Manejando índices

Es muy frecuente la utilización de índices en las bases de datos. Pero... ¿qué son los índices? ¿para qué sirven?.

Si nos planteamos cuál es la utilidad del **índice** de un libro la respuesta sería:

- Facilitar la búsqueda de datos
- Agilizar esa búsqueda

La utilidad de los índices en la bases de datos es la misma.

Tipos de índices

Si seguimos con el ejemplo del libro veremos que caben las posibilidades de que tenga:

- Un solo índice
- Varios índices

Algunos libros sólo contienen un **indice de contenidos**. Sin embargo, es frecuente que también dispongan de otros, tales como: *índices analíticos*, *índices onomásticos*, etcétera.

Cuando existe **un solo índice** es obvio que puede decirse de él que es **único** y cuando existen varios podemos decir que el *índice de contenidos* es el **índice principal** y los *demás* son **índices** sin más o *índices auxiliares*.

Coincidirás con nosotros en que en el *Índice de contenidos* de un libro de texto sólo existe *una referencia* al *Tema XIII* en la que puede decir: *Tema XIII página 314*.

También coincidirás en que si en ese índice, además de lo anterior, dijera: *Tema XIII página 714* nos encontraríamos en una **situación confusa** que nos obligaría a preguntarnos: *¿donde está el Tema XIII? ¿En la página 314? ¿En la 714? ¿En ambas?*.

Es por eso que las **tablas** de las bases de datos *no admiten nunca valores duplicados* ni en los *índices únicos* ni tampoco en los *índices principales*.

Los *índices auxiliares* tienen un comportamiento distinto. El índice onomástico de un libro puede hacer referencia a varias páginas y puede **tener duplicados**.

Por ejemplo: en un manual de Word puede existir un índice onomástico en el que se asocie la palabra **macros** con las páginas *37*, *234* y *832* siempre que en esas páginas existan contenidos que aludan a la palabra *macro*.

Es por eso que las **tablas** de las *bases de datos* también **admiten duplicados** cuando se trata de *índices auxiliares*.

Sintaxis de la definición de índices MySQL

Tanto al *crear una tabla* como al *modificarla* se pueden *añadir índices* de la misma forma que también se pueden *insertar campos*.

Añadir registros a una tabla

Las sentencias MySQL que permiten añadir registros a una tabla son las siguientes: **INSERT tabla (***campo1,campo2,..***) VALUES (***valor1,valor2,..***)**

Vamos a desarrollar un ejemplo completo de creación de una tabla e inserción de registros utilizando diversos métodos. Para ello seguiremos el siguiente proceso:

1.- Creación de la tabla

Empezaremos **creando una tabla** a la que llamaremos **demo4** y que contendrá los siguientes campos:

- **Contador**, que será de tipo **TINYINT(8)** (número entero con un máximo de 8 dígitos) que contenga solo **valores positivos**, que se rellene automáticamente con **ceros por la izquierda** y que **se autoincremente** cada vez que añadimos un registro.
- **DNI**, destinado a recoger valores de números de DNI y que debe poder contener *un máximo de ocho caracteres*.
  Lo definiremos de tipo **CHAR** porque sus valores, pese a ser numéricos, *nunca van a requerir ningún tratamiento aritmético*.
- **Nombre**, **Apellido1** y **Apellido2** serán tres campos tipo **VHAR** de tamaños máximos respectivos de **20**, **15** y **15** caracteres.
  Su finalidad la evidencian los *nombres de campo*. Les asignaremos el *flag* **NOT NULL** aunque no sea *totalmente correcto* dado que existen países en los que se utiliza un solo apellido.
- **Nacimiento** será un campo tipo **DATE** al que asignaremos como **valor por defecto** el de **1970-12-21**. Recuerda que MySQL trata las fechas en ese orden (año-mes-dia) y que admite como separadores tanto **-** como **/** por lo que son válidas entradas con cualquiera de estos formatos: **1970-12-21** ó **1970/12/21**, aunque esta segunda es convertida automaticamente al primer formato por MySQL.
- **Hora** será un campo tipo **TIME** al que asignaremos como **valor por defecto** el de **00:00:00**. Recuerda que MySQL trata las horas en ese orden (hh:mm:ss) y que sólo admite como separador **:**.
  Este campo está destinado a recoger *la hora de nacimiento* que aunque no tiene demasiado sentido es una *buena excusa* para introducir este tipo de campo.
- **Sexo** será un campo tipo **ENUM** que tendrá *dos opciones*: **M** (masculino) y **F** (femenino) y al que asignaremos **M** como **valor por defecto**.
- **Fumador** será un campo tipo **CHAR(0)** que por su estructura -cadena de longitud cero- tendrá *dos únicas opciones* de valor: **NULL** ó **""**, que, como veremos, son *valores distintos* para MySQL
- **Idiomas** será un campo tipo **SET** definido para los valores: **Castellano**, **Francés**, **Inglés**, **Alemán**, **Búlgaro** y **Chino** y que podrá contener, como ocurre con los campos de este tipo, **ninguno**, **uno** o **varios** de los valores de la lista.
- **Índice primario** será tratado como tal el campo **DNI** ya que se trata de un valor **único** para cada persona y que, como tal, **no puede tener duplicados**
- **Índices auxiliares**. Para ejemplificar su tratamiento consideraremos el campo **Contador** como **índice secundario**.

