

MySQL

01_ Introducción



MySQL_ Introducción a las bases de datos

Contenido

1.	Concepto y origen de las BD y de los SGBD	4
2.	Objetivos y servicios de los SGBD	6
2.1.	Consultas no predefinidas y complejas	6
2.2.	Flexibilidad e independencia	7
2.3.	Problemas de la redundancia	8
2.4.	Integridad de los datos	9
2.5.	Concurrencia de usuarios	10
2.6.	Seguridad	11
2.7.	Otros objetivos	12
3.	Arquitectura de los SGBD	13
3.1.	Esquemas y niveles	13
3.2.	Independencia de los datos	16
3.3.	Flujo de datos y de control	18
4.	Modelos de BD	20
5.	Lenguajes y usuarios	23
6.	Administración de BD	25
7.	¿Qué tipos de base de datos existen?	27
8.	Sistema Gestor de Bases de Datos	28
8.1.	SGBD Relacionales (SQL)	28
8.2.	SGBD No Relacionales (NoSQL)	35
9.	MySQL y SQL	37
9.1.	¿Qué es SQL?	37
9.2.	¿Qué es MYSQL?	37
9.3.	Diferencia entre SQL y MySQL	37
9.4.	Conclusión:	38
10.	Instalacion MySQL	39
10.1.	Instalación de "MySQL Community"	40
ANEXOS		
Referencia rapida de comandos basicos MySQL		55
SQL vs. noSQL: ¿qué es mejor?		70

MySQL_ Introducción a las bases de datos

Introducción

Las bases de datos son sistemas de almacenamiento de información que permiten a los usuarios organizar, acceder y manipular datos de manera eficiente y segura. SQL (Structured Query Language) es un lenguaje de programación utilizado para gestionar bases de datos relacionales. MySQL es un sistema de gestión de bases de datos relacionales de código abierto que utiliza el lenguaje SQL.

Algunos de los conceptos clave en la gestión de bases de datos son:

- **Tablas:** las tablas son la estructura fundamental de una base de datos relacional. Cada tabla contiene una serie de columnas que definen los campos de datos que se almacenan y una serie de filas que contienen los datos en sí.
- **Registros:** los registros son filas individuales en una tabla que contienen datos específicos.
- **Campos:** los campos son las columnas de una tabla, cada una de las cuales define un tipo de datos específico.
- **Claves primarias:** las claves primarias son campos o combinaciones de campos que identifican de manera única cada registro en una tabla.
- **Consultas:** las consultas son solicitudes que se hacen a una base de datos para recuperar o actualizar datos específicos.
- **Índices:** los índices son estructuras de datos que aceleran la búsqueda de datos en una base de datos.

MySQL es uno de los sistemas de gestión de bases de datos más utilizados en todo el mundo y se puede utilizar en una amplia variedad de aplicaciones y entornos, desde pequeñas aplicaciones web hasta sistemas empresariales de gran escala. Si estás interesado en aprender más sobre SQL y MySQL, existen muchos recursos en línea, incluyendo tutoriales, cursos en línea y libros especializados.



MySQL_ Introducción a las bases de datos

1. Concepto y origen de las BD y de los SGBD

Las aplicaciones informáticas de los años sesenta acostumbraban a darse totalmente por lotes (batch) y estaban pensadas para una tarea muy específica relacionada con muy pocas entidades tipo.

Cada aplicación (una o varias cadenas de programas) utilizaba ficheros de movimientos para actualizar (creando una copia nueva) y/o para consultar uno o dos ficheros maestros o, excepcionalmente, más de dos. Cada programa trataba como máximo un fichero maestro, que solía estar sobre cinta magnética y, en consecuencia, se trabajaba con acceso secuencial. Cada vez que se le quería añadir una aplicación que requería el uso de algunos de los datos que ya existían y de otros nuevos, se diseñaba un fichero nuevo con todos los datos necesarios (algo que provocaba redundancia) para evitar que los programas tuviesen que leer muchos ficheros.

A medida que se fueron introduciendo las líneas de comunicación, los terminales y los discos, se fueron escribiendo programas que permitían a varios usuarios consultar los mismos ficheros online y de forma simultánea. Más adelante fue surgiendo la necesidad de hacer las actualizaciones también online.

A medida que se integraban las aplicaciones, se tuvieron que interrelacionar sus ficheros y fue necesario eliminar la redundancia. El nuevo conjunto de ficheros se debía diseñar de modo que estuviesen interrelacionados; al mismo tiempo, las informaciones redundantes (como por ejemplo, el nombre y la dirección de los clientes o el nombre y el precio de los productos), que figuraban en los ficheros de más de una de las aplicaciones, debían estar ahora en un solo lugar.

El acceso online y la utilización eficiente de las interrelaciones exigían estructuras físicas que diesen un acceso rápido, como por ejemplo los índices, las multilistas, las técnicas de hashing, etc.

Estos conjuntos de ficheros interrelacionados, con estructuras complejas y compartidos por varios procesos de forma simultánea (unos online y otros por lotes), recibieron al principio el nombre de Data Banks, y después, a inicios de los años setenta, el de Data Bases. Aquí los denominamos bases de datos (BD).

El software de gestión de ficheros era demasiado elemental para dar satisfacción a todas estas necesidades. Por ejemplo, el tratamiento de las interrelaciones no estaba previsto, no era posible que varios usuarios actualizaran datos simultáneamente, etc. La utilización de estos conjuntos de ficheros por parte de los programas de aplicación era excesivamente compleja, de modo que, especialmente durante la segunda mitad de los años setenta, fue saliendo al mercado software más sofisticado: los Data Base Management Systems, que aquí denominamos sistemas de gestión de BD (SGBD).

MySQL_ Introducción a las bases de datos

Con todo lo que hemos dicho hasta ahora, podríamos definir el término BD; una base de datos de un SI es la representación integrada de los conjuntos de entidades instancia correspondientes a las diferentes entidades tipo del SI y de sus interrelaciones. Esta representación informática (o conjunto estructurado de datos) debe poder ser utilizada de forma compartida por muchos usuarios de distintos tipos.

En otras palabras, una base de datos es un conjunto estructurado de datos que representa entidades y sus interrelaciones. La representación será única e integrada, a pesar de que debe permitir utilizaciones varias y simultáneas.

Los ficheros tradicionales y las BD

Aunque de forma muy simplificada, podríamos enumerar las principales diferencias entre los ficheros tradicionales y las BD tal y como se indica a continuación:

Entidades tipos:

- Ficheros: tienen registros de una sola entidad tipo.
- BD: tienen datos de varias entidades tipo.

Interrelaciones:

- Ficheros: el sistema no interrelaciona ficheros.
- BD: el sistema tiene previstas herramientas para interrelacionar entidades.

Redundancia:

- Ficheros: se crean ficheros a la medida de cada aplicación, con todos los datos necesarios aunque algunos sean redundantes respecto de otros ficheros.
- BD: todas las aplicaciones trabajan con la misma BD y la integración de los datos es básica, de modo que se evita la redundancia.

Usuarios

- Ficheros: sirven para un solo usuario o una sola aplicación. Dan una sola visión del mundo real.
- BD: es compartida por muchos usuarios de distintos tipos. Ofrece varias visiones del mundo real.

MySQL_ Introducción a las bases de datos

2. Objetivos y servicios de los SGBD

Los SGBD que actualmente están en el mercado pretenden satisfacer un conjunto de objetivos directamente deducibles de lo que hemos explicado hasta ahora. A continuación los comentaremos, pero sin entrar en detalles que ya se verán más adelante en otras asignaturas.

2.1. Consultas no predefinidas y complejas

El objetivo fundamental de los SGBD es permitir que se hagan consultas no predefinidas (ad hoc) y complejas.

Consultas que afectan a más de una entidad tipo

- Se quiere conocer el número de alumnos de más de veinticinco años y con nota media superior a siete que están matriculados actualmente en la asignatura Bases de datos I.
- De cada alumno matriculado en menos de tres asignaturas, se quiere obtener el nombre, el número de matrícula, el nombre de las asignaturas y el nombre de profesores de estas asignaturas.

Los usuarios podrán hacer consultas de cualquier tipo y complejidad directamente al SGBD. El SGBD tendrá que responder inmediatamente sin que estas consultas estén preestablecidas; es decir, sin que se tenga que escribir, compilar y ejecutar un programa específico para cada consulta.

El usuario debe formular la consulta con un lenguaje sencillo (que se quede, obviamente, en el nivel lógico), que el sistema debe interpretar directamente. Sin embargo, esto no significa que no se puedan escribir programas con consultas incorporadas (por ejemplo, para procesos repetitivos).

La solución estándar para alcanzar este doble objetivo (consultas no predefinidas y complejas) es el lenguaje SQL, que explicaremos en otro módulo didáctico.

MySQL_ Introducción a las bases de datos

2.2. Flexibilidad e independencia

La complejidad de las BD y la necesidad de ir las adaptando a la evolución del SI hacen que un objetivo básico de los SGBD sea dar flexibilidad a los cambios.

Interesa obtener la máxima independencia posible entre los datos y los procesos usuarios para que se pueda llevar a cabo todo tipo de cambios tecnológicos

y variaciones en la descripción de la BD, sin que se deban modificar los programas de aplicación ya escritos ni cambiar la forma de escribir las consultas (o actualizaciones) directas.

En el mundo de los ficheros ya había independencia física en un cierto grado, pero en el mundo de las BD acostumbra a ser mucho mayor.

Ejemplos de independencia lógica

1. El personal administrativo de secretaría podría tener una visión de la entidad alumno sin que fuese necesario que existiese el atributo nota. Sin embargo, los usuarios profesores (o los programas dirigidos a ellos) podrían tener una visión en la que existiese el atributo nota pero no el atributo fecha de pago.
2. Decidimos ampliar la longitud del atributo nombre y lo aumentamos de treinta a cincuenta caracteres, pero no sería necesario modificar los programas que ya tenemos escritos si no nos importa que los valores obtenidos tengan sólo los primeros treinta caracteres del nombre.

MySQL_ Introducción a las bases de datos

2.3. Problemas de la redundancia

En el mundo de los ficheros tradicionales, cada aplicación utilizaba su fichero. Sin embargo, puesto que se daba mucha coincidencia de datos entre aplicaciones, se producía también mucha redundancia entre los ficheros. Ya hemos dicho que uno de los objetivos de los SGBD es facilitar la eliminación de la redundancia.

Seguramente pensáis que el problema de la redundancia es el espacio perdido. Antiguamente, cuando el precio del byte de disco era muy elevado, esto era un problema grave, pero actualmente prácticamente nunca lo es. ¿Qué problema hay, entonces? Simplemente, lo que todos hemos sufrido más de una vez; si tenemos algo apuntado en dos lugares diferentes no pasará demasiado tiempo hasta que las dos anotaciones dejen de ser coherentes, porque habremos modificado la anotación en uno de los lugares y nos habremos olvidado de hacerlo en el otro.

Así pues, el verdadero problema es el grave riesgo de inconsistencia o incoherencia de los datos; es decir, la pérdida de integridad que las actualizaciones pueden provocar cuando existe redundancia.

Por lo tanto, convendría evitar la redundancia. En principio, nos conviene hacer que un dato sólo figure una vez en la BD. Sin embargo, esto no siempre será cierto. Por ejemplo, para representar una interrelación entre dos entidades, se suele repetir un mismo atributo en las dos, para que una haga referencia a la otra.

Otro ejemplo podría ser el disponer de réplicas de los datos por razones de fiabilidad, disponibilidad o costes de comunicaciones.

La duplicación de datos es el tipo de redundancia más habitual, pero también tenemos redundancia cuando guardamos en la BD datos derivados (o calculados) a partir de otros datos de la misma BD. De este modo podemos responder rápidamente a consultas globales, ya que nos ahorramos la lectura de gran cantidad de registros.

En los casos de datos derivados, para que el resultado del cálculo se mantenga consistente con los datos elementales, es necesario rehacer el cálculo cada vez que éstos se modifican. El usuario (ya sea programador o no) puede olvidarse de hacer el nuevo cálculo; por ello convendrá que el mismo SGBD lo haga automáticamente.

MySQL_ Introducción a las bases de datos

2.4. Integridad de los datos

Nos interesará que los SGBD aseguren el mantenimiento de la calidad de los datos en cualquier circunstancia. Acabamos de ver que la redundancia puede provocar pérdida de integridad de los datos, pero no es la única causa posible. Se podría perder la corrección o la consistencia de los datos por muchas otras razones: errores de programas, errores de operación humana, avería de disco, transacciones incompletas por corte de alimentación eléctrica, etc.

En el subapartado anterior hemos visto que podremos decir al SGBD que nos lleve el control de las actualizaciones en el caso de las redundancias, para garantizar la integridad. Del mismo modo, podremos darle otras reglas de integridad—o restricciones— para que asegure que los programas las cumplen cuando efectúan las actualizaciones.

Aparte de las reglas de integridad que el diseñador de la BD puede definir y que el SGBD entenderá y hará cumplir, el mismo SGBD tiene reglas de integridad inherentes al modelo de datos que utiliza y que siempre se cumplirán. Son las denominadas reglas de integridad del modelo. Las reglas definibles por parte del usuario son las reglas de integridad del usuario. El concepto de integridad de los datos va más allá de prevenir que los programas usuarios almacenen datos incorrectos. En casos de errores o desastres, también podríamos perder la integridad de los datos. El SGBD nos debe dar las herramientas para reconstruir o restaurar los datos estropeados.

MySQL_ Introducción a las bases de datos

2.5. Concurrency de usuarios

Cuando los accesos concurrentes son todos de lectura (es decir, cuando la BD sólo se consulta), el problema que se produce es simplemente de rendimiento, causado por las limitaciones de los soportes de que se dispone: pocos mecanismos de acceso independientes, movimiento del brazo y del giro del disco demasiado lentos, buffers locales demasiado pequeños, etc.

Cuando un usuario o más de uno están actualizando los datos, se pueden producir problemas de interferencia que tengan como consecuencia la obtención de datos erróneos y la pérdida de integridad de la BD.

Para tratar los accesos concurrentes, los SGBD utilizan el concepto de transacción de BD, concepto de especial utilidad para todo aquello que hace referencia a la integridad de los datos, como veremos a continuación.

Ejemplos de transacciones

1. Imaginemos un programa pensado para llevar a cabo la operación de transferencia de dinero de una cuenta X a otra Y. Supongamos que la transferencia efectúa dos operaciones: en primer lugar, el cargo a X y después, el abono a Y. Este programa se debe ejecutar de forma que se hagan las dos operaciones o ninguna, ya que si por cualquier razón (por ejemplo, por interrupción del flujo eléctrico) el programa ejecutase sólo el cargo de dinero a X sin abonarlo a Y, la BD quedaría en un estado incorrecto. Queremos que la ejecución de este programa sea tratada por el SGBD como una transacción de BD.
2. Otro ejemplo de programa que querríamos que tuviera un comportamiento de transacción podría ser el que aumentara el 30% de la nota de todos los alumnos. Si sólo aumentara la nota a unos cuantos alumnos, la BD quedaría incorrecta.

Para indicar al SGBD que damos por acabada la ejecución de la transacción, el programa utilizará la operación de COMMIT. Si el programa no puede acabar normalmente (es decir, si el conjunto de operaciones se ha hecho sólo de forma parcial), el SGBD tendrá que deshacer todo lo que la transacción ya haya hecho. Esta operación se denomina ROLLBACK.

Acabamos de observar la utilidad del concepto de transacción para el mantenimiento de la integridad de los datos en caso de interrupción de un conjunto de operaciones lógicamente unitario. Sin embargo, entre transacciones que se ejecutan concurrentemente se pueden producir problemas de interferencia que hagan obtener resultados erróneos o que comporten la pérdida de la integridad de los datos.

MySQL_ Introducción a las bases de datos

2.6. Seguridad

El término seguridad se ha utilizado en diferentes sentidos a lo largo de la historia de la informática.

Estas cuestiones siempre han sido importantes en los SI militares, las agencias de información y en ámbitos similares, pero durante los años noventa han ido adquiriendo importancia en cualquier SI donde se almacenen datos sobre personas. Recordad que en el Estado español tenemos una ley*, que exige la protección de la confidencialidad de estos datos.

Los SGBD permiten definir autorizaciones o derechos de acceso a diferentes niveles: al nivel global de toda la BD, al nivel entidad y al nivel atributo.

Estos mecanismos de seguridad requieren que el usuario se pueda identificar. Se acostumbra a utilizar códigos de usuarios (y grupos de usuarios) acompañados de contraseñas (passwords), pero también se utilizan tarjetas magnéticas, identificación por reconocimiento de la voz, etc.

Nos puede interesar almacenar la información con una codificación secreta; es decir, con técnicas de encriptación (como mínimo se deberían encriptar las contraseñas). Muchos de los SGBD actuales tienen prevista la encriptación.

Prácticamente todos los SGBD del mercado dan una gran variedad de herramientas para la vigilancia y la administración de la seguridad. Los hay que, incluso, tienen opciones (con precio separado) para los SI con unas exigencias altísimas, como por ejemplo los militares.

MySQL_ Introducción a las bases de datos

2.7. Otros objetivos

Acabamos de ver los objetivos fundamentales de los SGBD actuales. Sin embargo, a medida que los SGBD evolucionan, se imponen nuevos objetivos adaptados a las nuevas necesidades y las nuevas tecnologías. Como ya hemos visto, en estos momentos podríamos citar como objetivos nuevos o recientes los siguientes:

1. Servir eficientemente los Data Warehouse.
2. Adaptarse al desarrollo orientado a objetos.
3. Incorporar el tiempo como un elemento de caracterización de la información.
4. Adaptarse al mundo de Internet.

MySQL_ Introducción a las bases de datos

3. Arquitectura de los SGBD

3.1. Esquemas y niveles

Para trabajar con nuestras BD, los SGBD necesitan conocer su estructura (qué entidades tipo habrá, qué atributos tendrán, etc.).

El esquema de la BD es un elemento fundamental de la arquitectura de un SGBD que permite independizar el SGBD de la BD; de este modo, se puede cambiar el diseño de la BD (su esquema) sin tener que hacer ningún cambio en el SGBD.

Anteriormente, ya hemos hablado de la distinción entre dos niveles de representación informática: el nivel lógico y el físico.

El nivel lógico nos oculta los detalles de cómo se almacenan los datos, cómo se mantienen y cómo se accede físicamente a ellos. En este nivel sólo se habla de entidades, atributos y reglas de integridad.

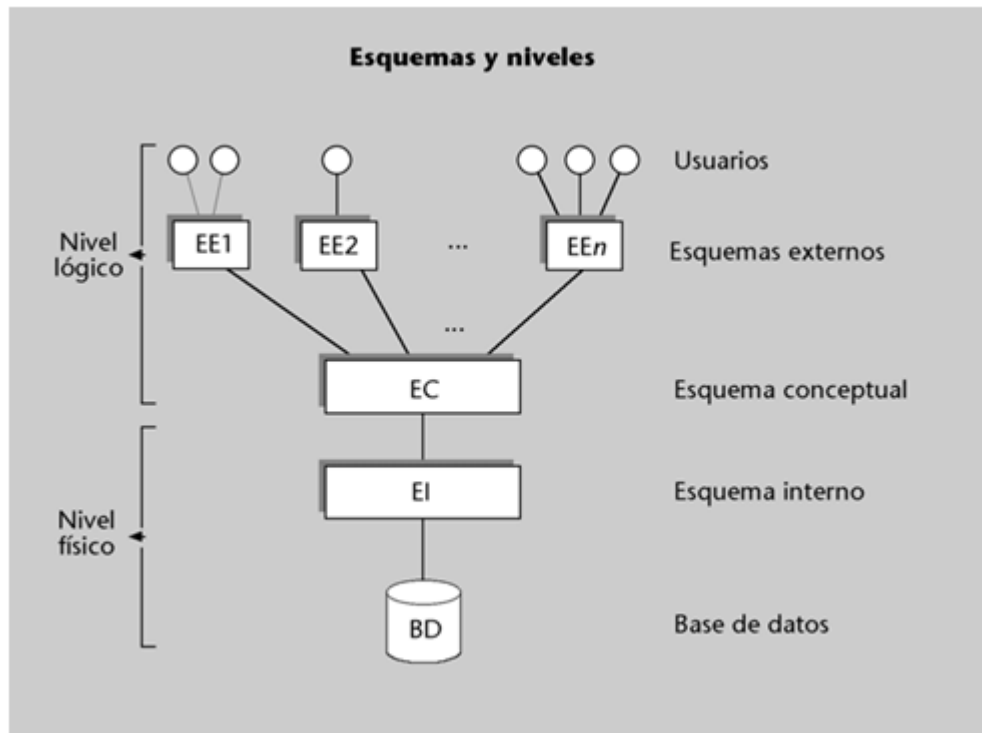
Por cuestiones de rendimiento, nos podrá interesar describir elementos de nivel físico como, por ejemplo, qué índices tendremos y qué características presentarán, cómo y dónde (en qué espacio físico) queremos que se agrupen físicamente los registros, de qué tamaño deben ser las páginas, etc.

