', '1'),
('A032', 'ES1018', 'San Bernardo', '45'),
('A033', 'ES1018', 'San Bernardo', '51'),
('A034', 'ES1019', 'Gta. Quevedo', '2'),
('A035', 'ES1020', 'Bravo Murillo', '40'),
('A036', 'ES1021', 'Plaza de Legazpi', '1'),
('A037', 'ES1021', 'Paseo de las Delicias', '132'),
('A038', 'ES1022', 'Paseo de las Delicias', '60'),
('A039', 'ES1022', 'Paseo de las Delicias', '78'),
('A040', 'ES1023', 'Gta. Embajadores', '7'),
('A041', 'ES1024', 'Argumosa', '1'),
('A042', 'ES1025', 'Jacometrezo', '1'),
('A043', 'ES1025', 'Plaza de Callao', '1'),
('A044', 'ES1025', 'Gran Via', '39'),
('A045', 'ES1026', 'Gran Via', '73'),
('A046', 'ES1026', 'Princesa', '1'),
('A047', 'ES1026', 'Plaza de España', '18'),
('A048', 'ES1026', 'Plaza Conde de Toreno', '1'),

('A049', 'ES1027', 'Arcipreste de Hita', '1'),
('A050', 'ES1027', 'Princesa', '96'),
('A051', 'ES1028', 'Plaza Santa Barbara', '5'),
('A052', 'ES1028', 'Genova', '2'),
('A053', 'ES1029', 'Genova', '29'),
('A054', 'ES1030', 'Goya', '15'),
('A055', 'ES1030', 'Goya', '23'),
('A056', 'ES1031', 'Diego de Leon', '69'),
('A057', 'ES1031', 'Francisco Silvela', '56'),
('A058', 'ES1031', 'Francisco Silvela', '47'),
('A059', 'ES1032', 'Avda. Barranquilla', '11'),
('A060', 'ES1033', 'Capitan Cortes', '4'),
('A061', 'ES1033', 'Costa del Sol', '20'),
('A062', 'ES1034', 'Alcala', '527'),
('A063', 'ES1035', 'Alcala', '258'),
('A064', 'ES1035', 'Alcalde López Casero', '2'),
('A065', 'ES1036', 'Velazquez', '92'),
('A066', 'ES1036', 'Nuñez de Balboa', '85'),
('A067', 'ES1036', 'Plaza Marques de Salamanca', '10'),
('A068', 'ES1037', 'Ruben Dario', '2'),
('A069', 'ES1037', 'Paseo de la Castellana', '39'),
('A070', 'ES1038', 'Toledo', '56'),
('A071', 'ES1039', 'Paseo de las Acacias', '50'),
('A072', 'ES1040', 'Gta. del Ejército', '1');

INSERT INTO cocheras

VALUES ('C01', 'ES1001', 50, 300),
('C02', 'ES1001', 20, 120),
('C03', 'ES1005', 20, 120),
('C04', 'ES1011', 35, 210),
('C05', 'ES1012', 15, 90),
('C06', 'ES1019', 15, 90),
('C07', 'ES1021', 50, 300),
('C08', 'ES1021', 30, 180),

('C09', 'ES1033', 40, 240),
('C10', 'ES1040', 70, 420);

INSERT INTO trenes

VALUES (100, 1, 'C01', '1994-02-01', 'T2000', 4),
(101, 1, 'C01', '1994-02-01', 'T2000', 4),
(102, 1, 'C01', '1996-02-15', 'T2000', 4),
(103, 1, 'C01', '1996-02-15', 'T2000', 4),
(104, 1, 'C01', '2006-02-04', 'T3000', 4),
(105, 1, 'C01', '2006-02-04', 'T3000', 4),
(106, 1, 'C01', '2006-02-04', 'T3000', 4),
(107, 1, 'C02', '2006-02-04', 'T3000', 4),
(108, 1, 'C02', '2006-02-04', 'T3000', 4),
(109, 1, 'C03', '2006-02-04', 'T3000', 4),
(110, 1, 'C03', '2006-02-04', 'T3000', 4),
(111, 1, 'C04', '2014-02-01', 'T9000', 6),
(112, 1, 'C04', '2014-02-01', 'T9000', 6),
(113, 2, 'C05', '1992-02-01', 'T5000', 6),
(114, 2, 'C05', '1994-02-01', 'T2000', 4),
(115, 2, 'C05', '2002-02-20', 'T7000', 6),
(116, 2, 'C05', '2002-02-20', 'T7000', 6),
(117, 2, 'C06', '2002-02-20', 'T7000', 6),
(118, 2, 'C06', '2002-02-20', 'T7000', 6),
(119, 2, 'C06', '2002-02-20', 'T7000', 6),
(120, 2, 'C06', '2002-02-20', 'T7000', 6),
(121, 2, 'C06', '2004-02-10', 'T7000', 6),
(122, 2, 'C06', '2004-02-10', 'T7000', 6),
(123, 3, 'C07', '2014-02-01', 'T9000', 6),
(124, 3, 'C07', '2014-02-01', 'T9000', 6),
(125, 3, 'C07', '2014-02-01', 'T9000', 6),
(126, 3, 'C07', '2014-02-01', 'T9000', 6),

(127, 3, 'C07', '2014-02-01', 'T9000', 6),
(128, 3, 'C07', '2014-02-01', 'T9000', 6),
(129, 3, 'C07', '2014-02-01', 'T9000', 6),
(130, 3, 'C07', '2014-02-01', 'T9000', 6),
(131, 4, 'C09', '1992-02-01', 'T5000', 6),
(132, 4, 'C09', '1996-02-15', 'T5000', 6),
(133, 4, 'C09', '1996-02-15', 'T5000', 6),
(134, 4, 'C09', '1998-02-13', 'T5000', 6),
(135, 4, 'C09', '2002-02-20', 'T7000', 6),
(136, 4, 'C09', '2002-02-20', 'T7000', 6),
(137, 4, 'C10', '2002-02-20', 'T8000', 4),
(138, 4, 'C10', '2002-02-20', 'T8000', 4),
(139, 5, 'C10', '2002-02-20', 'T7000', 6),
(140, 5, 'C10', '2002-02-20', 'T7000', 6),
(141, 5, 'C10', '2002-02-20', 'T8000', 4),
(142, 5, 'C10', '2002-02-20', 'T8000', 4),
(143, 5, 'C10', '2006-02-04', 'T3000', 4),
(144, 5, 'C10', '2006-02-04', 'T3000', 4),
(145, 5, 'C10', '2008-02-01', 'T3000', 4),
(146, 5, 'C10', '2008-02-01', 'T3000', 4),
(147, 5, 'C10', '2008-02-01', 'T3000', 4),
(148, 5, 'C10', '2008-02-01', 'T3000', 4),
(149, 5, 'C10', '2008-02-01', 'T3000', 4),
(150, 5, 'C10', '2008-02-01', 'T3000', 4),
(151, 5, 'C10', '2014-02-01', 'T9000', 6),
(152, 5, 'C10', '2014-02-01', 'T9000', 6),
(153, 5, 'C10', '2014-02-01', 'T9000', 6),
(154, NULL, 'C01', '2006-02-01', 'T3000', 4),
(155, NULL, 'C10', '2006-02-01', 'T3000', 4);