El *código fuente* del fichero que genera esta tabla puedes verlo aquí debajo

Ver código fuente

Crear tabla demo4

2.1.- Añadir un registro

Cuando se añade un registro en una tabla los valores pueden añadirse en la propia sentencia MySQL o ser recogidos de los valores de variables PHP previamente definidas.

En este ejemplo tienes el *código fuente* del primero de los casos, en el que se añade a la tabla anterior un registro cuyos valores son:

DNI	Nombre	Apellido1	Apellido2	Nacimiento	Sexo	Hora	Fumador	Idiomas
1234	Lupicinio	Servidor	Servido	1954-11-23	M	16:24:52	NULL	3

Ver código fuente	Añadir regitro

Presta atención a los siguientes aspectos:

- En la sentencia **no se alude** al campo **Contador**. La razón es que se trata un campo AUTOINCREMENTAL y en ese tipo de campos los valores de los registros **se escriben automáticamente** cada vez que se añade uno nuevo.
- El registro **no se añadiría** si el valor de **DNI *coincidiera con otra ya existente*** en la tabla. Recuerda que habíamos definido ese campo como **índice primario**. Si no hubiéramos incluido **el aviso de error** no tendríamos ninguna referencia sobre el **éxito de la inserción** y no detectaríamos el **problema de duplicidad**. Sencillamente ocurriría que *el registro no se añadiría* pero no nos enteraríamos de tal circunstancia.
- Si en los valores de **nombre y apellidos** hubiéramos insertado textos **más largos** del tamaño establecido para ellos al crear la tabla, las cadenas **se recortarían** y

La sintaxis (dentro de la sentencia MySQL que crea o modifica una tabla) es la siguiente:

PRIMARY KEY(*campo***)**

donde **campo** es el *nombre del campo* que se establece como **índice principal** de la tabla.

El *nombre del campo* no va entrecomillado y PRIMARY KEY (*campo*) se añade dentro de la sentencia **CREATE** como si se tratara de un campo más, delimitado por *comas*, salvo que estuviera al final de la sentencia CREATE, en cuyo caso se omitiría la coma final.

En el código fuente de la creación de la tabla que tienes a la derecha puedes ver un ejemplo práctico de la sintaxis.

Solo puede definirse **un índice primario por tabla** y el campo utilizado ha de ser un campo **no nulo**.

UNIQUE *nombre* **(***campo***)**

Similar a PRIMARY KEY, en cuanto a que no admite valores duplicados, pero con dos diferencias importantes. UNIQUE *permite la creación de más de un índice de este tipo por tabla* y además *no requiere* que los campos sobre los que se define sean *no nulos*.

INDEX *nombre* **(***campo***)**

Con esta sintaxis se crea un *índice secundario* que debe tener un *nombre* (la posibilidad de que existan varios obliga a identificarlos de esta forma) y -como en los casos anteriores- el campo que va a ser utilizado como índice.

INDEX *nombre* **(***campo***(***n***))**

Es un caso particular del anterior utilizable sólo en el caso de que el campo índice sea tipo **CHAR** y **VARCHAR** que permite **indexar por los *n* primeros caracteres** de esas cadenas.

El valor de **n** ha de ser:

n <= 256

dado que el *tamaño máximo de un índice* está limitado en MySQL a **256 bytes**.