En el periodo 1975-1982, ANSI intentaba establecer las bases para crear estándares en el campo de las BD. El comité conocido como ANSI/SPARC recomendó que la arquitectura de los SGBD previese tres niveles de descripción de la BD, no sólo dos*.

De este modo, de acuerdo con ANSI/SPARC, habría los tres niveles de esquemas que mencionamos a continuación:

- a. En el nivel externo se sitúan las diferentes visiones lógicas que los procesos usuarios (programas de aplicación y usuarios directos) tendrán de las partes de la BD que utilizarán. Estas visiones se denominan esquemas externos.
- b. En el nivel conceptual hay una sola descripción lógica básica, única y global, que denominamos esquema conceptual, y que sirve de referencia para el resto de los esquemas.
- c. En el nivel físico hay una sola descripción física, que denominamos esquema interno.

MySQL_ Introducción a las bases de datos



En el esquema conceptual se describirán las entidades tipo, sus atributos, las interrelaciones y las restricciones o reglas de integridad.

El esquema conceptual corresponde a las necesidades del conjunto de la empresa o del SI, por lo que se escribirá de forma centralizada durante el denominado diseño lógico de la BD.

Sin embargo, cada aplicación podrá tener su visión particular, y seguramente parcial, del esquema conceptual. Los usuarios (programas o usuarios directos) verán la BD mediante esquemas externos apropiados a sus necesidades. Estos esquemas se pueden considerar redefiniciones del esquema conceptual, con las partes y los términos que convengan para las necesidades de las aplicaciones (o grupos de aplicaciones). Algunos sistemas los denominan subesquemas.

Al definir un esquema externo, se citarán sólo aquellos atributos y aquellas entidades que interesen; los podremos red denominar, podremos definir datos derivados o redefinir una entidad para que las aplicaciones que utilizan este esquema externo creen que son dos, definir combinaciones de entidades para que parezcan una sola, etc.

Ejemplo de esquema externo

Imaginemos una BD que en el esquema conceptual tiene definida, entre muchas otras, una entidad alumno con los siguientes atributos: numatri, nombre, apellido, numDNI, direccion, fechanac, telefono. Sin embargo, nos puede interesar que unos determinados programas o usuarios vean la

MySQL_ Introducción a las bases de datos

BD formada de acuerdo con un esquema externo que tenga definidas dos entidades, denominadas estudiante y persona.

- a) La entidad estudiante podría tener definido el atributo numero-matricula (definido como derivable directamente de numatri), el atributo nombre-pila (de nombre), el atributo apellido y el atributo DNI (de numDNI).
- b) La entidad persona podría tener el atributo DNI (obtenido de numDNI), el atributo nombre (formado por la concatenación de nombre y apellido), el atributo direccion y el atributo edad (que deriva dinámicamente de fechanac).

El esquema interno o físico contendrá la descripción de la organización física de la BD: caminos de acceso (índices, hashing, apuntadores, etc.), codificación de los datos, gestión del espacio, tamaño de la página, etc.

El esquema de nivel interno responde a las cuestiones de rendimiento (espacio y tiempo) planteadas al hacer el diseño físico de la BD y al ajustarlo* posteriormente a las necesidades cambiantes.

De acuerdo con la arquitectura ANSI/SPARC, para crear una BD hace falta definir previamente su esquema conceptual, definir como mínimo un esquema externo y, de forma eventual, definir su esquema interno. Si este último esquema no se define, el mismo SGBD tendrá que decidir los detalles de la organización física. El SGBD se encargará de hacer las correspondencias (mappings) entre los tres niveles de esquemas.

Esquemas y niveles en los SGBD relacionales

En los SGBD relacionales (es decir, en el mundo de SQL) se utiliza una terminología ligeramente diferente. No se separan de forma clara tres niveles de descripción. Se habla de un solo esquema –schema–, pero en su interior se incluyen descripciones de los tres niveles. En el schema se describen los elementos de aquello que en la arquitectura ANSI/SPARC se denomina esquema conceptual (entidades tipo, atributos y restricciones) y las vistas –view–, que corresponden aproximadamente a los esquemas externos.

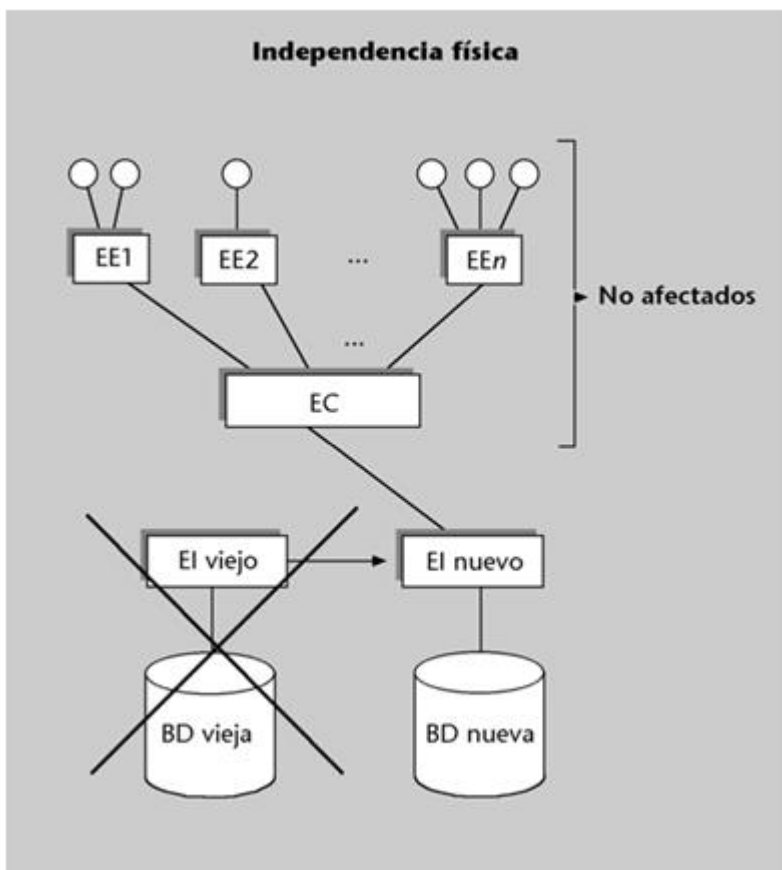
El modelo relacional en que está inspirado SQL se limita al mundo lógico. Por ello, el estándar ANSI-ISO de SQL no habla en absoluto del mundo físico o interno; lo deja en manos de los SGBD relacionales del mercado. Sin embargo, estos SGBD proporcionan la posibilidad de incluir dentro del schema descripciones de estructuras y características físicas (índice, tablespace, cluster, espacios para excesos, etc.).

MySQL_ Introducción a las bases de datos

3.2. Independencia de los datos

En este subapartado veremos cómo la arquitectura de tres niveles que acabamos de presentar nos proporciona los dos tipos de independencia de los datos: la física y la lógica.

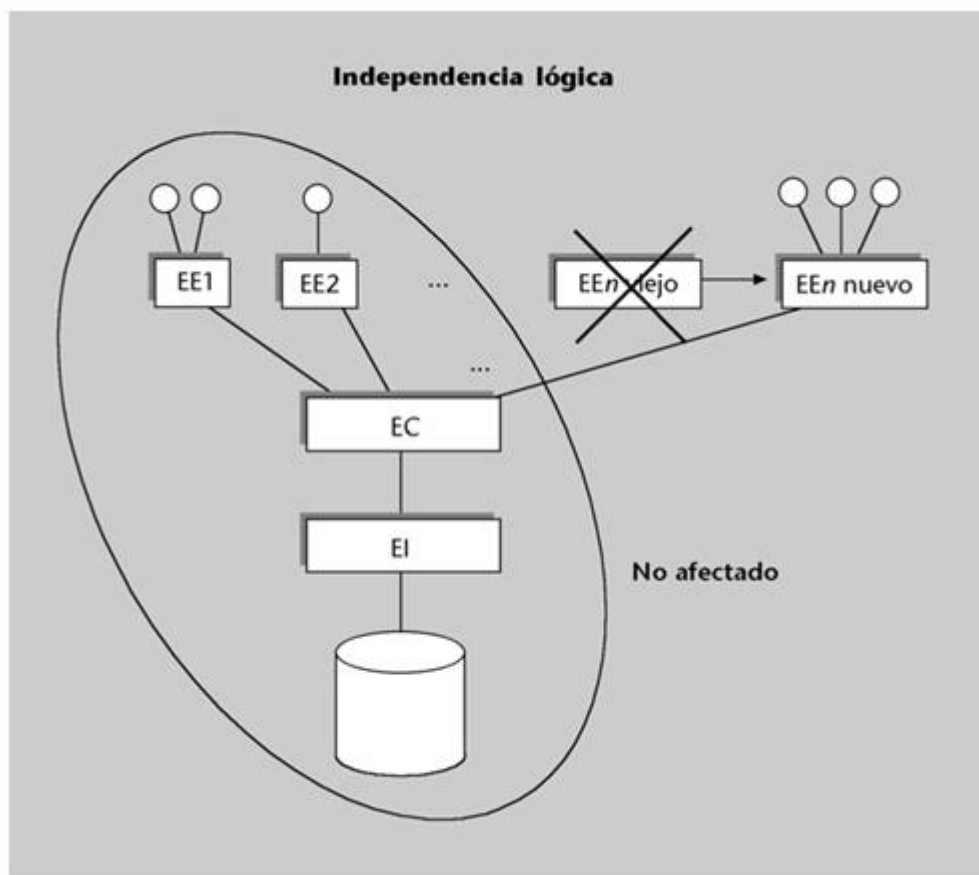
De acuerdo con la arquitectura ANSI/SPARC, habrá independencia física cuando los cambios en el esquema interno no afecten al esquema conceptual ni a los esquemas externos.



Es obvio que cuando cambiamos unos datos de un soporte a otro, o los cambiamos de lugar dentro de un soporte, no se verán afectados ni los programas de aplicación ni los usuarios directos, ya que no se modificará el esquema conceptual ni el externo. Sin embargo, tampoco tendrían que verse afectados si cambiásemos, por ejemplo, el método de acceso a unos registros determinados*, el formato o la codificación, etc. Ninguno de estos casos debería afectar al mundo exterior, sino sólo a la BD física, el esquema interno, etc.

Si hay independencia física de los datos, lo único que variará al cambiar el esquema interno son las correspondencias entre el esquema conceptual y el interno. Obviamente, la mayoría de los cambios del esquema interno obligarán a rehacer la BD real (la física).

MySQL_ Introducción a las bases de datos



Dados los dos niveles lógicos de la arquitectura ANSI/SPARC, diferenciaremos las dos situaciones siguientes:

- 1) Cambios en el esquema conceptual. Un cambio de este tipo no afectará a los esquemas externos que no hagan referencia a las entidades o a los atributos modificados.
- 2) Cambios en los esquemas externos. Efectuar cambios en un esquema externo afectará a los usuarios que utilicen los elementos modificados. Sin embargo, no debería afectar a los demás usuarios ni al esquema conceptual, y tampoco, en consecuencia, al esquema interno y a la BD física.

Usuarios no afectados por los cambios

Notad que no todos los cambios de elementos de un esquema externo afectarán a sus usuarios. Veamos un ejemplo de ello: antes hemos visto que cuando eliminábamos el atributo apellido del esquema conceptual, debíamos modificar el esquema externo donde definíamos nombre, porque allí estaba definido como concatenación de nombre y apellido. Pues bien, un programa que utilizase el atributo nombre no se vería afectado si modificásemos el esquema externo de modo que nombre fuese la concatenación de nombre y una cadena constante (por ejemplo, toda en blanco). Como resultado, habría desaparecido el apellido de nombre, sin que hubiera sido necesario modificar el programa.

MySQL_ Introducción a las bases de datos

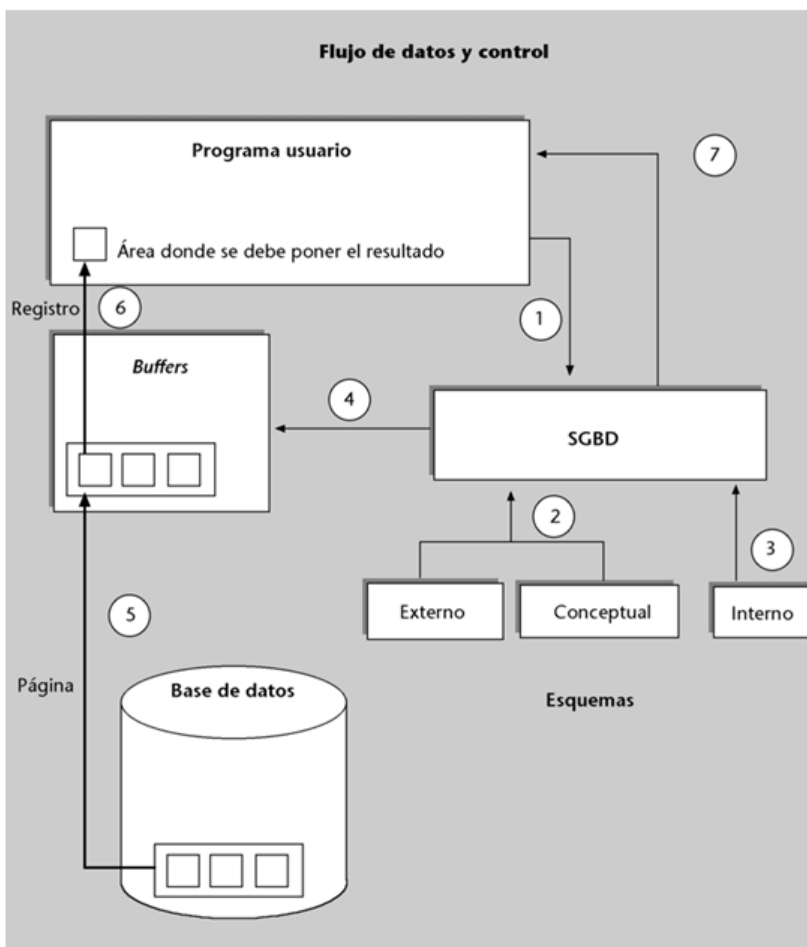
Los SGBD actuales proporcionan bastante independencia lógica, pero menos de la que haría falta, ya que las exigencias de cambios constantes en el SI piden grados muy elevados de flexibilidad. Los sistemas de ficheros tradicionales, en cambio, no ofrecen ninguna independencia lógica.

3.3. Flujo de datos y de control

Para entender el funcionamiento de un SGBD, a continuación veremos los principales pasos de la ejecución de una consulta sometida al SGBD por un programa de aplicación. Explicaremos las líneas generales del flujo de datos y de control entre el SGBD, los programas de usuario y la BD.

Recordad que el SGBD, con la ayuda del SO, lee páginas (bloques) de los soportes donde está almacenada la BD física, y las lleva a un área de buffers o memorias caché en la memoria principal. El SGBD pasa registros desde los buffers hacia el área de trabajo del mismo programa.

Supongamos que la consulta pide los datos del alumno que tiene un determinado DNI. Por lo tanto, la respuesta que el programa obtendrá será un solo registro y lo recibirá dentro de un área de trabajo propia*.



MySQL_ Introducción a las bases de datos

El proceso que se sigue es el siguiente:

- a. Empieza con una llamada (1) del programa al SGBD, en la que se le envía la operación de consulta. El SGBD debe verificar que la sintaxis de la operación recibida sea correcta, que el usuario del programa esté autorizado a hacerla, etc. Para poder llevar a cabo todo esto, el SGBD se basa (2) en el esquema externo con el que trabaja el programa y en el esquema conceptual.
- b. Si la consulta es válida, el SGBD determina, consultando el esquema interno (3), qué mecanismo debe seguir para responderla. Ya sabemos que el programa usuario no dice nada respecto a cómo se debe hacer físicamente la consulta. Es el SGBD el que lo debe determinar. Casi siempre hay varias formas y diferentes caminos para responder a una consulta*. Supongamos que ha elegido aplicar un hashing al valor del DNI, que es el parámetro de la consulta, y el resultado es la dirección de la página donde se encuentra (entre muchos otros) el registro del alumno buscado.
- c. Cuando ya se sabe cuál es la página, el SGBD comprobará (4) si por suerte esta página ya se encuentra en aquel momento en el área de los buffers (tal vez como resultado de una consulta anterior de este usuario o de otro). Si no está, el SGBD, con la ayuda del SO, la busca en disco y la carga en los buffers (5). Si ya está, se ahorra el acceso a disco.
- d. Ahora, la página deseada ya está en la memoria principal. El SGBD extrae, de entre los distintos registros que la página puede contener, el registro buscado, e interpreta la codificación y el resultado según lo que diga el esquema interno.
- e. El SGBD aplica a los datos las eventuales transformaciones lógicas que implica el esquema externo (tal vez cortando la dirección por la derecha) y las lleva al área de trabajo del programa (6).
- f. A continuación, el SGBD retorna el control al programa (7) y da por terminada la ejecución de la consulta.

MySQL_ Introducción a las bases de datos

4. Modelos de BD

Una BD es una representación de la realidad (de la parte de la realidad que nos interesa en nuestro SI). Dicho de otro modo, una BD se puede considerar un modelo de la realidad. El componente fundamental utilizado para modelar en un SGBD relacional son las tablas (denominadas relaciones en el mundo teórico). Sin embargo, en otros tipos de SGBD se utilizan otros componentes.

El conjunto de componentes o herramientas conceptuales que un SGBD proporciona para modelar recibe el nombre de modelo de BD. Los cuatro modelos de BD más utilizados en los SI son el modelo relacional, el modelo jerárquico, el modelo en red y el modelo relacional con objetos.

Todo modelo de BD nos proporciona tres tipos de herramientas:

- Estructuras de datos con las que se puede construir la BD: tablas, árboles, etc.
- Diferentes tipos de restricciones (o reglas) de integridad que el SGBD tendrá que hacer cumplir a los datos: dominios, claves, etc.
- Una serie de operaciones para trabajar con los datos. Un ejemplo de ello, en el modelo relacional, es la operación SELECT, que sirve para seleccionar (o leer) las filas que cumplen alguna condición. Un ejemplo de operación típica del modelo jerárquico y del modelo en red podría ser la que nos dice si un determinado registro tiene “hijos” o no.

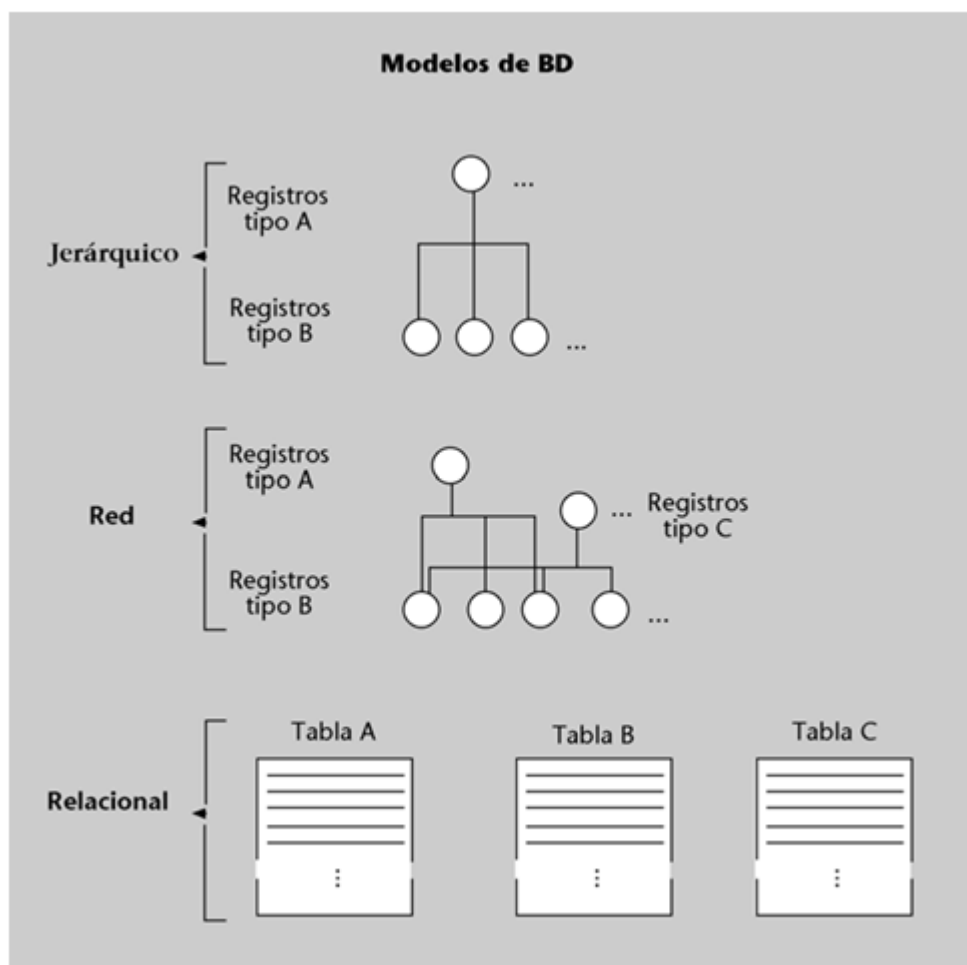
Evolución de los modelos de BD

De los cuatro modelos de BD que hemos citado, el que apareció primero, a principios de los años sesenta, fue el modelo jerárquico. Sus estructuras son registros interrelacionados en forma de árboles. El SGBD clásico de este modelo es el IMS/DL1 de IBM.