Otra limitación de MySQL es el *número máximo de índices de una tabla* que no puede ser mayor de *dieciséis*.

Los errores MySQL

PHP dispone de dos funciones que nos permiten detectar si una *sentencia MySQL* se ha *ejecutado correctamente* o si se *ha producido algún error*.

Son las siguientes:

mysql_errno(*$enl***)**

Indica **el número de error** que se ha producido en la *transacción* MySQL realizada a través del *identificador de enlace* **$enl**.

Cuando el **número de error** es **CERO** significa que **no se ha producido error**.

Otros valores bastante usuales son los siguientes:

Error número 1050

sólo se añadirían -de izquierda a derecha- los **n** primeros caracteres de la cadena.

- Si en la fecha de nacimiento hubiéramos introducido una **cadena vacía** nos habría puesto el **valor por defecto**, pero **si hubiéramos introducido un valor no válido** nos habría escrito **0000-00-00**.
  Serían valores **no válidos** en este caso:
  - Los que tuvieran como valor de mes alguno no perteneciente al intervalo **[1,12]**.
  - Los que tuvieran como valor de día un valor **no válido** en concordancia con el mes y el año. Admitiría **29** en un mes de **febrero** sólo en el caso de que el año fuera **bisiesto**.
  - Cuando la secuencia **no coincidiera** con esta **AAAA-MM-DD**.
  - Cuando los *separadores* no fueran (**-**) o (***/***)
- En el campo **Sexo** si hubiéramos introducido una **cadena vacía** nos habría puesto **M** (valor por defecto) pero si hubiéramos intentado introducir un valor que no fuera **M** ni **F** nos habría insertado una **cadena vacía**.
- El campo **hora** se comportaría de idéntica forma al de fecha salvo que aquí la secuencia es: **hh:mm:ss**, que la **hora** debe pertenecer al intervalo **[0,23]**, los **minutos** y los **segundos** a **[0,59]** y que el único separador válido en este caso es (**:**).
- El campo **Fumador** requiere **particular** atención ya que se trata de un campo **CHAR(0)** que sólo admite dos valores: NULL o **cadena vacía**, que *como recordarás* son distintos para MySQL.
  - Para introducir el valor **NULL** -utilizando el procedimiento de inserción de este ejemplo- tienes dos posibilidades, o escribir **NULL *sin ponerlo entre comillas*** -tal como lo he hecho en el ejemplo o escribir **'\N'** que como ves, es el carácter especial **\N** esta vez ***colocado entre comillas***.
  - Para introducir **una cadena vacía** en este campo bastaría con que pusieramos **''** (¡ojo no es una comilla doble es la comilla sencilla (') repetida dos veces!
    Date cuenta de que si pusiéramos **comillas dobles** tendríamos un **error** ya que hay unas comillas dobles delante del **INSERT** que se cierran al final de la sentencia MySQL y que **no podemos volver a escribirlas** entre ambas para evitar un **falso cierre** de la cadena que contienen.

- En este caso el valor del campo **Idiomas** puede contener valores decimales comprendidos entre **0** y **64** o entre sus equivalentes binarios que son **0** y **111111**. El valor **64** lo justifica el hecho de que **son seis los elementos** que puede contener el campo (hemos definido: Castellano, Francés, Inglés, Alemán, Búlgaro y Chino que son **seis** y que el caso de insertarlos todos requeriría el número binario **111111**, cuyo valor decimal es precisamente 64.
  El valor **3** significa lo mismo que su equivalente **binario** (**11**) o mejor (**000011**) lo cual quiere decir que, **como el primer carácter de la derecha es uno** el campo toma **el primer elemento de la lista** (Castellano), como el segundo (de derecha a izquierda) también es **uno** tomará también el segundo elemento de la lista (Francés) y por ser **cero** todos los demás no tomará ningún otro valor de la lista, con lo que la cadena resultante sería en este caso **Castellano, Francés**.
- Fíjate también en que el valor **3** no lo hemos puesto entre comillas porque se trata de una expresión decimal. ¿Qué ocurría si hubiera puesto **11**? ¿Lo habría interpretado como **once** (decimal) o como **tres** (binario)?.
  La solución es simple: **'11'** (entre comillas) sería interpretado como binario, sin comillas como decimal.
- **No es necesario introducir valores en todos los campos**, es decir, que la lista de campos de la sentencia **INSERT** puede no contenerlos a todos.
- **No es necesario** escribir el nombre de los campos **en el mismo orden** en el que fueron creados pero si es **imprescindible** que **campos** y **valores** estén escritos en la sentencia INSERT **exactamente en el mismo orden** y también que en esa sentencia el **número de campos** y el **número de valores** sea el mismo.
- Recuerda que los **values** tipo **numérico** *no se incluyen entre comillas*, mientras que los **no numéricos** tienen que **estar contenidos entre comillas** incluso en el caso de no se inserten directamente sino a través de una variable PHP.

2.2- Añadir un registro a partir de datos contenidos en variables

También es posible añadir registros a partir de valores contenidos en variables PHP. Esta opción es, sin ninguna duda, la más utilizada ya que lo habitual será **escribir** el contenido a añadir en un **form** y después -a través del **method** (POST o GET)- pasar al *script* de inserción esos valores como **variables PHP**.

Aquí tienes el *código fuente* de un ejemplo con la **tabla anterior**.

Ver código fuente	Añadir regitro

Quizá te resulten de alguna utilidad estos comentarios

- Observa en el código fuente que al **insertar las variables** en los *VALUES* de la sentencia MySQL ponemos ***cuando se trata de valores tipo cadena*** '$variable' (el nombre de la variable entre comillas) y cuando se trata de ***valores numéricos*** **sin comillas**.
- Si quieres introducir el valor **NULL** en un campo tipo VAR(0) define la variable así: **$var="\N"** y si quieres introducir una **cadena vacía** defínela de este otro modo: **$var="''"** -comillas dobles (") seguidas de **dos** comillas sencillas (') y para terminar otras comillas dobles (").

3.- Variantes de la sentencia INSERT

La sentencia INSERT cuya sintaxis se indica más arriba como:

INSERT *tabla* (*campo1*,*campo2*,..) VALUES (*valor1*, *valor2*,..)

permite algunos modificadores opciones tales como:

Indica que hemos tratado de crear una tabla ya existente.

Error número 1062

Indica que hemos tratado de introducir un valor con clave duplicada. Aparecerá cuando tratemos de introducir un valor ya existente en un campo con índice único o principal.

mysql_error(*$enl*)

Devuelve la *descripción del error*. Cuando el número de error es CERO devuelve una **cadena vacía**.

Resulta de muchísima utilidad para depurar *scripts*.

Ejercicio nº 39

Como otra práctica de creación de tablas trata de crear una tabla con los –supuestos– datos más significativos de todos los compañeros de este Curso.

INSERT *[LOW_PRIORITY | DELAYED] [IGNORE] tabla (campo1,..) VALUES (valor1,..)*

de ellos **LOW_PRIORITY** y **DELAYED** son incompatibles por lo que solo cabe **uno** u **otro** pero **ambos a la vez**.

Veamos su utilidad. La **inserción de registros** y la **lectura de una tabla** son procesos incompatibles, pero cabe la posibilidad de que se **intenten ejecutar** simultáneamente. No olvides que *estamos en Internet* y es perfectamente posible que desde dos ordenadores distintos, dos personas distintas **estén accediendo a la misma tabla** simultáneamente y que uno de los accesos sea de **escritura**, es decir: **añadir**, **modificar** o **borrar** campos en la tabla

¿Quién tiene *preferencia de paso*? ¿Quien tiene que esperar?. Si la opción **LOW_PRIORITY** está activada, **el proceso de escritura esperará a que terminen los procesos de lectura activos** pero se si está activa la opción **DELAYED** el proceso de lectura **se interrumpirá automáticamente** para *ceder el paso* al de escritura.

Respecto a la opción **IGNORE** tiene utilidad cuando se trata de **realizar** una **secuencia de inserciones**. Si **no está activa** en el momento en el que aparezca **una clave duplicada** se **interrumpirá el proceso de inserción**, por el contrario, **si estuviera activa** el proceso de inserción **continuará** con los siguientes registros de la secuencia, aunque -como es lógico- seguirán sin insertarse los registros con clave duplicada.

4.- Tablas para pruebas

Aquí tienes comentado un *script* que permite agregar **aleatoriamente** y de forma **automática** registros a la tabla **demo4**.

Dado que en las páginas siguientes trataremos de consultas va a resultarnos muy cómodo poder rellenarlas de forma automática.

Ver código fuente

Insertar datos demo4