A principios de los setenta surgieron SGBD basados en un modelo en red. Como en el modelo jerárquico, hay registros e interrelaciones, pero un registro ya no está limitado a ser “hijo” de un solo registro tipo. El comité CODASYLDBTG propuso un estándar basado en este modelo, que fue adoptado por muchos constructores de SGBD*. Sin embargo, encontró la oposición de IBM, la empresa entonces dominante. La propuesta de CODASYL-DBTG ya definía tres niveles de esquemas.

Durante los años ochenta apareció una gran cantidad de SGBD basados en el modelo relacional propuesto en 1969 por E.F. Codd, de IBM, y prácticamente todos utilizaban como lenguaje nativo el SQL**. El modelo relacional se basa en el concepto matemático de relación, que aquí podemos considerar de momento equivalente al término tabla (formada por filas y columnas). La mayor parte de los SI que actualmente están en funcionamiento utilizan SGBD relacionales, pero algunos siguen utilizando los jerárquicos o en red (especialmente en SI antiguos muy grandes).

MySQL_ Introducción a las bases de datos



Así como en los modelos prerrelacionales (jerárquico y en red), las estructuras de datos constan de dos elementos básicos (los registros y las interrelaciones), en el modelo relacional constan de un solo elemento: la tabla, formada por filas y columnas. Las interrelaciones se deben modelizar utilizando las tablas.

Otra diferencia importante entre los modelos prerrelacionales y el modelo relacional es que el modelo relacional se limita al nivel lógico (no hace absolutamente ninguna consideración sobre las representaciones físicas). Es decir, nos da una independencia física de datos total. Esto es así si hablamos del modelo teórico, pero los SGBD del mercado nos proporcionan una independencia limitada.

Estos últimos años se está extendiendo el modelo de BD relacional con objetos. Se trata de ampliar el modelo relacional, añadiéndole la posibilidad de que los tipos de datos sean tipos abstractos de datos, TAD. Esto acerca los sistemas relacionales al paradigma de la OO. Los primeros SGBD relacionales que dieron esta posibilidad fueron Oracle (versión 8), Informix (versión 9) e IBM/DB2/UDB (versión 5).

MySQL_ Introducción a las bases de datos

Hablamos de modelos de BD, pero de hecho se acostumbran a denominar modelos de datos, ya que permiten modelarlos. Sin embargo, hay modelos de datos que no son utilizados por los SGBD del mercado: sólo se usan durante el proceso de análisis y diseño, pero no en las realizaciones.

Los más conocidos de estos tipos de modelos son los modelos semánticos y los funcionales. Éstos nos proporcionan herramientas muy potentes para describir las estructuras de la información del mundo real, la semántica y las interrelaciones, pero normalmente no disponen de operaciones para tratarlas. Se limitan a ser herramientas de descripción lógica. Son muy utilizados en la etapa del diseño de BD y en herramientas CASE. El más extendido de estos modelos es el conocido como modelo ER (entity-relationship), que estudiaremos más adelante.

Actualmente, la práctica más extendida en el mundo profesional de los desarrolladores de SI es la utilización del modelo ER durante el análisis y las primeras etapas del diseño de los datos, y la utilización del modelo relacional para acabar el diseño y construir la BD con un SGBD.

En esta asignatura hablamos sólo de BD con modelos de datos estructurados, que son los que normalmente se utilizan en los SI empresariales. Sin embargo, hay SGBD especializados en tipos de aplicaciones concretas que no siguen ninguno de estos modelos. Por ejemplo, los SGBD documentales o los de BD geográficas.

MySQL_ Introducción a las bases de datos

5. Lenguajes y usuarios

Para comunicarse con el SGBD, el usuario, ya sea un programa de aplicación o un usuario directo, se vale de un lenguaje. Hay muchos lenguajes diferentes, según el tipo de usuarios para los que están pensados y el tipo de cosas que los usuarios deben poder expresar con ellos:

- a. Habrá usuarios informáticos muy expertos que querrán escribir procesos complejos y que necesitarán lenguajes complejos.
- b. Sin embargo, habrá usuarios finales no informáticos, ocasionales (esporádicos), que sólo harán consultas. Estos usuarios necesitarán un lenguaje muy sencillo, aunque dé un rendimiento bajo en tiempo de respuesta.
- c. También podrá haber usuarios finales no informáticos, dedicados o especializados. Son usuarios cotidianos o, incluso, dedicados exclusivamente a trabajar con la BD*. Estos usuarios necesitarán lenguajes muy eficientes y compactos, aunque no sea fácil aprenderlos. Tal vez serán lenguajes especializados en tipos concretos de tareas.

El lenguaje SQL, que es el más utilizado en las BD relacionales, tiene verbos –instrucciones– de tres tipos diferentes:

1. Verbos del tipo DML; por ejemplo, SELECT para hacer consultas, e INSERT, UPDATE y DELETE para hacer el mantenimiento de los datos.
2. Verbos del tipo DDL; por ejemplo, CREATE TABLE para definir las tablas, sus columnas y las restricciones.
3. Además, SQL tiene verbos de control del entorno, como por ejemplo

COMMIT y ROLLBACK para delimitar transacciones.

En cuanto a los aspectos DML, podemos diferenciar dos tipos de lenguajes:

- a. Lenguajes muy declarativos (o implícitos), con los que se especifica qué se quiere hacer sin explicar cómo se debe hacer.
- b. Lenguajes más explícitos o procedimentales, que nos exigen conocer más cuestiones del funcionamiento del SGBD para detallar paso a paso cómo se deben realizar las operaciones (lo que se denomina navegar por la BD).

Como es obvio, los aspectos DDL (las descripciones de los datos) son siempre declarativos por su propia naturaleza.

MySQL_ Introducción a las bases de datos

Los lenguajes utilizados en los SGBD prerrelacionales eran procedimentales. SQL es básicamente declarativo, pero tiene posibilidades procedimentales.

Tanto los 4GL como las herramientas visuales (con frecuencia unidas en una sola herramienta) traducen lo que hace el usuario a instrucciones SQL por distintas vías:

- En el caso de los 4GL, la traducción se suele hacer mediante la compilación.
- En el caso de las herramientas visuales, se efectúa por medio del intérprete de SQL integrado en el SGBD.

Si queremos escribir un programa de aplicación que trabaje con BD, seguramente querremos utilizar nuestro lenguaje habitual de programación*. Sin embargo, generalmente estos lenguajes no tienen instrucciones para realizar el acceso a las BD. Entonces tenemos las dos opciones siguientes:

1. Las llamadas a funciones: en el mercado hay librerías de funciones especializadas en BD (por ejemplo, las librerías ODBC). Sólo es preciso incluir llamadas a las funciones deseadas dentro del programa escrito con el lenguaje habitual. Las funciones serán las que se encargarán de enviar las instrucciones (generalmente en SQL) en tiempo de ejecución al SGBD.
2. El lenguaje hospedado: otra posibilidad consiste en incluir directamente las instrucciones del lenguaje de BD en nuestro programa. Sin embargo, esto exige utilizar un precompilador especializado que acepte en nuestro lenguaje de programación habitual las instrucciones del lenguaje de BD. Entonces se dice que este lenguaje (casi siempre SQL) es el lenguaje hospedado o incorporado (embedded), y nuestro lenguaje de programación (Pascal, C, Cobol, etc.) es el lenguaje anfitrión (host).

MySQL_ Introducción a las bases de datos

6. Administración de BD

Los administradores de BD son los responsables del correcto funcionamiento de la BD y velan para que siempre se mantenga útil. Intervienen en situaciones problemáticas o de emergencia, pero su responsabilidad fundamental es velar para que no se produzcan incidentes.

A continuación damos una lista de tareas típicas del ABD:

1. Mantenimiento, administración y control de los esquemas. Comunicación de los cambios a los usuarios.
2. Asegurar la máxima disponibilidad de los datos; por ejemplo, haciendo copias (back-ups), administrando diarios (journals o logs), reconstruyendo la BD, etc.
3. Resolución de emergencias.
4. Vigilancia de la integridad y de la calidad de los datos.
5. Diseño físico, estrategia de caminos de acceso y reestructuraciones.
6. Control del rendimiento y decisiones relativas a las modificaciones en los esquemas y/o en los parámetros del SGBD y del SO, para mejorarlo.
7. Normativa y asesoramiento a los programadores y a los usuarios finales sobre la utilización de la BD.
8. Control y administración de la seguridad: autorizaciones, restricciones, etc.

La tarea del ABD no es sencilla.

Los SGBD del mercado procuran reducir al mínimo el volumen de estas tareas, pero en sistemas muy grandes y críticos se llega a tener grupos de ABD de más de diez personas. Buena parte del software que acompaña el SGBD está orientado a facilitar la gran diversidad de tareas controladas por el ABD: monitores del rendimiento, monitores de la seguridad, verificadores de la consistencia entre índices y datos, reorganizadores, gestores de las copias de seguridad, etc. La mayoría de estas herramientas tienen interfaces visuales para facilitar la tarea del ABD.

MySQL_ Introducción a las bases de datos

Resumen

En esta unidad hemos hecho una introducción a los conceptos fundamentales del mundo de las BD y de los SGBD. Hemos explicado la evolución de los SGBD, que ha conducido de una estructura centralizada y poco flexible a una distribuida y flexible, y de una utilización procedimental que requería muchos conocimientos a un uso declarativo y sencillo.

Hemos revisado los objetivos de los SGBD actuales y algunos de los servicios que nos dan para conseguirlos. Es especialmente importante el concepto de transacción y la forma en que se utiliza para velar por la integridad de los datos.

La arquitectura de tres niveles aporta una gran flexibilidad a los cambios, tanto a los físicos como a los lógicos. Hemos visto cómo un SGBD puede funcionar utilizando los tres esquemas propios de esta arquitectura.

Hemos explicado que los componentes de un modelo de BD son las estructuras, las restricciones y las operaciones. Los diferentes modelos de BD se diferencian básicamente por sus estructuras. Hemos hablado de los modelos más conocidos, especialmente del modelo relacional, que está basado en tablas y que estudiaremos más adelante.

Cada tipo de usuario del SGBD puede utilizar un lenguaje apropiado para su trabajo. Unos usuarios con una tarea importante y difícil son los administradores de las BD.

7. ¿Qué tipos de base de datos existen?

Existen diferentes tipos de bases de datos, que se diferencian por su estructura, funcionamiento y la forma en que se almacenan y organizan los datos. A continuación, se describen algunos de los tipos de bases de datos más comunes:

- **Bases de datos relacionales:** son las más utilizadas y se basan en el modelo relacional. La información se organiza en tablas con filas y columnas, y se relacionan mediante claves primarias y foráneas. Ejemplos de bases de datos relacionales son MySQL, Oracle y SQL Server.
- **Bases de datos NoSQL:** son aquellas que no utilizan el modelo relacional, sino que se basan en otros tipos de estructuras de datos, como documentos, grafos o columnas. Estas bases de datos se utilizan en aplicaciones web y móviles, y se caracterizan por su escalabilidad y flexibilidad. Ejemplos de bases de datos NoSQL son MongoDB, Cassandra y Neo4j.
- **Bases de datos orientadas a objetos:** se utilizan para almacenar y manipular objetos en lugar de datos. Son útiles para aplicaciones que requieren la gestión de objetos complejos y su relación con otros objetos. Ejemplos de bases de datos orientadas a objetos son ObjectStore y ObjectDB.
- **Bases de datos en memoria:** son aquellas que almacenan los datos en la memoria RAM en lugar de en discos duros o en otros dispositivos de almacenamiento. Esto permite un acceso mucho más rápido a los datos y se utilizan en aplicaciones que requieren un alto rendimiento. Ejemplos de bases de datos en memoria son Redis y Memcached.
- **Bases de datos espaciales:** son aquellas que permiten la gestión de datos relacionados con la ubicación geográfica. Estas bases de datos se utilizan en aplicaciones de geolocalización y análisis de datos geoespaciales. Ejemplos de bases de datos espaciales son PostGIS y Oracle Spatial.
- **Bases de datos de series temporales:** se utilizan para almacenar y analizar datos que cambian con el tiempo, como mediciones de sensores, datos climáticos y financieros. Estas bases de datos se caracterizan por su capacidad para manejar grandes volúmenes de datos y realizar consultas complejas en series temporales. Ejemplos de bases de datos de series temporales son InfluxDB y TimescaleDB.

MySQL_ Introducción a las bases de datos

8. Sistema Gestor de Bases de Datos

Un **Sistema Gestor de Base de Datos (SGBD)** o DataBase Managenent System (**DBMS**) es un sistema que permite la **creación, gestión y administración de bases de datos**, así como la **elección y manejo de las estructuras necesarias** para el almacenamiento y búsqueda de información del modo más eficiente posible.

En la actualidad, existen multitud de SGBD y pueden ser clasificados según la forma en que administran los datos en:

- Relacionales (SQL)
- No relacionales (NoSQL)

A lo largo de este tema vamos a mostrar los principales sistemas gestores de bases de datos más usados de cada tipo.

8.1. Sistemas Gestores de bases de datos Relacionales (SQL)

Estructura de las bases de datos relacionales

El modelo relacional implica que las estructuras lógicas de los datos (las tablas, las vistas y los índices) estén separadas de las estructuras de almacenamiento físico. Gracias a esta separación, los administradores de bases de datos pueden gestionar el almacenamiento físico de datos sin que eso influya en el acceso a esos datos como estructura lógica. Por ejemplo, si se cambia el nombre del archivo de una base de datos, eso no significa que vayan a cambiar también los nombres de sus tablas.

La distinción entre estructura lógica y física también se aplica a las operaciones de base de datos: acciones claramente definidas que permiten a las aplicaciones gestionar los datos y las estructuras de la base de datos. Con las operaciones lógicas, las aplicaciones pueden especificar el contenido que necesitan, mientras que las operaciones físicas determinan cómo se debe acceder a esos datos y llevan a cabo la tarea.

Para garantizar la precisión y accesibilidad continua de los datos, las bases de datos relacionales siguen ciertas reglas de integridad. Por ejemplo, una regla de integridad podría especificar que no se permite duplicar filas en una tabla, a fin de evitar que se introduzca información errónea en la base de datos.

El modelo relacional

En los primeros años de las bases de datos, cada aplicación almacenaba datos en su propia estructura única. Cuando los desarrolladores querían crear aplicaciones para usar esos datos, tenían que conocer muy bien esa estructura de datos concreta a fin de encontrar los datos que necesitaban. Esas estructuras de datos eran poco eficaces, el mantenimiento era complicado y era difícil optimizarlas para ofrecer un buen rendimiento en las aplicaciones. El modelo de base de datos relacional se diseñó para resolver el problema causado por estructuras de datos múltiples y arbitrarias.

MySQL_ Introducción a las bases de datos

El modelo relacional proporcionó una forma estándar de representar y consultar datos que podía utilizarse en cualquier aplicación. Desde el principio, los desarrolladores se dieron cuenta de que la virtud principal del modelo de base de datos relacional era el uso de tablas, ya que era una forma intuitiva, eficiente y flexible de almacenar y acceder a información estructurada.

Con el tiempo, los desarrolladores comenzaron a usar el lenguaje de consulta estructurado (SQL) para escribir y hacer consultas en una base de datos: esto sería otra de las grandes virtudes de este modelo. Durante muchos años, el SQL se ha utilizado como el lenguaje para realizar consultas en bases de datos. Se basa en el álgebra relacional y proporciona un lenguaje matemático de uniformidad interna que facilita la mejora del rendimiento de todas las consultas en bases de datos. Otros métodos empleados necesitan definir consultas individuales.

Ventajas de las bases de datos relacionales

El modelo relacional es sencillo pero muy potente, y lo utilizan organizaciones de todos los tipos y tamaños para una gran variedad de aplicaciones con datos. Las bases de datos relacionales se usan para rastrear inventarios, procesar transacciones de comercio electrónico, administrar cantidades enormes y esenciales de información de clientes y mucho más. Las bases de datos relacionales se pueden emplear para cualquier aplicación de datos en la que los puntos de datos se relacionen entre sí y deban gestionarse de forma segura, conforme a normas y de un modo uniforme.

Las bases de datos relacionales han existido desde la década de los setenta. En la actualidad, el modelo relacional sigue siendo el más aceptado para las bases de datos, gracias a todas sus virtudes.

Uniformidad de los datos

El modelo relacional es el ideal para mantener la uniformidad de los datos en todas las aplicaciones y copias de la base de datos (llamadas *instancias*). Por ejemplo, cuando un cliente deposita dinero en un cajero automático y, a continuación, mira el saldo en un teléfono móvil, el cliente espera ver ese depósito reflejado inmediatamente. Las bases de datos relacionales son perfectas para este tipo de uniformidad, y garantizan que todas las instancias de una base de datos tengan los mismos datos en todo momento.

Es muy difícil que otros tipos de bases de datos mantengan este nivel de concordancia tan inmediato con grandes cantidades de datos. Algunas bases de datos recientes, como NoSQL, solo pueden suministrar una “uniformidad final”. Según este principio, cuando la base de datos se adapta o cuando varios usuarios acceden a los mismos datos al mismo tiempo, los datos necesitan tiempo para “actualizarse”. La uniformidad final es aceptable en algunos casos (por ejemplo, para mantener listas en un catálogo de productos), pero para operaciones comerciales críticas, como transacciones de carritos de compra, la base de datos relacional sigue siendo la referencia.

MySQL_ Introducción a las bases de datos

Compromiso y atomicidad

Las bases de datos relacionales gestionan las reglas y políticas comerciales a un nivel muy detallado, y tienen políticas estrictas sobre el *compromiso* (es decir, el establecimiento de un cambio en la base de datos como algo permanente). Por ejemplo, imaginemos una base de datos de inventario para rastrear tres piezas que siempre se usan juntas. Cuando se saca una pieza del inventario, las otras dos también deben salir. Si una de las tres piezas no está disponible, ninguna de ellas debe salir del inventario; las tres deben estar disponibles antes de que la base de datos establezca el compromiso. Una base de datos relacional no se comprometerá sobre una pieza hasta que pueda comprometerse sobre las tres. Esta capacidad de compromiso multifacético se llama *atomicidad*. La atomicidad es la clave para mantener los datos precisos y garantizar el cumplimiento de las reglas, normas y políticas de la empresa.

Principales Bases de Datos relacionales

Desde que se comenzó a usar el modelo de **bases de datos relacionales**, en 1970, ha ido sufriendo una serie de transformaciones hasta convertirse, hoy en día, en el **modelo más utilizado** para administrar bases de datos.

Este modelo se basa fundamentalmente en establecer **relaciones o vínculos** entre los datos, imaginando una tabla aparte por cada relación existente con sus propios registros y atributos.

Los principales Sistemas gestores de bases de datos relacionales (**SGBD SQL**) actualmente son:

MySQL



Es el sistema gestor de bases de datos relacional por excelencia. Es un SGBD **multihilo y multiusuario** utilizado en la gran parte de las páginas web actuales. Además es el más usado en aplicaciones creadas como software libre.

Se ofrece bajo la GNU GPL aunque también es posible adquirir una licencia para empresas que quieran incorporarlo en productos privativos (Desde la compra por parte de Oracle se está orientando a este ámbito empresarial).

Las principales **ventajas** de este Sistema Gestor de Bases de datos son:

- Facilidad de uso y gran rendimiento
- Facilidad para instalar y configurar
- Soporte multiplataforma
- Soporte SSL

La principal **desventaja** es la escalabilidad, es decir, no trabaja de manera eficiente con bases de datos muy grandes que superan un determinado tamaño.

MySQL_ Introducción a las bases de datos

[MariaDB](#)



Este SGBD es una **derivación de MySQL** que cuenta con la mayoría de características de este e incluye varias extensiones. Nace a partir de la adquisición de MySQL por parte de Oracle para seguir la filosofía **Open Source** y tiene la ventaja de que es totalmente compatible con MySQL.

Entre las principales **características** de este Sistema Gestor de Bases de datos se encuentran:

- Aumento de motores de almacenamiento
- Gran escalabilidad
- Seguridad y rapidez en transacciones
- Extensiones y nuevas características relacionadas con su aplicación para Bases de datos NoSQL.

No tiene desventajas muy aparentes salvo algunas pequeñas incompatibilidades en la migración de MariaDB y MySQL o pequeños atrasos en la liberación de versiones estables.

[SQLite](#)



Más que un Sistema Gestor de bases de datos como tal, SQLite es una **biblioteca** escrita en C **que implementa un SGBD** y que permite transacciones sin necesidad de un servidor ni configuraciones.

Es una biblioteca utilizada en multitud de aplicaciones actuales ya que es **open source** y las consultas son muy eficientes.

Las principales **características** de SQLite son:

- El tamaño, al tratarse de una biblioteca, es mucho menor que cualquier SGBD
- Reúne los cuatro criterios ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) logrando gran estabilidad
- Gran portabilidad y rendimiento

La gran **desventaja** de SQLite es la escalabilidad ya que no soporta bases de datos que sean muy grandes.

MySQL_ Introducción a las bases de datos

PostgreSQL



Este sistema gestor de base de datos relacional está **orientado a objetos** y es libre, publicado bajo la licencia BSD.

Sus principales **características** son:

- Control de Concurrencias multiversión (MVCC)
- Flexibilidad en cuanto a lenguajes de programación
- Multiplataforma
- Dispone de una herramienta (pgAdmin, <https://www.pgadmin.org/>) muy fácil e intuitiva para la administración de las bases de datos.
- Robustez, Eficiencia y Estabilidad.

La principal **desventaja** es la lentitud para la administración de bases de datos pequeñas ya que está optimizado para gestionar grandes volúmenes de datos.

MySQL_ Introducción a las bases de datos

Microsoft SQL Server



Es un sistema gestor de bases de datos relacionales basado en el lenguaje **Transact-SQL**, capaz de poner a disposición de muchos usuarios grandes cantidades de datos de manera simultánea.

Es un sistema propietario de **Microsoft**. Sus principales **características** son:

- Soporte exclusivo por parte de Microsoft.
- Escalabilidad, estabilidad y seguridad.
- Posibilidad de cancelar consultas.
- Potente entorno gráfico de administración que permite utilizar comandos DDL y DML.
- Aunque es nativo para Windows puede utilizarse desde hace ya un tiempo en otras plataformas como Linux o Docker.

Su principal **desventaja** es el precio. Cuenta con un plan gratuito (Express) pero lo normal es la elección de alguno de los **planes de pago** disponibles (Standard, Developer, Enterprise o SQL Azure, la versión de SQL Server en la nube).

MySQL_ Introducción a las bases de datos

[Oracle](#)



Tradicionalmente, Oracle ha sido el **SGBD por excelencia para el mundo empresarial**, considerado siempre como el más **completo y robusto**, destacando por:

- Soporte de transacciones.
- Estabilidad.
- Escalabilidad.
- Multiplataforma.

La principal **desventaja**, al igual que SQL Server, es el coste del software ya que, aunque cuenta con una versión gratuita (Express Edition o XE), sus principales opciones son de pago.

MySQL_ Introducción a las bases de datos

8.2. Sistemas Gestores de bases de datos No Relacionales (NoSQL)

Una base de datos NoSQL (Not only SQL) es una base de datos que no utiliza el modelo relacional de tablas y relaciones utilizado por las bases de datos relacionales tradicionales. En cambio, utiliza modelos de datos alternativos, como documentos, grafos, columnas o llave-valor, que permiten una mayor flexibilidad y escalabilidad.

Las bases de datos NoSQL se han vuelto populares en los últimos años debido a su capacidad para manejar grandes volúmenes de datos no estructurados o semiestructurados, y para funcionar en entornos distribuidos y en la nube. Estas bases de datos se utilizan en una amplia variedad de aplicaciones, desde redes sociales hasta sistemas de comercio electrónico, análisis de big data y aplicaciones móviles.

Algunos ejemplos de bases de datos NoSQL populares incluyen MongoDB, Cassandra, Redis, Couchbase, Neo4j, Amazon DynamoDB y Google Cloud Datastore. Cada una de estas bases de datos NoSQL tiene sus propias características, fortalezas y debilidades, y es importante elegir la base de datos adecuada para el tipo de datos y la aplicación específicos que se estén utilizando. Para la administración de este tipo de bases de datos, actualmente los principales sistemas gestores de bases de datos (**SGBD NoSQL**) son:

[MongoDB](#)



Estamos ante el Sistema Gestor de Bases de Datos no relacionales (SGBD NoSQL) más **popular** y **utilizado** actualmente.

MongoDB es un SGBD NoSQL **orientado a ficheros** que almacena la información en **estructuras BSON** con un esquema dinámico que permite su facilidad de integración.

Empresas como **Google, Facebook, eBay, Cisco o Adobe** utilizan MongoDB como Sistema Gestor de Bases de datos.

Las principales **características** de MongoDB son:

- Indexación y replicación
- Balanceo de carga
- Almacenamiento en ficheros
- Consultas ad hoc
- Escalabilidad horizontal
- Open Source
-

Como **desventaja** principal, MongoDB no es un SGBD adecuado para realizar transacciones complejas.

MySQL_ Introducción a las bases de datos

Redis



Redis está basado en el **almacenamiento clave-valor**. Podríamos verlo como un vector enorme que almacena todo tipo de datos, desde cadenas, hashses, listas, etc.

El principal uso de este SGBD es para el **almacenamiento en memoria caché y la administración de sesiones**.

Las **características** principales son:

- Atomicidad y persistencia
- Gran velocidad
- Simplicidad
- Multiplataforma

Cassandra



Al igual que Redis, Cassandra también utiliza **almacenamiento clave-valor**. Es un SGBD NoSQL **distribuido y masivamente escalable**.

Facebook, Twitter, Instagram, Spotify o Netflix utilizan Cassandra.

Dispone de un lenguaje propio para las consultas denominado **CQL** (Cassandra Query Language).

Las principales **características** de este SGBD NoSQL son:

- Multiplataforma
- Propio lenguaje de consultas (CQL)
- Escalado lineal y horizontal
- Es un SGBD distribuido
- Utiliza una arquitectura peer-to-peer

Otros SGBD NoSQL

Otros Sistemas Gestores de bases de datos no relacionales muy utilizados son:

- [Azure Cosmos DB](#)
- [RavenDB](#)
- [ObjectDB](#)
- [Apache CouchDB](#)
- [Neo4j](#)
- [Google BigTable](#)
- [Apache Hbase](#)
- [Amazon DynamoDB](#)

MySQL_ Introducción a las bases de datos

9. MySQL y SQL

9.1. ¿Qué es SQL?

SQL es un lenguaje que se usa para operar su base de datos. SQL es el lenguaje básico utilizado para todas las bases de datos. Hay cambios de sintaxis menores entre diferentes bases de datos, pero la sintaxis SQL básica sigue siendo en gran medida la misma. SQL es una abreviatura de **Structured Query Language**. De acuerdo con ANSI (American National Standards Institute), SQL es el lenguaje estándar para operar un sistema de administración de bases de datos relacionales.

SQL se usa para acceder, actualizar y manipular datos en una base de datos. Su diseño permite la gestión de datos en un RDBMS, como MYSQL. El lenguaje SQL también se usa para controlar el acceso a datos y para la creación y modificación de esquemas de Base de datos.

9.2. ¿Qué es MYSQL?

Desarrollado a mediados de los años 90, MySQL fue una de las primeros gestores de bases de datos de código abierto disponibles en el mercado. Hoy en día existen muchas alternativas variantes de MySQL. Sin embargo, las diferencias entre las variantes no son significativas ya que usan la misma sintaxis, y la funcionalidad básica también permanece igual.

MySQL es un sistema de gestión de bases de datos relacionales (RDBMS) que permite mantener organizados los datos que existen en una base de datos. MySQL se pronuncia como “My S-Q-L”, pero también se llama “My Sequel.”. Lleva el nombre de la hija del cofundador Michael Widenius. MySQL proporciona un acceso multiusuario a las bases de datos. Este sistema RDBMS se usa con la combinación de PHP y Apache Web Server, además de una distribución de Linux. MySQL usa el lenguaje SQL para consultar la base de datos.

9.3. Diferencia entre SQL y MySQL

Parámetro	SQL	MYSQL
Definición	SQL es un lenguaje de consulta estructurado. Es útil para administrar bases de datos relacionales.	MySQL es un RDBMS para almacenar, recuperar, modificar y administrar una base de datos utilizando SQL.
Complejidad	Necesita aprender el lenguaje SQL para usarlo efectivamente.	Está disponible a través de la descarga y la instalación.
Tipo	SQL es un lenguaje de consulta.	MySQL es un software de base de datos. Usó el lenguaje “SQL” para consultar la base de datos.
Soporte para conector	SQL no proporciona conectores.	MySQL ofrece una herramienta integrada llamada ‘MySQL workbench’ para diseñar y desarrollar bases de datos.
Propósito	Para consultar y operar el sistema de base de datos.	Permite el manejo, almacenamiento, modificación y eliminación de datos en formato tabular.

MySQL_ Introducción a las bases de datos

Parámetro	SQL	MYSQL
Uso	El código SQL y los comandos se usan en varios sistemas DBMS y RDMS, incluido MYSQL.	MYSQL se usa como una base de datos RDBMS.
Actualizaciones	El lenguaje es fijo y el comando sigue siendo el mismo.	Recibe las actualizaciones frecuentes

9.4. Conclusión:

- SQL es un lenguaje que se usa para operar su base de datos
- MySQL fue una de las primeras bases de datos de código abierto disponibles en el mercado
- SQL se usa para acceder, actualizar y manipular datos en una base de datos
- MySQL es un RDBMS que permite mantener organizados los datos que existen en una base de datos
- SQL es un lenguaje de consulta estructurado
- MySQL es un RDBMS para almacenar, recuperar, modificar y administrar una base de datos utilizando MYSQL
- SQL es un lenguaje de consulta, mientras que MYSQL es un software de base de datos

MySQL_ Introducción a las bases de datos

10. Instalacion MySQL

SQL, Structure Query Language (Lenguaje de Consulta Estructurado) es un lenguaje de programación para trabajar con base de datos relacionales como MySQL, Oracle, etc.

MySQL es un interpretador de SQL, es un servidor de base de datos.

MySQL permite crear base de datos y tablas, insertar datos, modificarlos, eliminarlos, ordenarlos, hacer consultas y realizar muchas operaciones, etc., resumiendo: administrar bases de datos.

Ingresando instrucciones en la línea de comandos o embebidas en un lenguaje como PHP nos comunicamos con el servidor. Cada sentencia debe acabar con punto y coma (;).

La sensibilidad a mayúsculas y minúsculas, es decir, si hace diferencia entre ellas, depende del sistema operativo, Windows no es sensible, pero Linux si. Por ejemplo Windows interpreta igualmente las siguientes sentencias:

```
create database administracion;  
Create DataBase administracion;
```

Pero Linux interpretará como un error la segunda.

Se recomienda usar siempre minúsculas.

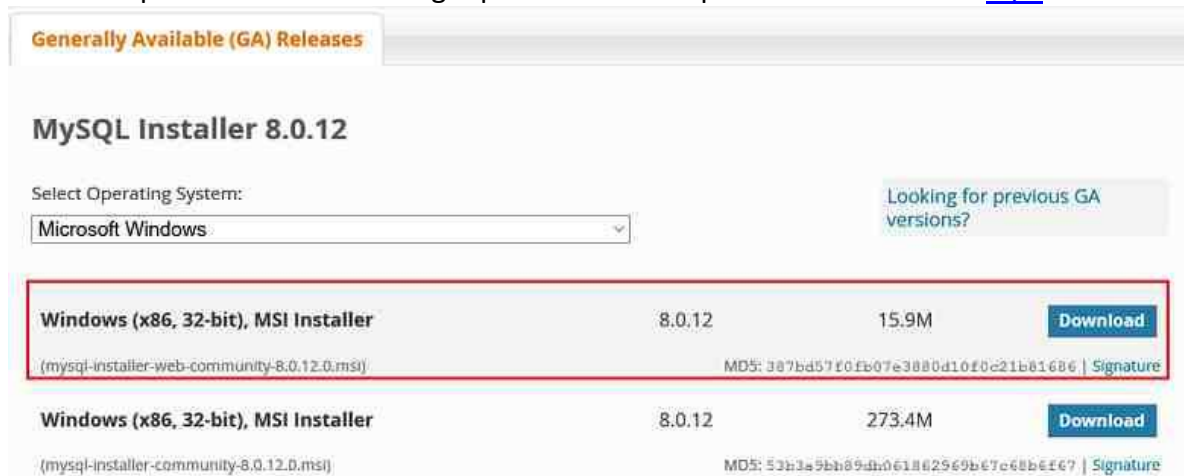
Durante este curso utilizaremos la versión "MySQL Community" que es Open Source.

Si no podemos instalar el servidor de base de datos en nuestro equipo podemos hacer cada ejercicio en este mismo sitio y probarlo, pero también podemos instalar el Servidor en nuestro equipo para probar y ejercitar en forma más cómoda.

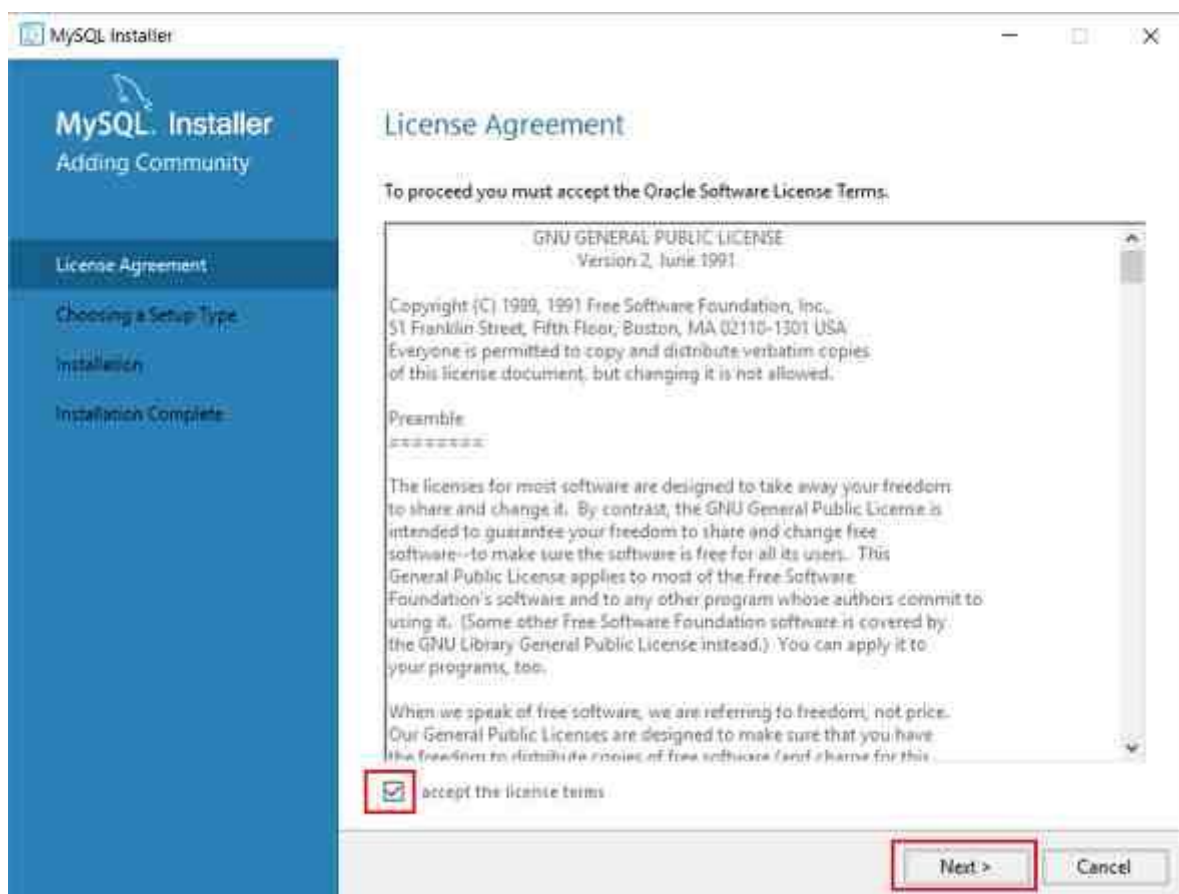
MySQL_ Introducción a las bases de datos

10.1. Instalación de "MySQL Community".

Dijimos que si queremos seguir el curso de MySQL en nuestro equipo el primer paso será instalar el software que lo debemos descargar para el sistema operativo Windows de [aquí](#).

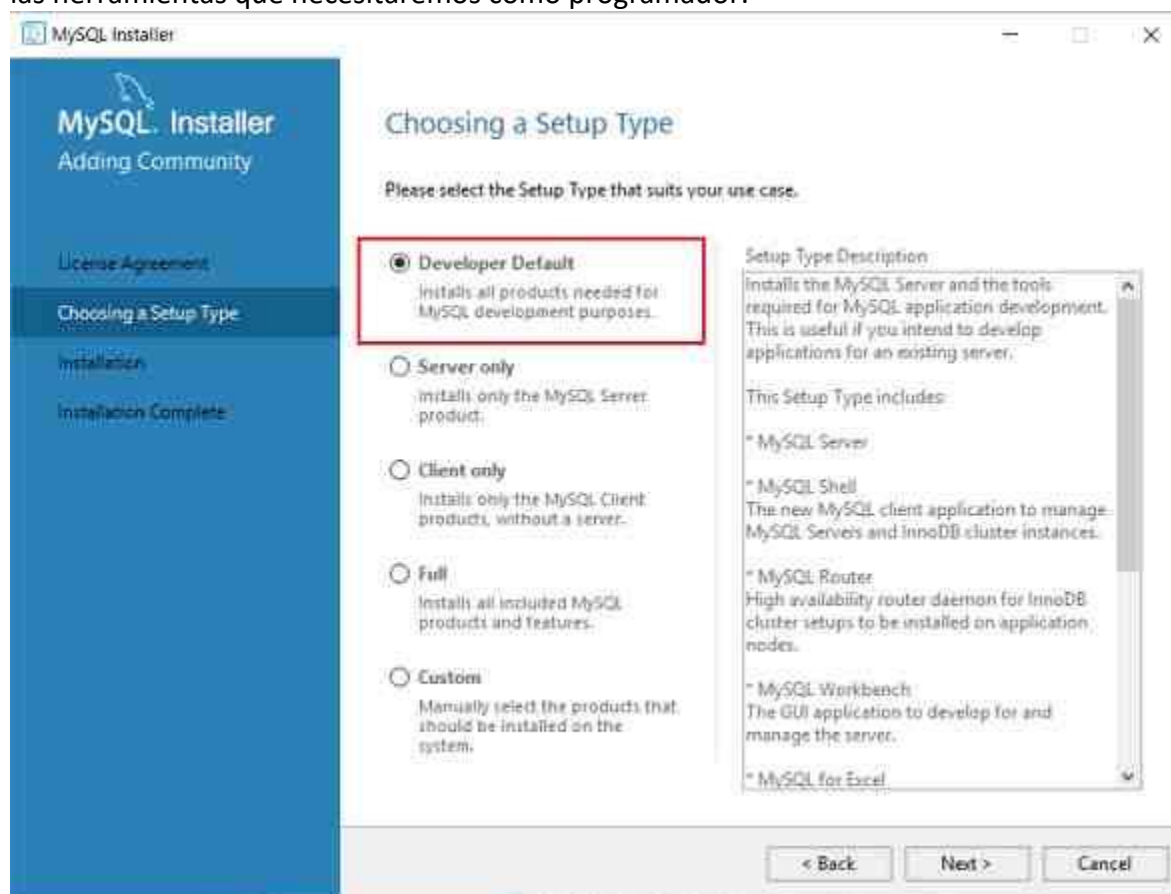


En la siguiente pantalla seleccionamos "No thanks, just start my download." y procedemos a descargar el instalador de MySQL (este programa luego nos descargará la última versión) Una vez descargado el archivo "mysql-installer-web-community-8.0.12.0.msi" (o una versión posterior) procedemos a ejecutarlo y luego de aceptar "los términos y condiciones" dejaremos la mayoría de las opciones por defecto:



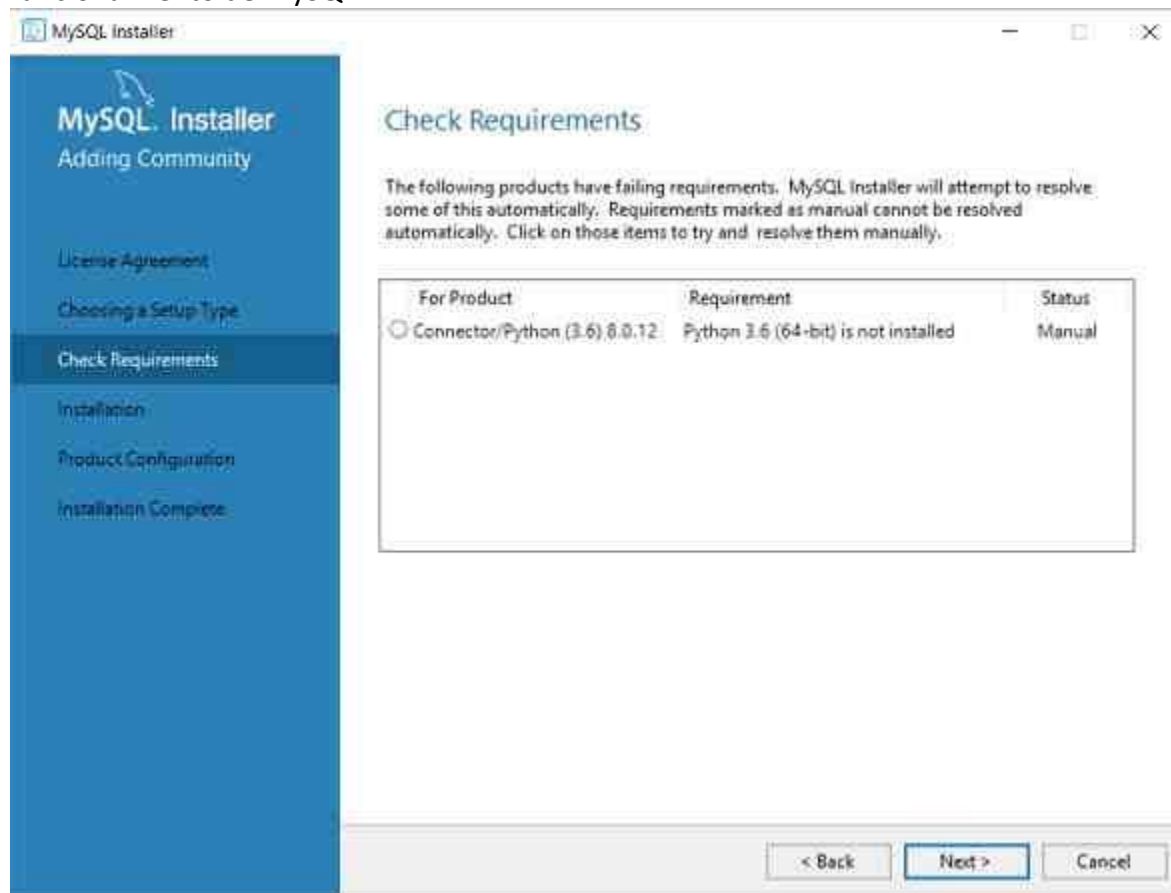
MySQL_ Introducción a las bases de datos

En la segunda pantalla dejamos seleccionado "Developer Default" que nos instalará la mayoría de las herramientas que necesitaremos como programador:



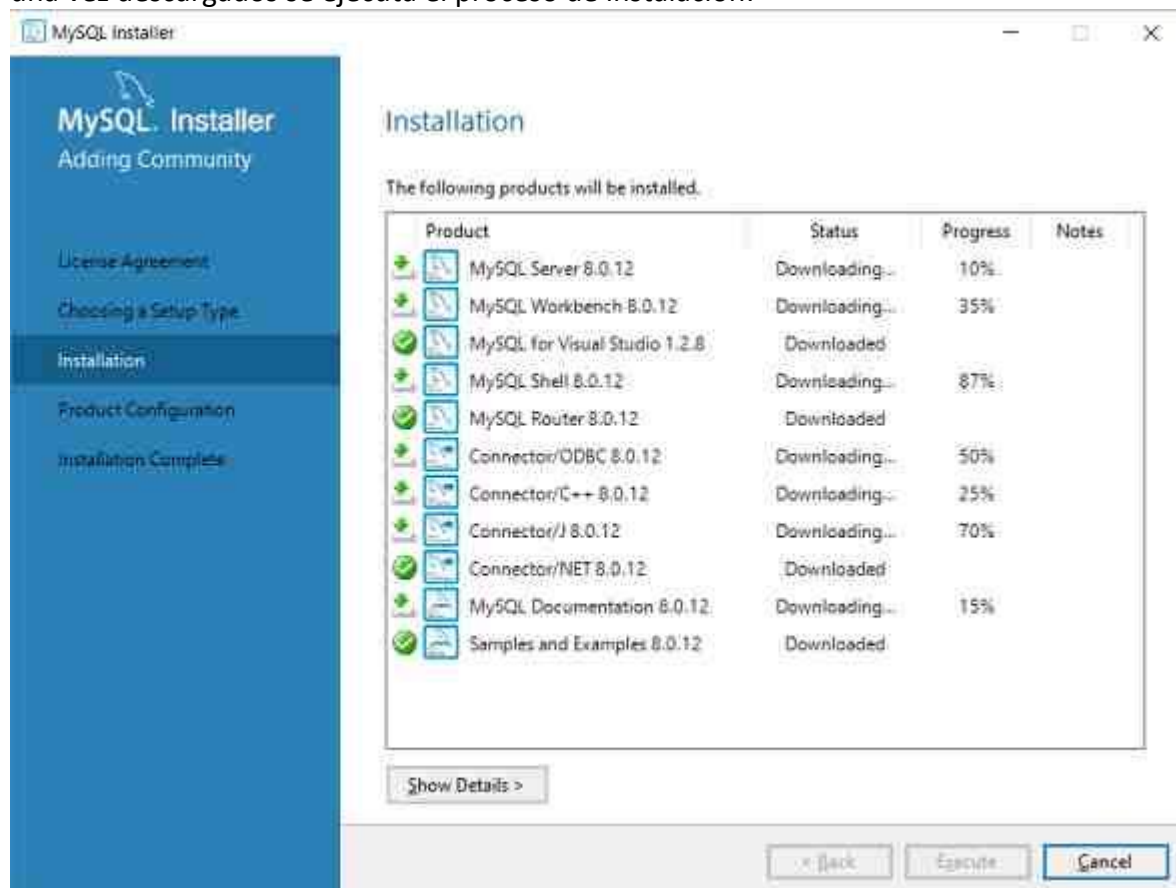
MySQL_ Introducción a las bases de datos

La pantalla siguiente nos informa algunos otros programas que se instalarán para el correcto funcionamiento de MySQL:



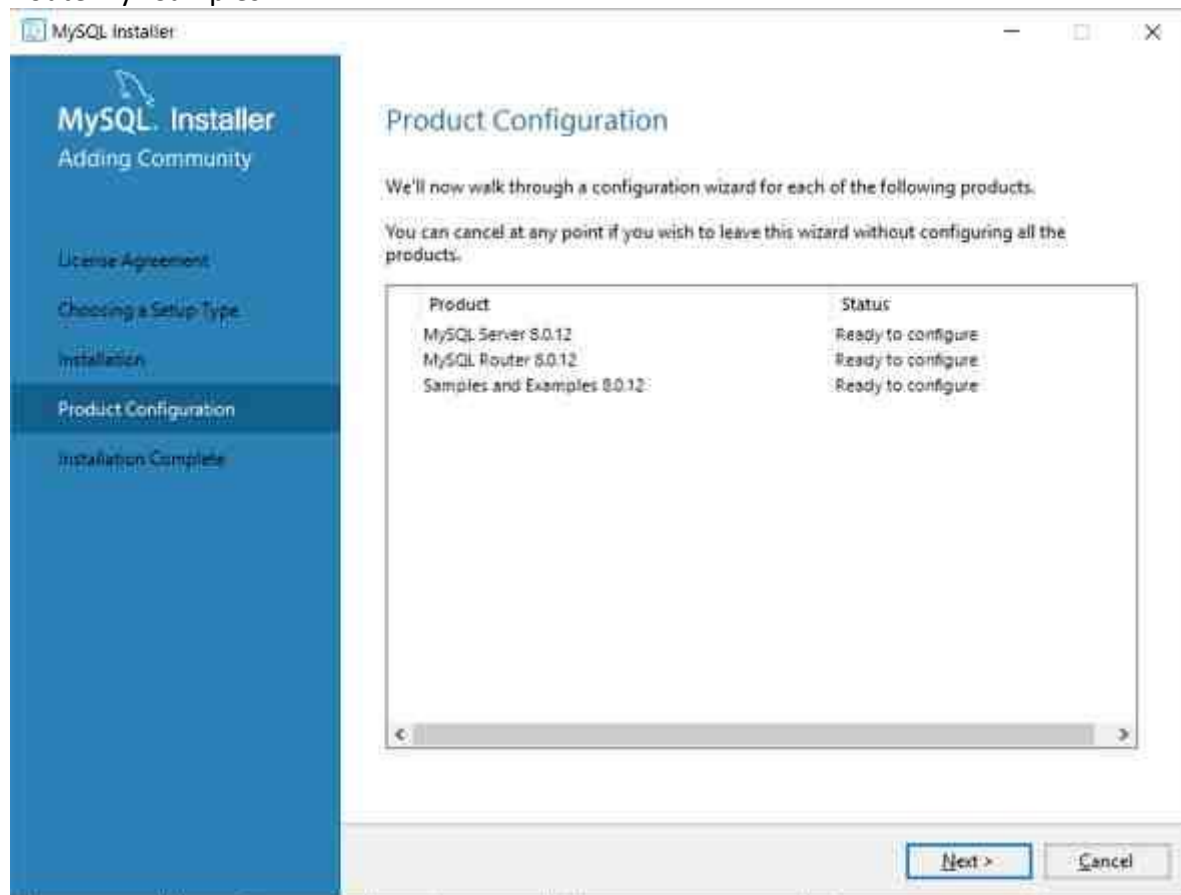
MySQL_ Introducción a las bases de datos

El siguiente paso es el que más tiempo demorará debido a que se descargarán de internet el servidor de MySQL propiamente dicho, manuales, drivers de conexión para distintos lenguajes etc, una vez descargados se ejecuta el proceso de instalación:



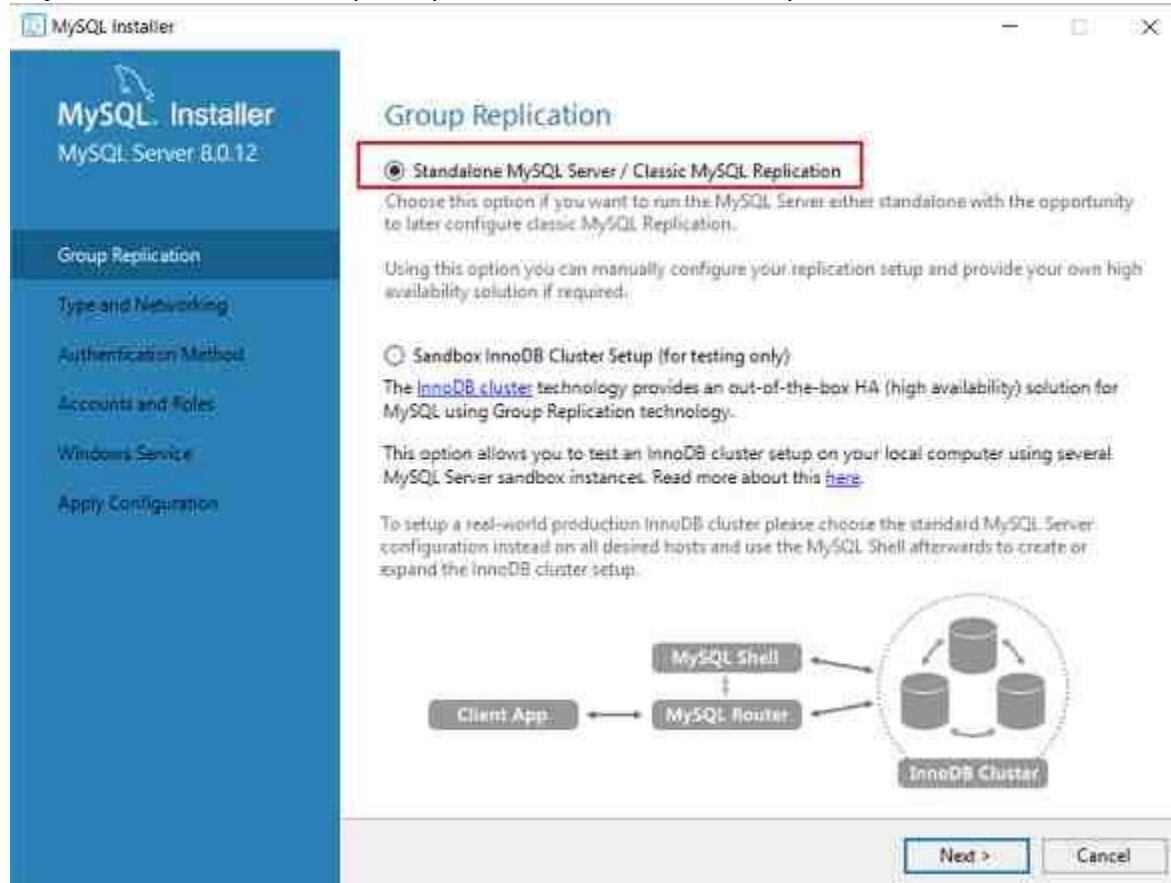
MySQL_ Introducción a las bases de datos

La siguiente ventana nos informa que se procederá a configurar el "MySQL Server", "MySQL Router" y "Samples":



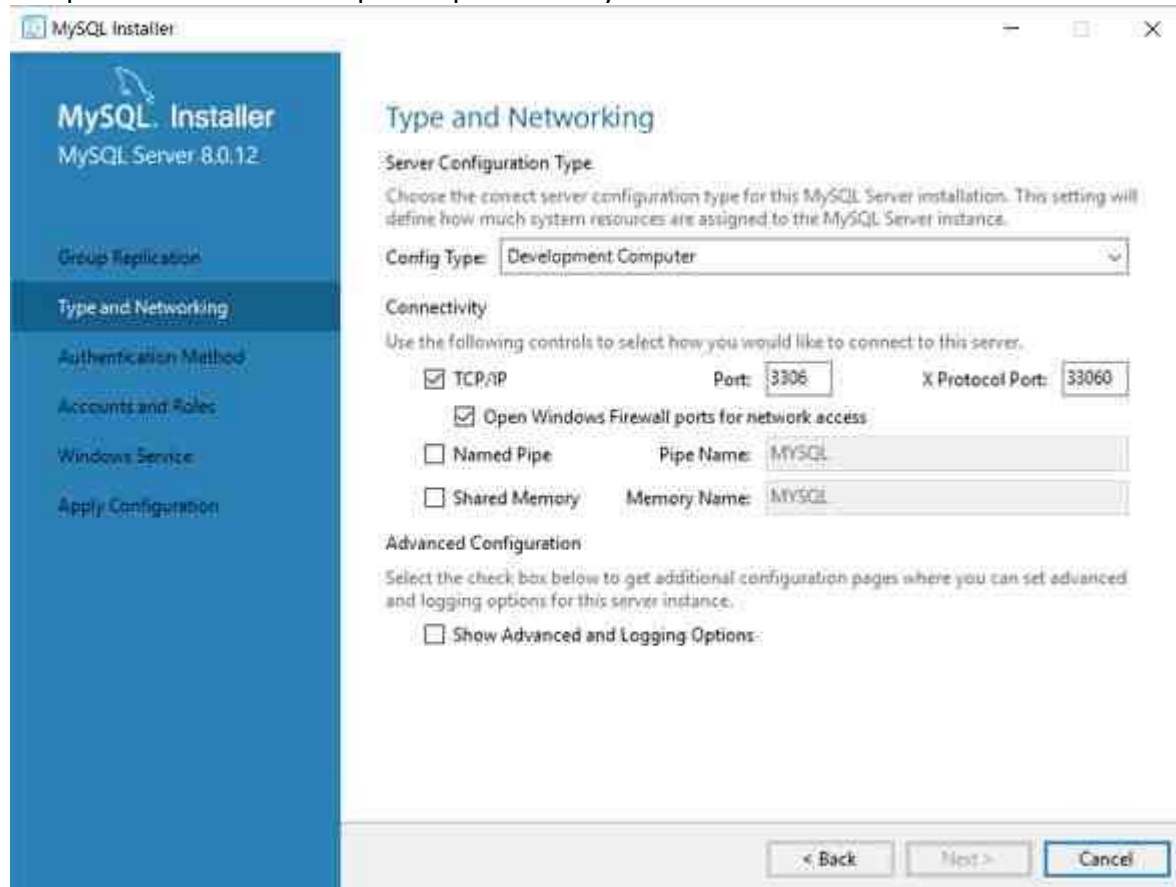
MySQL_ Introducción a las bases de datos

Dejemos seleccionada la opción por defecto "Standalone MySQL Server":



MySQL_ Introducción a las bases de datos

Para configurar el servidor debemos indicar en "Connectivity" el puerto de comunicaciones que por defecto está configurado con el valor "3306" (si ya tiene instalada otra versión de MySQL en su computadora cambie este puerto por "3307" y no tendrá conflictos con la versión anterior):



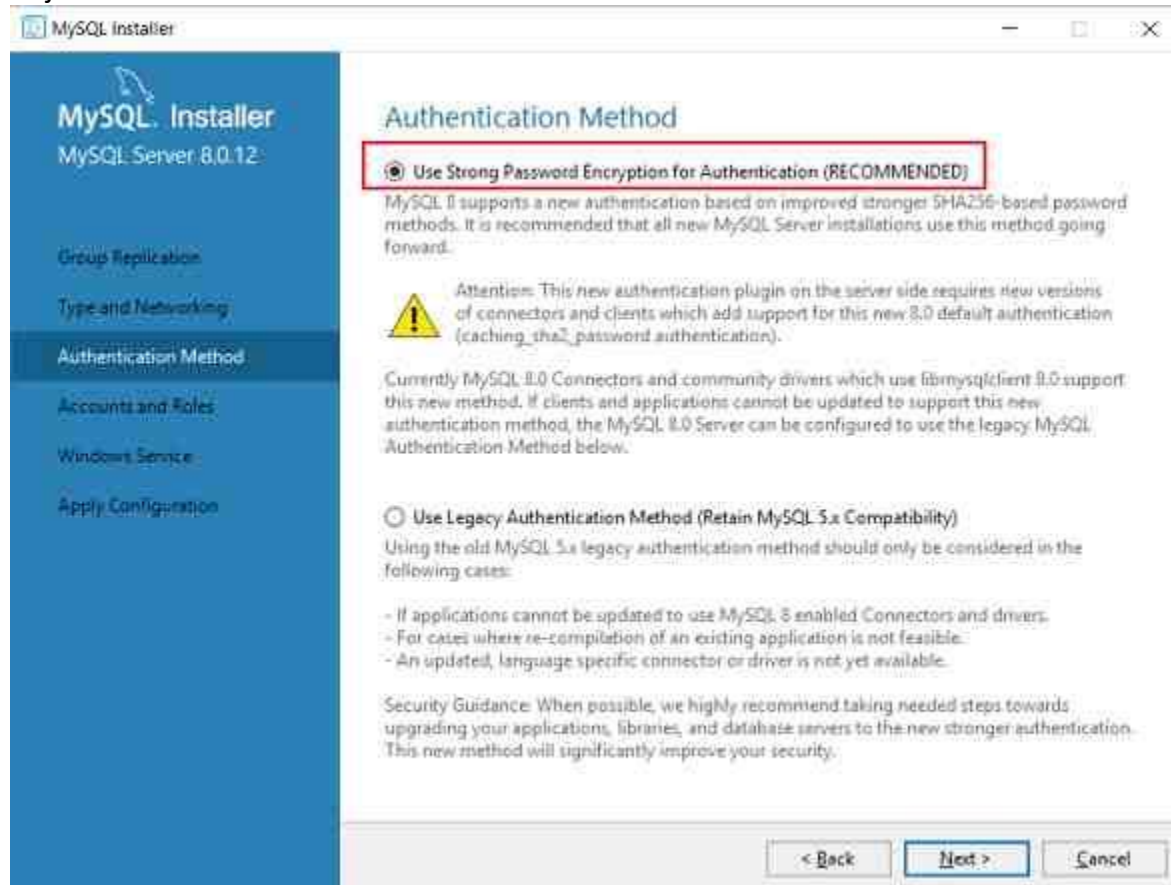
The screenshot shows the 'MySQL Installer' window for 'MySQL Server 8.0.12'. The left sidebar contains the following options: 'Group Replication', 'Type and Networking' (which is selected and highlighted in blue), 'Authentication Method', 'Accounts and Roles', 'Windows Service', and 'Apply Configuration'. The main area is titled 'Type and Networking' and contains the following sections:

- Server Configuration Type:** A text box with the instruction 'Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.' Below it, a dropdown menu labeled 'Config Type:' is set to 'Development Computer'.
- Connectivity:** A text box with the instruction 'Use the following controls to select how you would like to connect to this server.' Below it, there are three checked options:
 - ☒ TCP/IP: Port: 3306, X Protocol Port: 33060
 - ☒ Open Windows Firewall ports for network access
 - ☐ Named Pipe: Pipe Name: MYSQL
 - ☐ Shared Memory: Memory Name: MYSQL
- Advanced Configuration:** A text box with the instruction 'Select the check box below to get additional configuration pages where you can set advanced and logging options for this server instance.' Below it, there is an unchecked checkbox labeled 'Show Advanced and Logging Options:'.

At the bottom right of the window, there are three buttons: '< Back', 'Next >', and 'Cancel'.

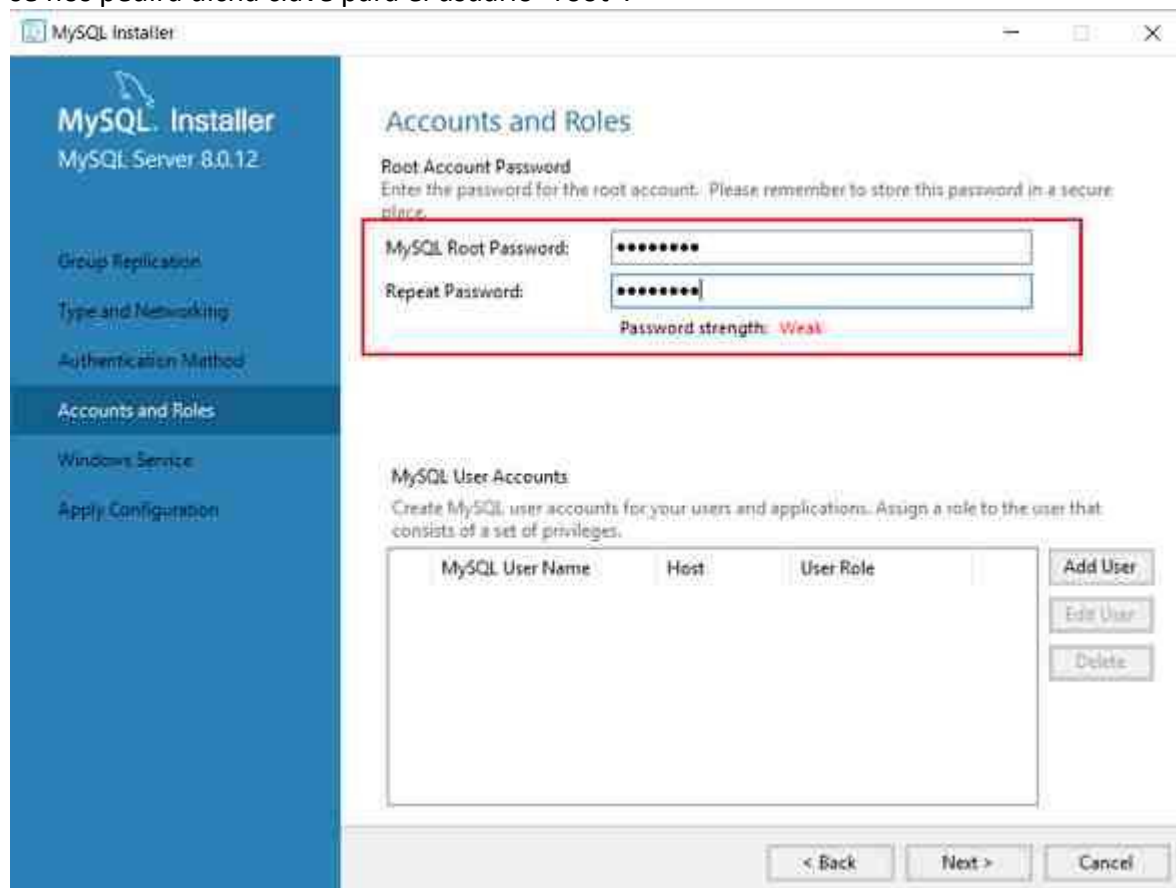
MySQL_ Introducción a las bases de datos

Dejemos el método de autenticación recomendado:



MySQL_ Introducción a las bases de datos

En el siguiente paso debemos definir la clave para el usuario "root" o raíz del servidor de base de datos. No debemos olvidar dicha clave ya que cada vez que necesitemos acceder al servidor MySQL se nos pedirá dicha clave para el usuario "root":

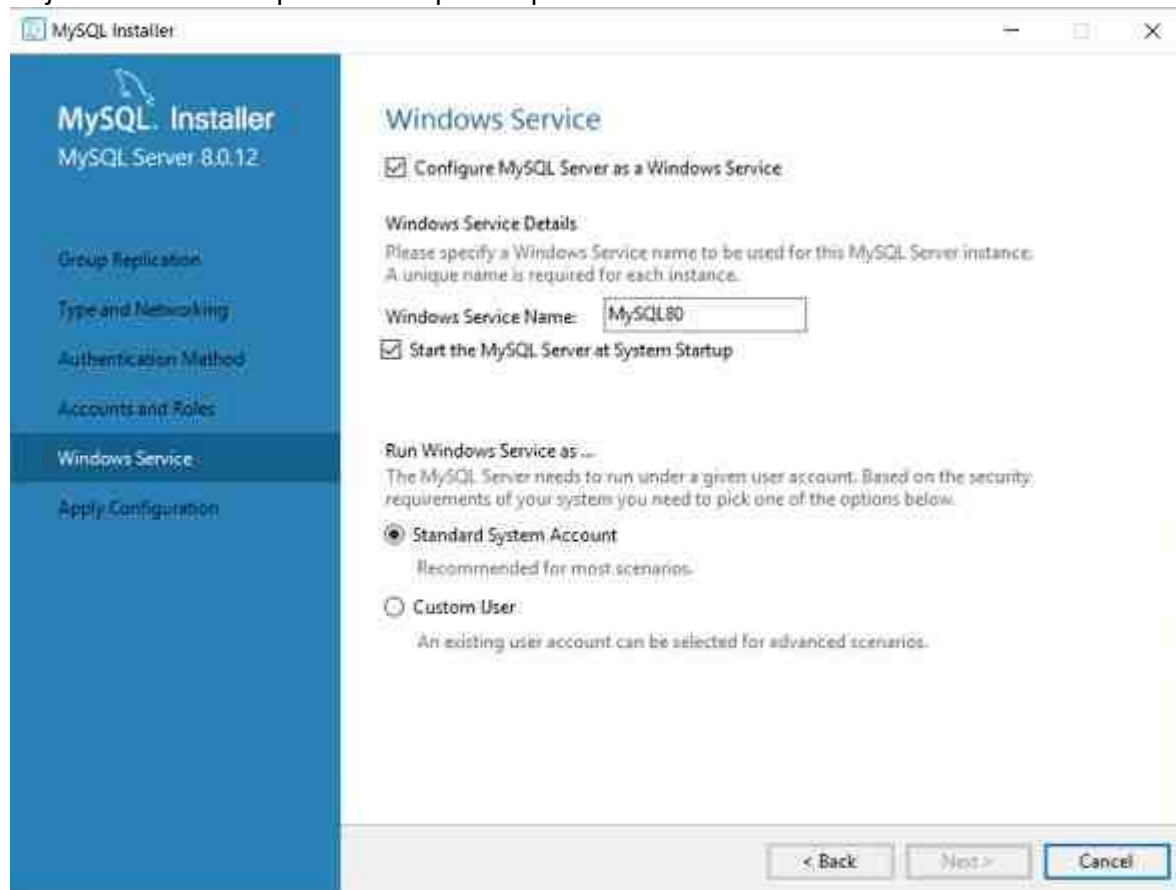


The screenshot shows the 'MySQL Installer' window for 'MySQL Server 8.0.12'. The left sidebar lists the installation steps: Group Replication, Type and Networking, Authentication Method, Accounts and Roles (selected), Windows Service, and Apply Configuration. The main area is titled 'Accounts and Roles' and contains two sections. The first section, 'Root Account Password', prompts the user to enter a password for the root account, with a reminder to store it securely. It features two input fields: 'MySQL Root Password' and 'Repeat Password', both masked with dots. A red box highlights these fields and the 'Password strength' indicator below them, which shows 'Weak' in red text. The second section, 'MySQL User Accounts', instructs the user to create accounts and assign roles. It includes a table with columns for 'MySQL User Name', 'Host', and 'User Role', and buttons for 'Add User', 'Edit User', and 'Delete'. At the bottom of the window are navigation buttons: '< Back', 'Next >', and 'Cancel'.

No crearemos en este momento otras cuentas de usuarios para el servidor MySQL (lo veremos más adelante en este curso)

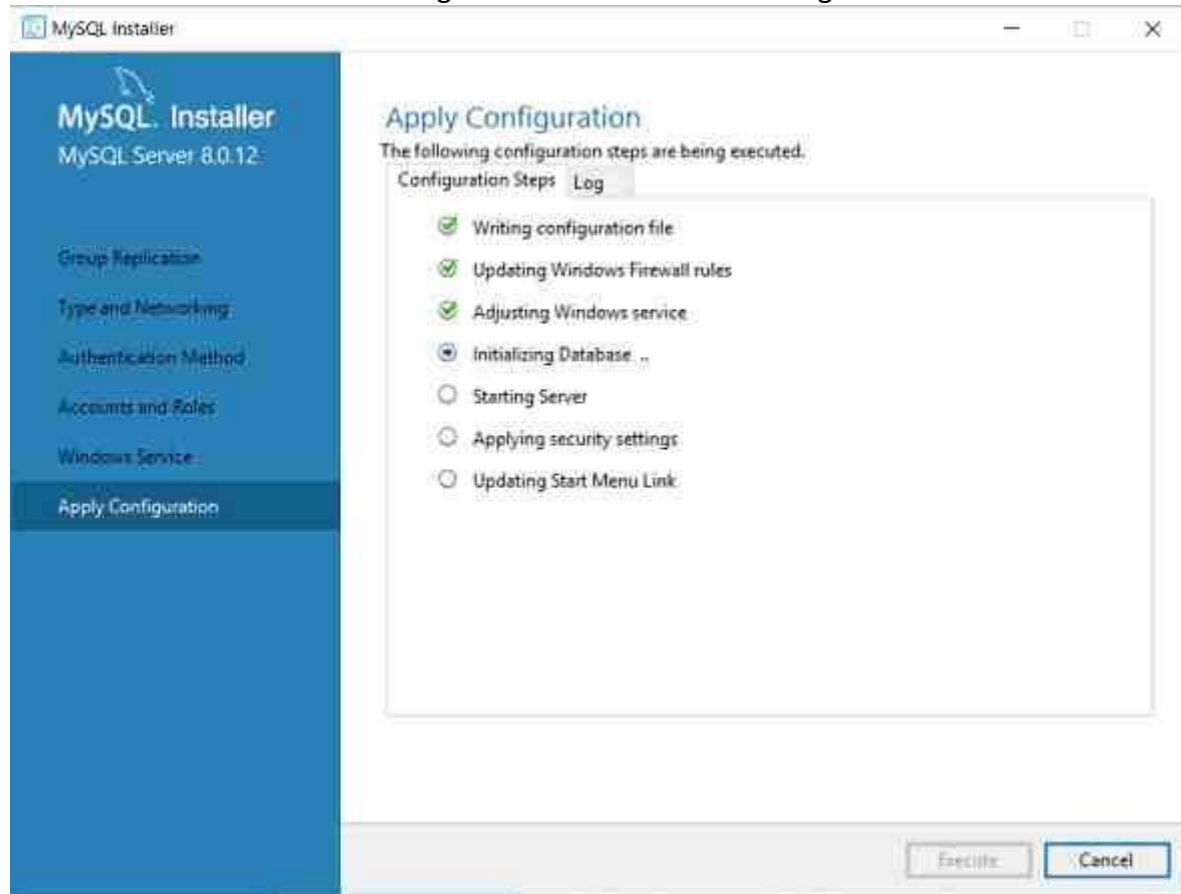
MySQL_ Introducción a las bases de datos

Dejamos los valores por defecto para la pantalla de "Window Service":



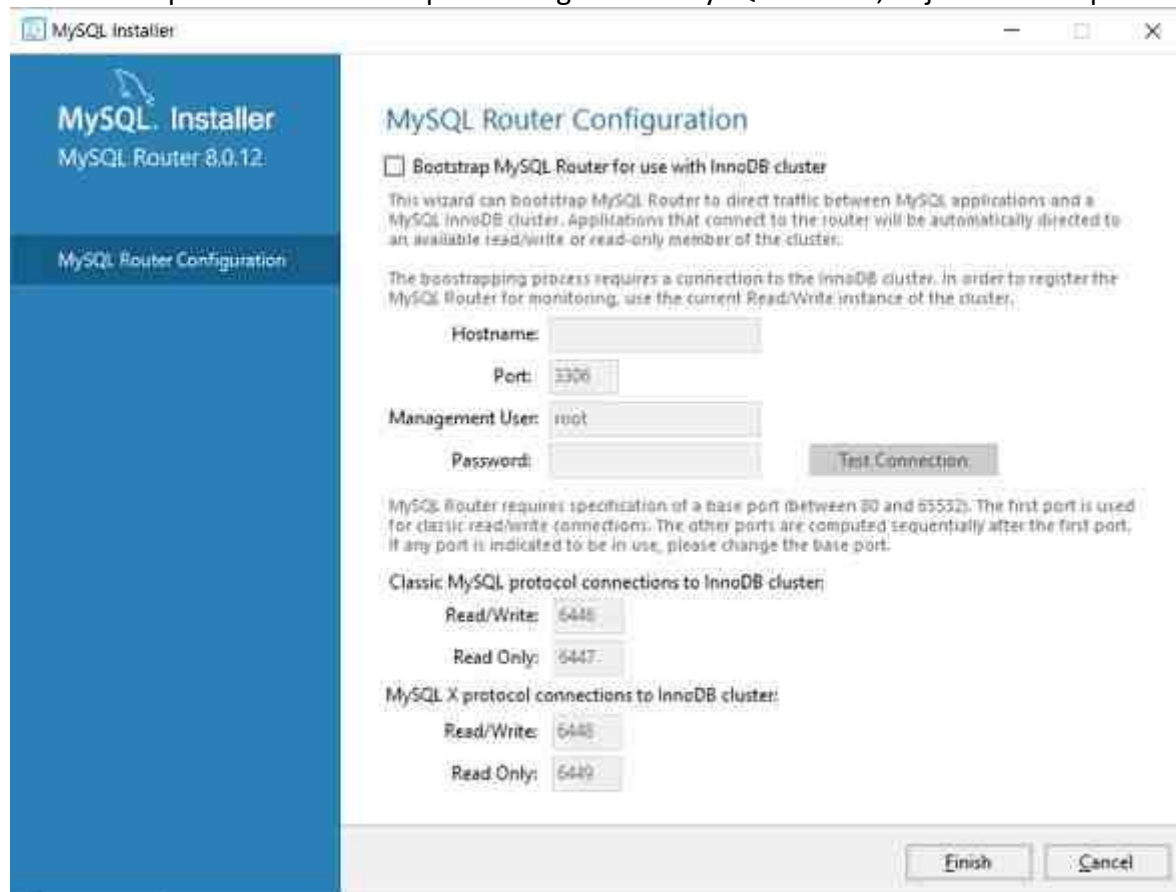
MySQL_ Introducción a las bases de datos

Finalmente confirmamos la configuración definida en los diálogos anteriores:



MySQL_ Introducción a las bases de datos

Los mismos pasos ahora damos para configurar el "MySQL Router", dejamos datos por defecto:



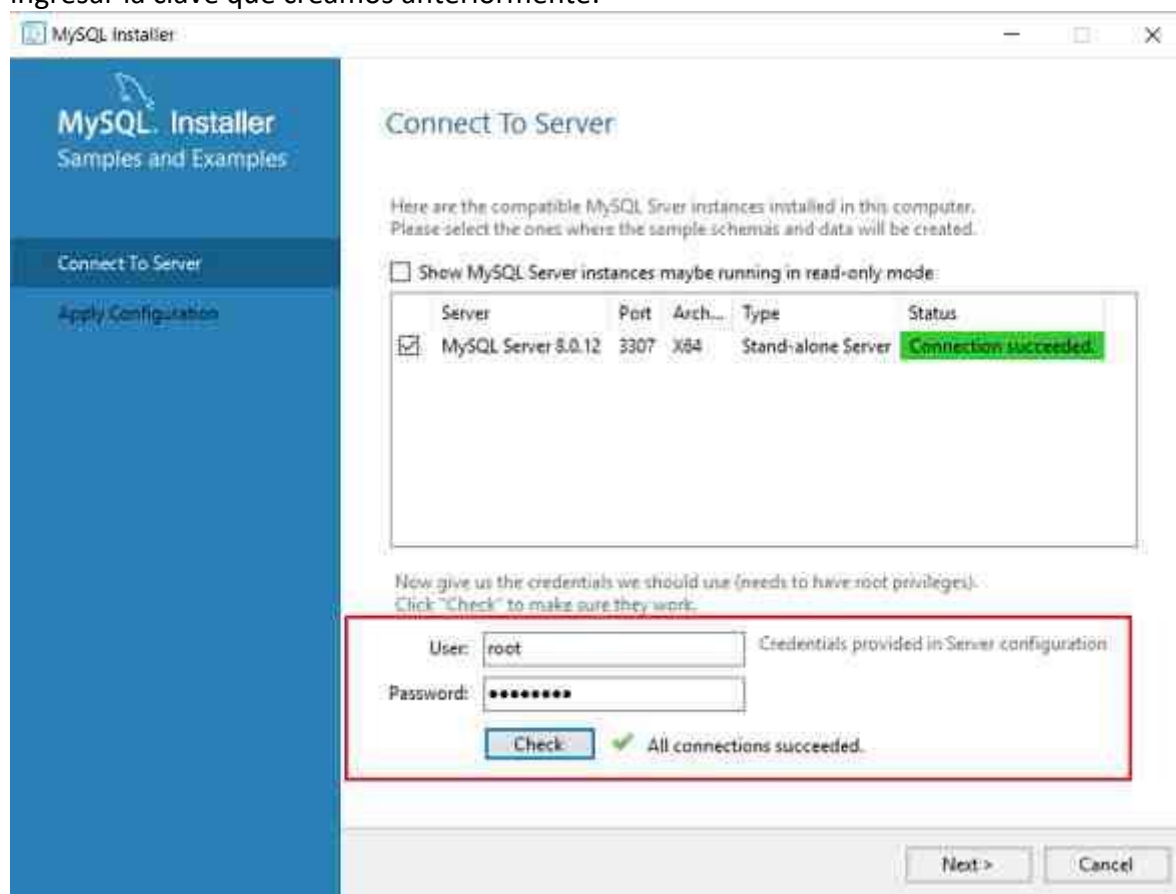
The screenshot shows the 'MySQL Router Configuration' window within the 'MySQL Installer' application. The window title is 'MySQL Router Configuration'. On the left, there is a blue sidebar with the text 'MySQL Installer' and 'MySQL Router 8.0.12.' at the top, and 'MySQL Router Configuration' below it. The main area contains the following configuration options:

- ☐ Bootstrap MySQL Router for use with InnoDB cluster
- This wizard can bootstrap MySQL Router to direct traffic between MySQL applications and a MySQL InnoDB cluster. Applications that connect to the router will be automatically directed to an available read/write or read-only member of the cluster.
- The bootstrapping process requires a connection to the InnoDB cluster. In order to register the MySQL Router for monitoring, use the current Read/Write instance of the cluster.
- Hostname:
- Port:
- Management User:
- Password:
-
- MySQL Router requires specification of a base port (between 30 and 65532). The first port is used for classic read/write connections. The other ports are computed sequentially after the first port. If any port is indicated to be in use, please change the base port.
- Classic MySQL protocol connections to InnoDB cluster:
 - Read/Write:
 - Read Only:
- MySQL X protocol connections to InnoDB cluster:
 - Read/Write:
 - Read Only:

At the bottom right, there are 'Finish' and 'Cancel' buttons.

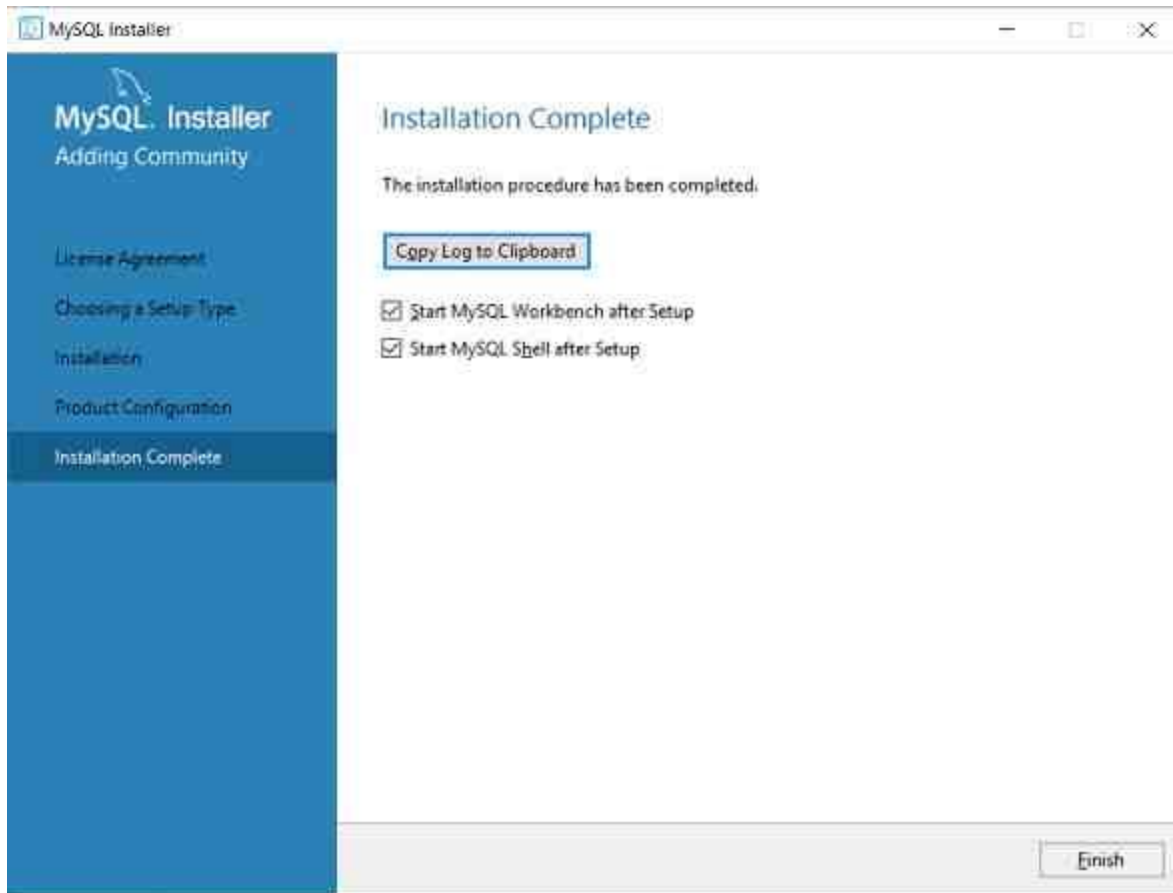
MySQL_ Introducción a las bases de datos

La siguiente pantalla nos permite verificar el correcto funcionamiento del servidor, debemos ingresar la clave que creamos anteriormente:



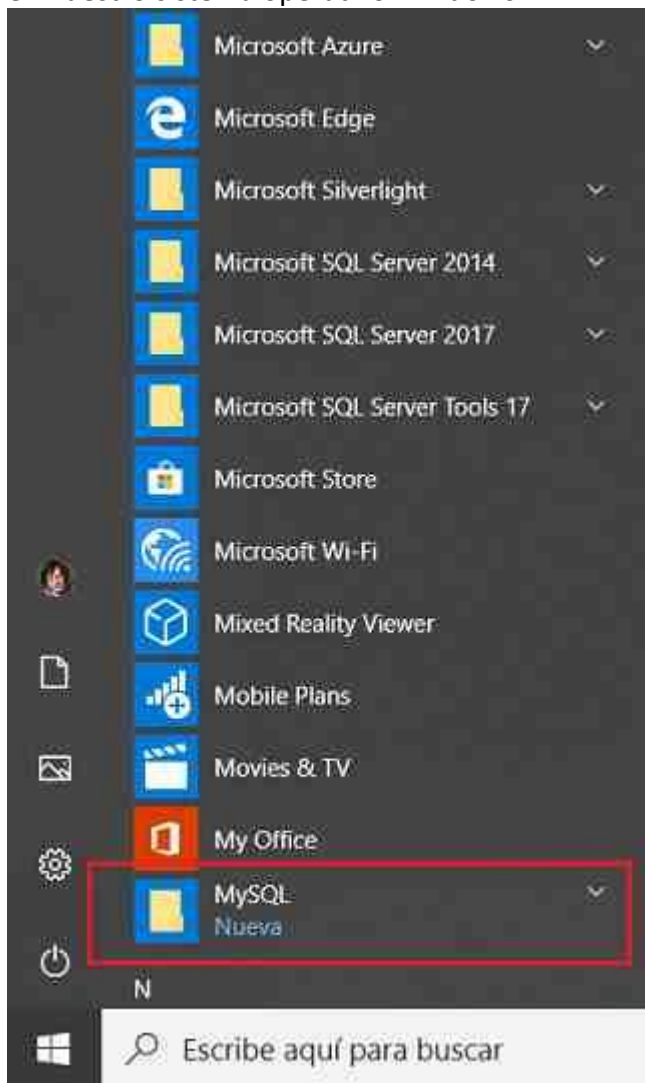
MySQL_ Introducción a las bases de datos

La última pantalla inicia las aplicaciones "Workbench" y "MySQL Shell" que luego veremos en este curso:



MySQL_ Introducción a las bases de datos

Tenemos ya todo el software necesario para desarrollar en forma local el curso de MySQL instalado en nuestro sistema operativo Windows:



MySQL_ Introducción a las bases de datos

Referencia rapida de comandos basicos MySQL

Introducción

El lenguaje de consulta estructurado o SQL (por sus siglas en inglés **Structured Query Language**) es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas. Una de sus características es el manejo del álgebra y el cálculo relacional que permiten efectuar consultas con el fin de recuperar de forma sencilla información de interés de bases de datos, así como hacer cambios en ella.

El SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales y permite así gran variedad de operaciones.

Componentes del SQL

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

Existen tres tipos de comandos SQL:

Los **DLL(Data Definition Language)** que permiten crear y definir nuevas bases de datos, campos e índices. Los **DML(Data Manipulation Language)** que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos. Los **DCL(Data Control Language)** que se encargan de definir los permisos sobre los datos

MySQL_ Introducción a las bases de datos

Lenguaje de definición de datos (DDL)

Comando Descripción

CREATE	Utilizado para crear nuevas tablas, campos e índices
DROP	Empleado para eliminar tablas e índices
ALTER	Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.

El lenguaje de definición de datos (en inglés Data Definition Language, o DDL), es el que se encarga de la modificación de la estructura de los objetos de la base de datos. Incluye órdenes para modificar, borrar o definir las tablas en las que se almacenan los datos de la base de datos. Existen cuatro operaciones básicas: CREATE, ALTER, DROP y TRUNCATE.

CREATE

Este comando crea un objeto dentro del gestor de base de datos. Puede ser una base de datos, tabla, índice, procedimiento almacenado o vista.

Ejemplo (crear una tabla):

```
# CREATE TABLE Empleado
(
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    Nombre VARCHAR(50),
    Apellido VARCHAR(50),
    Direccion VARCHAR(255),
    Ciudad VARCHAR(60),
    Telefono VARCHAR(15),
    Peso VARCHAR (5),
    Edad (2),
    Actividad Específica (100),
    idCargo INT
)
```

ALTER

Este comando permite modificar la estructura de un objeto. Se pueden agregar/quitar campos a una tabla, modificar el tipo de un campo, agregar/quitar índices a una tabla, modificar un trigger, etc.

Ejemplo (agregar columna a una tabla):

```
# ALTER TABLE 'NOMBRE_TABLA' ADD NUEVO_CAMPO INT;
# ALTER TABLE 'NOMBRE_TABLA' DROP COLUMN NOMBRE_COLUMNNA;
```


MySQL_ Introducción a las bases de datos

DROP

Este comando elimina un objeto de la base de datos. Puede ser una tabla, vista, índice, trigger, función, procedimiento o cualquier otro objeto que el motor de la base de datos soporte. Se puede combinar con la sentencia ALTER.

Ejemplo:

```
# DROP TABLE 'NOMBRE_TABLA';  
# DROP SCHEMA 'ESQUEMA';  
# DROP DATABASE 'BASEDATOS';
```

TRUNCATE

Este comando trunca todo el contenido de una tabla. La ventaja sobre el comando DROP, es que si se quiere borrar todo el contenido de la tabla, es mucho más rápido, especialmente si la tabla es muy grande. La desventaja es que TRUNCATE sólo sirve cuando se quiere eliminar absolutamente todos los registros, ya que no se permite la cláusula WHERE. Si bien, en un principio, esta sentencia parecería ser DML (Lenguaje de Manipulación de Datos), es en realidad una DDL, ya que internamente, el comando TRUNCATE borra la tabla y la vuelve a crear y no ejecuta ninguna transacción.

Ejemplo:

```
# TRUNCATE TABLE 'NOMBRE_TABLA';
```

MySQL_ Introducción a las bases de datos

Lenguaje de manipulación de datos DML(Data Manipulation Language)

Comando Descripción

SELECT	Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado
INSERT	Utilizado para cargar lotes de datos en la base de datos en una única operación.
UPDATE	Utilizado para modificar los valores de los campos y registros especificados Utilizado para modificar las tablas agregando campos o cambiando la definición de los campos.
DELETE	Utilizado para eliminar registros de una tabla

Definición

Un lenguaje de manipulación de datos (Data Manipulation Language, o DML en inglés) es un lenguaje proporcionado por el sistema de gestión de base de datos que permite a los usuarios llevar a cabo las tareas de consulta o manipulación de los datos, organizados por el modelo de datos adecuado. El lenguaje de manipulación de datos más popular hoy día es SQL, usado para recuperar y manipular datos en una base de datos relacional.

INSERT

Una sentencia INSERT de SQL agrega uno o más registros a una (y sólo una) tabla en una base de datos relacional.

Forma básica:

```
# INSERT INTO "tabla" ("columna1", ["columna2,..."]) VALUES ("valor1", ["valor2,..."])
```

Las cantidades de columnas y valores deben ser iguales. Si una columna no se especifica, le será asignado el valor por omisión. Los valores especificados (o implícitos) por la sentencia INSERT deberán satisfacer todas las restricciones aplicables. Si ocurre un error de sintaxis o si alguna de las restricciones es violada, no se agrega la fila y se devuelve un error.

Ejemplo:

```
# INSERT INTO agenda_telefonica (nombre, numero) VALUES ('Roberto Jeldrez', 4886850);
```

Cuando se especifican todos los valores de una tabla, se puede utilizar la sentencia acortada:

```
# INSERT INTO "VALUES ("valor1", ["valor2,..."])
```

Ejemplo (asumiendo que 'nombre' y 'número' son las únicas columnas de la tabla 'agenda_telefonica'):

```
# INSERT INTO agenda_telefonica VALUES ('Jhonny Aguiar', 080473968);
```

UPDATE

Una sentencia UPDATE de SQL es utilizada para modificar los valores de un conjunto de registros existentes en una tabla.

Ejemplo:

MySQL_ Introducción a las bases de datos

```
# UPDATE mi_tabla SET campo1 = 'nuevo valor campo1' WHERE campo2 = 'N';
```

DELETE

Una sentencia DELETE de SQL borra uno o más registros existentes en una tabla.

Forma básica:

```
# DELETE FROM 'tabla' WHERE 'columna1' = 'valor1'
```

Ejemplo:

```
# DELETE FROM My_table WHERE field2 = 'N';
```

MySQL_ Introducción a las bases de datos

MySQL_ Introducción a las bases de datos

Clausulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular.

Comando Descripción

FROM Utilizada para especificar la tabla de la cual se van a seleccionar los registros
GROUP BY Utilizada para separar los registros seleccionados en grupos específicos
HAVING Utilizada para expresar condición que debe satisfacer cada grupo
ORDER BY Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico
WHERE Utilizada para determinar los registros seleccionados en la clausula FROM

Operadores Lógicos

Operador Uso

AND Es el “y” lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas.
OR Es el “o” lógico. Evalúa dos condiciones y devuelve un valor de verdad si alguna de las dos es cierta.
NOT Negación lógica. Devuelve el valor contrario de la expresión.

MySQL_ Introducción a las bases de datos

Operadores de comparación

Operador Uso

< Menor que

> Mayor que

<> Distinto de

<= Menor o igual que

>= Mayor o igual que

BETWEEN Intervalo

LIKE Comparación

In Especificar

MySQL_ Introducción a las bases de datos

Funciones de agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

Comando Descripción

AVG	Utilizada para calcular el promedio de los valores de un campo determinado
COUNT	Utilizada para devolver el número de registros de la selección
SUM	Utilizada para devolver la suma de todos los valores de un campo determinado
MAX	Utilizada para devolver el valor más alto de un campo especificado
MIN	Utilizada para devolver el valor más bajo de un campo especificado

Consultas

Consultas de selección

Las consultas de selección se utilizan para indicar al motor de datos que devuelva información de las bases de datos, esta información es devuelta en forma de conjunto de registros. Este conjunto de registros es modificable.

Básicas

La sintaxis básica de una consulta de selección es:

```
# SELECT Campos FROM Tabla;  
# SELECT Nombre, Telefono FROM Clientes;
```

Ordenar los registros

Se puede especificar el orden en que se desean recuperar los registros de las tablas mediante la cláusula **ORDER BY**:

```
# SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY Nombre;
```

Se pueden ordenar los registros por mas de un campo:

```
# SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal, Nombre;
```

Y se puede especificar el orden de los registros: ascendente mediante la cláusula (**ASC** -se toma este valor por defecto) ó descendente (**DESC**):

```
# SELECT CodigoPostal, Nombre, Telefono FROM Clientes ORDER BY CodigoPostal DESC , Nombre  
ASC;
```

MySQL_ Introducción a las bases de datos

Consultas con predicado

ALL Si no se incluye ninguno de los predicados se asume ALL. El Motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL:

```
# SELECT ALL FROM Empleados;  
# SELECT * FROM Empleados;
```

TOP Devuelve un cierto número de registros que entran entre al principio o al final de un rango especificado por una cláusula ORDER BY. Supongamos que queremos recuperar los nombres de los 25 primeros estudiantes del curso 1994:

```
# SELECT TOP 25 Nombre, Apellido  
FROM Estudiantes  
ORDER BY Nota DESC;
```

Si no se incluye la cláusula ORDER BY, la consulta devolverá un conjunto arbitrario de 25 registros de la tabla Estudiantes. El predicado TOP no elige entre valores iguales. En el ejemplo anterior, si la nota media número 25 y la 26 son iguales, la consulta devolverá 26 registros. Se puede utilizar la palabra reservada PERCENT para devolver un cierto porcentaje de registros que caen al principio o al final de un rango especificado por la cláusula ORDER BY. Supongamos que en lugar de los 25 primeros estudiantes deseamos el 10 por ciento del curso::

```
# SELECT TOP 10 PERCENT Nombre, Apellido  
FROM Estudiantes  
ORDER BY Nota DESC;
```

DISTINCT Omite los registros que contienen datos duplicados en los campos seleccionados. Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos:

```
# SELECT DISTINCT Apellido FROM Empleados;
```

DISTINCTROW Devuelve los registros diferentes de una tabla; a diferencia del predicado anterior que sólo se fijaba en el contenido de los campos seleccionados, éste lo hace en el contenido del registro completo independientemente de los campos indicados en la cláusula SELECT:

```
1. # SELECT DISTINCTROW Apellido FROM Empleados;
```


MySQL_ Introducción a las bases de datos

-Criterios de selección

Operadores Lógicos

Los operadores lógicos soportados por SQL son:

AND, OR, XOR, Eqv, Imp, Is y Not.

A excepción de los dos últimos todos poseen la siguiente sintaxis:

```
<expresión1> operador <expresión2>
```

En donde expresión1 y expresión2 son las condiciones a evaluar, el resultado de la operación varía en función del operador lógico:

```
# SELECT * FROM Empleados WHERE Edad > 25 AND Edad < 50;
# SELECT * FROM Empleados WHERE (Edad > 25 AND Edad < 50) OR Sueldo = 100;
# SELECT * FROM Empleados WHERE NOT Estado = 'Soltero';
# SELECT * FROM Empleados WHERE (Sueldo > 100 AND Sueldo < 500) OR (Provincia = 'Madrid'
AND Estado = 'Casado');
```

Operador BETWEEN

Para indicar que deseamos recuperar los registros según el intervalo de valores de un campo emplearemos el operador **Between**:

```
# SELECT * FROM Pedidos WHERE CodPostal Between 28000 And 28999;
(Devuelve los pedidos realizados en la provincia de Madrid)

# SELECT IIf(CodPostal Between 28000 And 28999, 'Provincial', 'Nacional') FROM Editores;
(Devuelve el valor 'Provincial' si el código postal se encuentra en el intervalo,'Nacional' en caso contrario)
```

Operador LIKE

Se utiliza para comparar una expresión de cadena con un modelo en una expresión SQL. Su sintaxis es:

```
expresión LIKE modelo
```

Operador IN

Este operador devuelve aquellos registros cuyo campo indicado coincide con alguno de los indicados en una lista. Su sintaxis es:

```
expresión [Not] In(valor1, valor2, . . .)
```

```
# SELECT * FROM Pedidos WHERE Provincia In ('Madrid', 'Barcelona', 'Sevilla');
```

MySQL_ Introducción a las bases de datos

Clausula WHERE

La cláusula WHERE puede usarse para determinar qué registros de las tablas enumeradas en la cláusula FROM aparecerán en los resultados de la instrucción SELECT. WHERE es opcional, pero cuando aparece debe ir a continuación de FROM:

```
# SELECT Apellidos, Salario FROM Empleados
WHERE Salario > 21000;
# SELECT Id_Producto, Existencias FROM Productos
WHERE Existencias <= Nuevo_Pedido;
```

-Agrupamiento de registros (Agregación)

AVG

Calcula la media aritmética de un conjunto de valores contenidos en un campo especificado de una consulta:

Avg(expr)

La función Avg no incluye ningún campo Null en el cálculo. Un ejemplo del funcionamiento de **AVG**:

```
# SELECT Avg(Gastos) AS Promedio FROM
Pedidos WHERE Gastos > 100;
```

MAX, MIN

Devuelven el mínimo o el máximo de un conjunto de valores contenidos en un campo específico de una consulta. Su sintaxis es:

Min(expr)

Max(expr)

Un ejemplo de su uso:

```
# SELECT Min(Gastos) AS ElMin FROM Pedidos
WHERE Pais = 'Costa Rica';
# SELECT Max(Gastos) AS ElMax FROM Pedidos
WHERE Pais = 'Costa Rica';
```

SUM

Devuelve la suma del conjunto de valores contenido en un campo específico de una consulta. Su sintaxis es:

Sum(expr)

Por ejemplo:

```
# SELECT Sum(PrecioUnidad * Cantidad)
AS Total FROM DetallePedido;
```

MySQL_ Introducción a las bases de datos

GROUP BY

Combina los registros con valores idénticos, en la lista de campos especificados, en un único registro:

```
# SELECT campos FROM tabla WHERE criterio  
GROUP BY campos del grupo
```

Todos los campos de la lista de campos de SELECT deben o bien incluirse en la cláusula GROUP BY o como argumentos de una función SQL agregada:

```
# SELECT Id_Familia, Sum(Stock)  
FROM Productos GROUP BY Id_Familia;
```

HAVING es similar a WHERE, determina qué registros se seleccionan. Una vez que los registros se han agrupado utilizando GROUP BY, HAVING determina cuales de ellos se van a mostrar.

```
# SELECT Id_Familia Sum(Stock) FROM Productos GROUP BY Id_Familia HAVING Sum(Stock) > 100  
AND NombreProducto Like BOS*;
```

Manejo de varias tablas

Partiendo de la definición de las siguientes tablas:

Tabla clientes

cid	nombre	telefono
1	jose	111
2	maria	222
3	manuel	333
4	jesus	4444

Tabla Acciones

aid	cid	accion	cantidad
1	2	REDHAT	10
2	4	NOVELL	20
3	4	SUN	30
4	5	FORD	100

MySQL_ Introducción a las bases de datos

-Consultas mediante JOIN

JOIN

La sentencia SQL JOIN se utiliza para relacionar varias tablas. Nos permitirá obtener un listado de los campos que tienen coincidencias en ambas tablas:

```
# select nombre, telefono, accion, cantidad from clientes join acciones on clientes.cid=acciones.cid;
```

resultando:

```
+-----+-----+-----+-----+
| nombre | telefono | accion | cantidad |
+-----+-----+-----+-----+
| maria  | 222     | REDHAT | 10       |
| jesus  | 4444    | NOVELL | 20       |
| jesus  | 4444    | SUN    | 30       |
+-----+-----+-----+-----+
```

LEFT JOIN

La sentencia LEFT JOIN nos dará el resultado anterior mas los campos de la tabla de la izquierda del **JOIN** que no tienen coincidencias en la tabla de la derecha:

```
# select nombre, telefono, accion, cantidad from clientes left join acciones on
clientes.cid=acciones.cid;
```

con resultado:

```
+-----+-----+-----+-----+
| nombre | telefono | accion | cantidad |
+-----+-----+-----+-----+
| jose   | 111     | NULL   | NULL     |
| maria  | 222     | REDHAT | 10       |
| manuel | 333     | NULL   | NULL     |
| jesus  | 4444    | NOVELL | 20       |
| jesus  | 4444    | SUN    | 30       |
+-----+-----+-----+-----+
```

RIGHT JOIN

Identico funcionamiento que en el caso anterior pero con la tabla que se incluye en la consulta a la derecha del **JOIN**:

```
# select nombre, telefono, accion, cantidad from clientes right join acciones on
clientes.cid=acciones.cid;
```

cuyo resultado será:

```
+-----+-----+-----+-----+
| nombre | telefono | accion | cantidad |
+-----+-----+-----+-----+
| maria  | 222     | REDHAT | 10       |
| jesus  | 4444    | NOVELL | 20       |
| jesus  | 4444    | SUN    | 30       |
| NULL   | NULL    | FORD   | 100      |
+-----+-----+-----+-----+
```

MySQL_ Introducción a las bases de datos

UNION y UNION ALL

Podemos combinar el resultado de varias sentencias con UNION o UNION ALL. UNION no nos muestra los resultados duplicados, pero UNION ALL si los muestra:

```
# select nombre, telefono, accion, cantidad from clientes left join acciones on
clientes.cid=acciones.cid where accion is null union select nombre, telefono, accion, cantidad from
clientes right join acciones on clientes.cid=acciones.cid where nombre is null;
```

que mostrará:

```
+-----+-----+-----+-----+
| nombre | telefono | accion | cantidad |
+-----+-----+-----+-----+
| jose   | 111     | NULL   | NULL     |
| manuel | 333     | NULL   | NULL     |
| NULL   | NULL    | FORD   | 100      |
+-----+-----+-----+-----+
```

Vistas

Las vistas ("views") en SQL son un mecanismo que permite generar un resultado a partir de una consulta (query) almacenado, y ejecutar nuevas consultas sobre este resultado como si fuera una tabla normal. Las vistas tienen la misma estructura que una tabla: filas y columnas. La única diferencia es que sólo se almacena de ellas la definición, no los datos.

La cláusula CREATE VIEW permite la creación de vistas. La cláusula asigna un nombre a la vista y permite especificar la consulta que la define. Su sintaxis es:

```
# CREATE VIEW id_vista [(columna,...)]AS especificación_consulta;
```

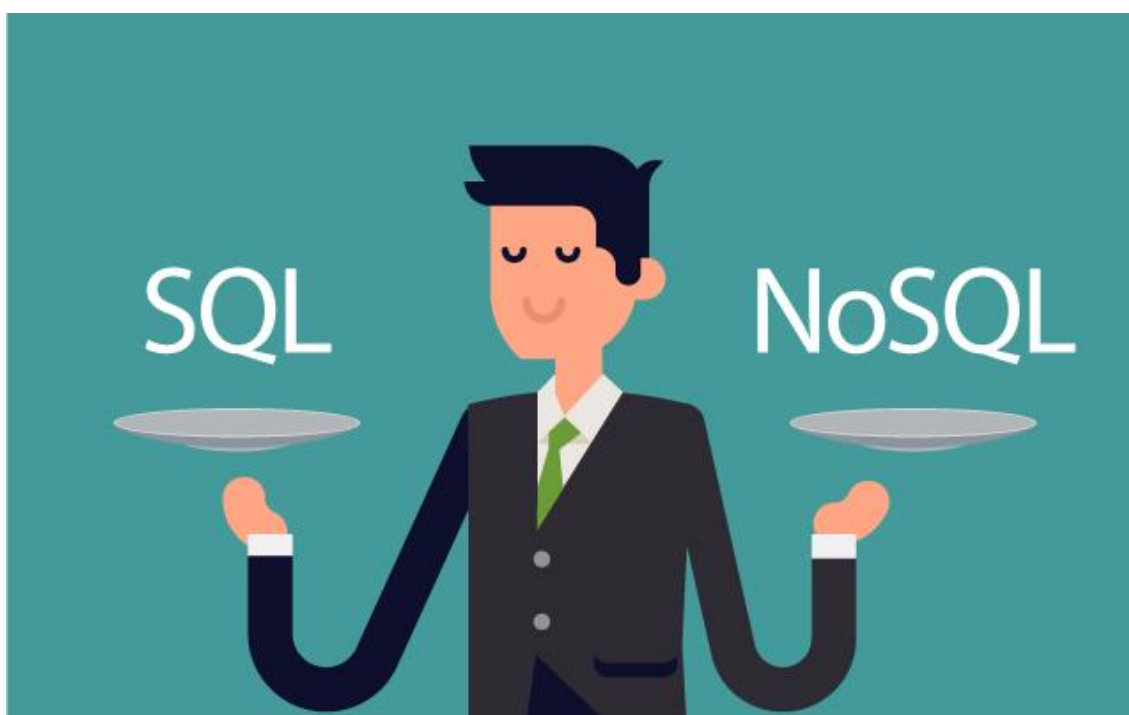
Opcionalmente se puede asignar un nombre a cada columna de la vista. Si se especifica, la lista de nombres de las columnas debe de tener el mismo número de elementos que el número de columnas producidas por la consulta. Si se omiten, cada columna de la vista¹ adopta el nombre de la columna correspondiente en la consulta.

MySQL_ Introducción a las bases de datos

SQL vs. noSQL: ¿qué es mejor?

Es muy común entre desarrolladores de aplicaciones encontrarse en una situación de **tener que elegir si se va a usar una base de datos relacional o no relacional**. La mayoría no se lo piensa demasiado y opta por la opción que más conocen y con la que más cómodos trabajan. Tampoco es una decisión catastrófica; en realidad, ya sea la base de datos relacional o no, se puede construir cualquier cosa.

Vale, entonces, **¿por qué es importante saber en qué se diferencian y cuál deberíamos usar en cada caso?** Pues porque un buen diseño de base de datos con la tecnología apropiada indudablemente **aporta calidad al proyecto**. Dependiendo de la naturaleza de la aplicación, interesa que la base de datos tenga unas características u otras.



MySQL_ Introducción a las bases de datos

Un poco de historia

Las **bases de datos relacionales** o de **lenguaje de consulta SQL** se empezaron a usar en los años 80 y a día de hoy siguen siendo la opción más popular. En cambio, las **bases de datos no relacionales** o de **lenguaje de consulta NoSQL** solo están empezando a ser más populares en los últimos años. Entre 2012 y 2015, hubo un crecimiento importante en el uso de este tipo de bases de datos. Y aunque desde 2016 su racha se ha quedado un poco estancada, siguen siendo también muy populares.

Bases de datos relacionales

El principio de las bases de datos relacionales se basa en la organización de la información en trozos pequeños, que se relacionan entre ellos mediante la relación de identificadores.

En el ámbito informático se habla mucho de **ACID**, cuyas siglas vienen de las palabras en inglés: **atomicidad, consistencia, aislamiento y durabilidad**. Son propiedades que las bases de datos relacionales aportan a los sistemas y les permiten ser **más robustos y menos vulnerables** ante fallos.

La base de datos relacional más usada y conocida es **MySQL** junto con **Oracle**, seguida por **SQL Server** y **PostgreSQL**, entre otras.

Bases de datos no relacionales

Como su propio nombre indica, las bases de datos no relacionales son las que, a diferencia de las relacionales, no tienen un identificador que sirva de relación entre un conjunto de datos y otros.

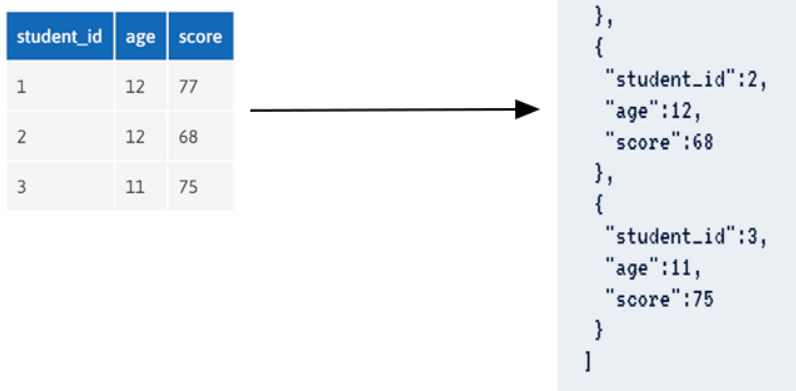
Como veremos, la información se organiza normalmente mediante documentos y es muy útil cuando **no tenemos un esquema exacto de lo que se va a almacenar**.

La indiscutible reina del reciente éxito de las bases de datos no relacionales es **MongoDB** seguida por **Redis**, **Elasticsearch** y **Cassandra**.

MySQL_ Introducción a las bases de datos

Formatos

La información puede organizarse en **tablas** o en **documentos**. Cuando organizamos información en un Excel, lo hacemos en formato tabla y, cuando los médicos hacen fichas a sus pacientes, están guardando la información en documentos. Lo habitual es que las bases de datos **basadas en tablas** sean bases de datos **relacionales** y las **basadas en documentos** sean **no relacionales**, pero esto **no tiene que ser siempre así**. En realidad, una tabla puede transformarse en documentos, cada uno formado por cada fila de la tabla. Solo es una cuestión de visualización.



Los dos esquemas de la imagen contienen exactamente la misma información. Lo único que cambia aquí es el formato: cada documento de la figura de la derecha es una fila de la figura de la izquierda.

Se ve más claro en la tabla, ¿verdad? Lo que pasa es que a menudo en una base de datos no relacional **una unidad de datos puede llegar a ser demasiado compleja como para plasmarlo en una tabla**. Por ejemplo, en el documento JSON de la imagen que se muestra a continuación, al tener elementos jerárquicos, es más difícil plasmarlo en una tabla plana. Una solución sería plasmarlo en varias tablas y, por tanto, necesitar de relaciones.

MySQL_ Introducción a las bases de datos

```
[
{
  "student_id":1,
  "age":12,
  "subjects":{
    "mathematics":{
      "scores":[7,8,7,10],
      "final_score":8
    },
    "biology":{
      "scores":[6,6,5,7],
      "final_score":6
    }
  }
}
]
```

Esto explica por qué las bases de datos relacionales suelen servirse de tablas y las no relacionales de documentos JSON. En cualquier caso, a día de hoy, **las bases de datos más competitivas suelen permitir, de una forma u otra, operaciones de los dos tipos**. Por ejemplo, el servicio de base de datos en la nube **BigQuery** que ofrece Google es, en principio, una base de datos de lenguaje de consulta SQL, por lo que permite fácilmente relacionar varias tablas, pero, a su vez, permite insertar elementos jerárquicos JSON, más propios de bases de datos no relacionales.

La diferencia entre el éxito y el fracaso recae, sobre todo, en el diseño del modelo. Es decir, si se decide que el mejor enfoque es usar una base de datos relacional, no es suficiente con meter la información a lo bruto en una base de datos relacional y esperar a que se relacione sola, porque eso no va a ocurrir. De nada sirve elegir la base de datos más apropiada para nuestro sistema, si luego no se hace un buen diseño.

¿Un poco perdidos? ¡Pasemos a la acción!

Ejemplo práctico: base de datos relacional

Imaginemos que **tenemos una plataforma online que ofrece cursos de idiomas**. Los clientes contratan o se suscriben al idioma y al nivel que más les puede interesar, y, además, tienen la opción de elegir qué tipo de suscripción quieren: mensual, trimestral o anual. Y dependiendo de esta opción, se les aplicará un descuento u otro.

El **primer diseño** de base de datos que propongo es una tabla donde cada fila corresponde con un servicio contratado por un cliente. Toda la información está contenida en una sola tabla, por tanto, **no es relacional**.

MySQL_ Introducción a las bases de datos

fecha	cliente	idioma	nivel	suscripción	precio	descuento_%	precio final
25/06/2018	Pedro	Inglés	Intermedio	Mensual	7	0	7
25/06/2018	Pedro	Chino	Principiante	Mensual	9	0	9
01/07/2018	Aurelia	Francés	Avanzado	Anual	8	25	6
03/07/2018	Federico	Inglés	Intermedio	Trimestral	7	10	6.3

Problemas que podemos encontrar con este modelo

- **No sabemos si el Pedro de la primera fila es el mismo cliente que el Pedro de la segunda fila** o si son dos clientes con el mismo nombre. Sí, podríamos incluir el e-mail para que haga de identificador único, pero es una solución cogida con pinzas.
- Si algún precio o descuento cambia, **hay que modificarlo en todas las filas en las que aparece** y, si no se hace correctamente, puede dar lugar a discrepancias. No tiene sentido que la información esté duplicada de esta manera.
- Si un cliente cambia su suscripción, **habría que cambiar tanto el campo de suscripción como el precio**. Y también puede dar lugar a discrepancias si no se hace correctamente.
- Al tener la columna de “precio final” **se está duplicando información**, ya que se puede calcular fácilmente con las columnas “precio” y “descuento_%”. Esto también puede dar lugar a discrepancias.
- No hay manera de saber **qué idiomas y niveles hay disponibles**, ni cuál es su precio hasta que alguien lo contrate.

¿Cómo solucionamos todos estos problemas?

Parece que esta situación está pidiendo a gritos **un diseño de base de datos relacional, donde se recoja la información en varias tablas y no solo en una**.

Empezamos con la primera tabla; esta contendrá solamente la información del **cliente**.

cliente_id nombre_cliente

1	Pedro
2	Aurelia
3	Federico

Por otro lado tenemos la tabla contenedora de las **clases disponibles**. Cada una con su nivel y precio base.

programa_id	idioma	nivel	precio
1	alemán	principiante	7
2	chino	principiante	9
3	francés	avanzado	8
4	inglés	intermedio	7

MySQL_ Introducción a las bases de datos

En esta tabla podemos ver todos los cursos disponibles. En el diseño principal, como nadie se había suscrito al curso de alemán, ni siquiera podíamos saber que existía.

A este precio base luego se descontará un porcentaje, según la suscripción que los clientes elijan.

suscripcion_id	tipo	descuento_%
1	Mensual	0
2	Trimestral	10
3	Anual	25

Y por último, tenemos la **tabla que relaciona todo**: a cada cliente con la clase o clases contratadas y el tipo de suscripción.

	id cliente_id	programa_id	suscripcion_id
1	1	4	1
2	1	2	1
3	2	3	3
4	3	4	2

Pedro, que es el cliente con identificador 1, se había suscrito mensualmente (id=1) a inglés intermedio (id=4) y chino principiante (id=2). Por eso, en las dos filas en las que el identificador de cliente “client_id” es 1, el identificador de programa es 4 y 2, y el identificador de suscripción es un 1.

Fijaos también que **no hemos apuntado el precio final en ningún lado**. No es necesario, ya que, conociendo el precio del programa y el descuento de la suscripción elegida, se puede calcular de inmediato. Y así evitaremos duplicidad y la posibilidad de discrepancias en nuestros datos.

¡Y ya lo tenemos! No es tan difícil, ¿no?

Antes de continuar, **os propongo un ejercicio: ¿cómo haríais para incluir la posibilidad de tener cupones de descuento?** Pensad que estos cupones son canjeables por cada curso que se contrata y cada uno puede proporcionar un descuento diferente.

MySQL_ Introducción a las bases de datos

Ejemplo práctico: bases de datos no relacionales

Quizá os estéis preguntando “**si las bases de datos relacionales son tan prácticas, ¿en qué situaciones es buena idea trabajar con las no relacionales?**”. Si algo tienen de malo las bases de datos relacionales, es que son como Sheldon Cooper, **tienen que saber de antemano qué es y cómo es lo que van a almacenar**. En cambio, las bases de datos no relacionales son más flexibles, se lo *tragan* todo, sin importar su estructura.

Imaginemos que **hemos mandado unas máquinas al espacio para que nos reporten qué es lo que encuentran en su viaje**. Obviamente, no sabemos a ciencia cierta qué se van a encontrar. De alguna forma, tienen una inteligencia artificial instalada que reconoce los objetos con los que se va encontrando y también tienen sensores instalados. Pero no sabemos bien qué miden, ya que cada máquina tiene sensores diferentes. Cada 24 horas envían un resumen de lo que han visto durante el día.

```
{
  "maquina_id":1,
  "timestamp":149992693000,
  "coordenadas":"75988823.567, 55375867.098, 12676444.311",
  "encontrado":[
    "roca",
    "agua",
    "roca",
    "roca",
    "algo que parece un animal",
    "roca"
  ],
  "temperatura":{
    "min":-50,
    "max":-49
  },
  "ruido":{
    "min":72,
    "max":4549
  }
}
{
  "maquina_id":2,
  "timestamp":1499925677000,
  "coordenadas":"66635675.920, 78021134.727, 53580995.751",
  "temperatura":{
    "min":-50,
    "max":-49
  },
  "humedad":{
    "min":2%,
    "max":5%
  }
}
```

MySQL_ Introducción a las bases de datos

Cada uno de estos documentos JSON contiene la información reportada en cada envío por cada máquina. La máquina con identificador 1 está reportando datos de temperatura y ruido, mientras que la de identificador 2 reporta temperatura y humedad. No sabemos qué sensores tendrá instalados la siguiente y, mucho menos, qué y cómo reportarán las máquinas que aún no se han mandado y los ingenieros están montando.

No merece la pena ponerse a diseñar una base de datos relacional para almacenar esta información. En este caso, **lo mejor es dejar a una base de datos no relacional que se trague todo lo que las máquinas reportan, tal cual.**

Además, la **finalidad** del sistema es meramente **científica** y no se contempla la existencia de usuarios a los que se les deba la garantía de consistencia que ofrecen las bases de datos SQL. Simplemente, **se quiere almacenar todo para un futuro análisis.**

Una vez almacenados en la base de datos no relacional se podrá pedir y visualizar la información de diferentes maneras. Y si en algún momento se necesita consumir los datos de una forma más estructurada, **siempre podremos procesar y volcar la información a una base de datos relacional.** Pero es que, muy probablemente, no sea necesario.