2 CREACIÓN DE BASES DE DATOS

2.1 Introducción

Crear la base de datos implica indicar los archivos y ubicaciones que se utilizarán para la misma, además de otras indicaciones técnicas y administrativas de las que no son objeto en el presente curso.

Lógicamente sólo es posible crear una base de datos si se tienen privilegios de Administrador de la Base de Datos (**DBA**: DataBase Administrator)

- **Crear una Base de Datos**

El comando SQL de creación de una base de datos es CREATE DATABASE. Este comando crea una base de datos con el nombre que se indique.

Ejemplo:

```
CREATE DATABASE Ejemplo;
```

Pero normalmente se indican más parámetros. Ejemplo (parámetros de Oracle):

```
CREATE DATABASE Ejemplo
LOGFILE prueba.log
MAXLOGFILES 25
MAXINSTANCES 10
ARCHIVELOG
CHARACTER SET WIN1214
NATIONAL CHARACTER SET UTF8
DATAFILE prueba1.dbf AUTOEXTEND ON MAXSIZE 500MB;
```

- **Borrar una Base de Datos**

El comando SQL de borrado de una base de datos es DROP DATABASE. Este comando borra una base de datos existente.

Ejemplo: DROP DATABASE Ejemplo;

- **Consultar Bases de Datos**

Es muy normal consultar las bases de datos existentes en un SGBD. Para ello se suele utilizar la instrucción Show Databases

- **Uso Bases de Datos**

Para utilizar una base de datos se usa la instrucción USE DATABASE.

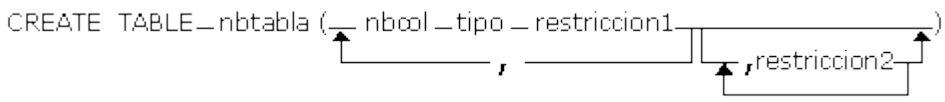
Ejemplo: USE DATABASE Ejemplo;

2.2 Creación de Tablas

2.2.1 Sentencia Create Table

La sentencia **CREATE TABLE** sirve para **crear la estructura de una tabla** no para rellenarla con datos, permite **definir las columnas** que tiene **y ciertas restricciones** que deben cumplir esas columnas.

La **sintaxis** es la siguiente:



Dónde:

- nbtabla: nombre de la tabla a definir
- nbcol: nombre de la columna a definir
- tipo: **tipo de dato** de la columna, todos los datos almacenados en la columna deberán ser de ese tipo.

Una **restricción** consiste en la definición de una **característica adicional que tiene una columna** o una combinación de columnas, suelen ser características como valores no nulos (campo requerido), definición de índice sin duplicados, definición de clave principal y definición de clave foránea (clave ajena o externa, campo que sirve para relacionar dos tablas entre sí).

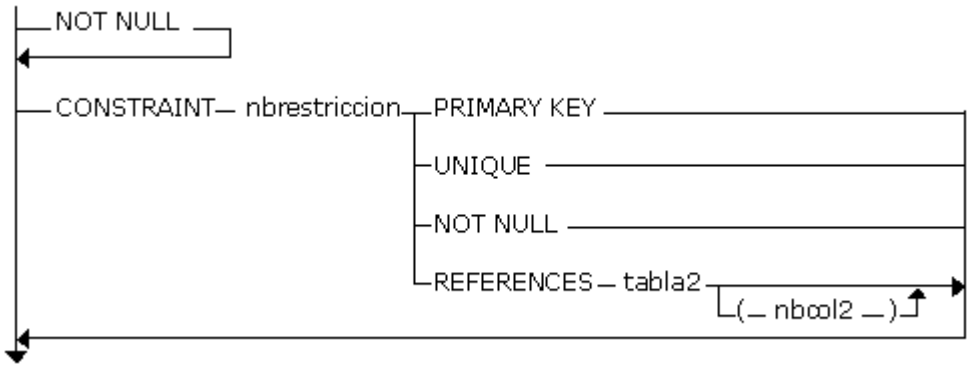
- restricción1: una **restricción de tipo 1** es una restricción que aparece **dentro de la definición de la columna** después del tipo de dato y **afecta** a **una columna**, la que se está definiendo.
- restricción2: una **restricción de tipo 2** es una restricción que se define **después de definir todas las columnas** de la tabla y **afecta** a **una** columna **o** a una **combinación de columnas.**

Para escribir una sentencia **CREATE TABLE** se empieza por indicar el nombre de la tabla que se quiere crear y a continuación entre paréntesis se indica separando por comas las definiciones de cada columna de la tabla, la definición de una columna consta de su nombre, el tipo de dato que tiene y se le puede añadir si queremos una serie de especificaciones que deberán cumplir los datos almacenados en la columna, después de

definir cada una de las columnas que compone la tabla se pueden añadir una serie de restricciones, esas restricciones son las mismas que se pueden indicar para cada columna pero ahora pueden afectar a más de una columna por eso tienen una sintaxis ligeramente diferente.

RESTRICCIÓN A NIVEL DE COLUMNA (Restricción tipo 1)

Una **restricción de tipo 1** se utiliza para indicar una característica de la columna que se está definiendo, tiene la siguiente sintaxis:



La cláusula **CONSTRAINT** sirve para definir una **restricción** que se podrá eliminar cuando se quiera sin tener que borrar la columna. A cada restricción se le asigna un nombre que se utiliza para identificarla y para poder eliminarla cuando se quiera.

Como restricciones se tienen la de clave primaria (clave principal), la de índice único (sin duplicados), la de valor no nulo, y la de clave foránea.

La cláusula **PRIMARY KEY** se utiliza para definir la columna como **clave principal de la tabla**. Esto supone que **la columna no puede contener valores nulos ni pueden haber valores duplicados** en esa columna, es decir que dos filas no pueden tener el mismo valor en esa columna.

En una tabla **no puede haber varias claves principales**, por lo que no se puede incluir la cláusula **PRIMARY KEY** más de una vez, en caso contrario la sentencia da un error. No hay que confundir la definición de varias claves principales con la definición de una clave principal compuesta por varias columnas, esto último sí está permitido y se define con una restricción de tipo 2.

La cláusula **UNIQUE** sirve para definir un **índice único** sobre la columna. Un índice único es un índice que **no permite valores duplicados**, es decir que si una columna tiene definida un restricción de **UNIQUE** no podrán haber dos filas con el mismo valor en esa columna. Se suele emplear para que el sistema compruebe el mismo que no se añaden valores que ya existen, por ejemplo si en una tabla de clientes se quiere asegurar que dos

clientes no puedan tener el mismo D.N.I. y la tabla tiene como clave principal un código de cliente, se definirá la columna dni con la restricción de **UNIQUE.**

La cláusula **NOT NULL** indica que **la columna no podrá contener un valor nulo**, es decir que se deberá rellenar obligatoriamente y con un valor válido (equivale a la propiedad requerido Sí de las propiedades del campo).

Ejemplo:

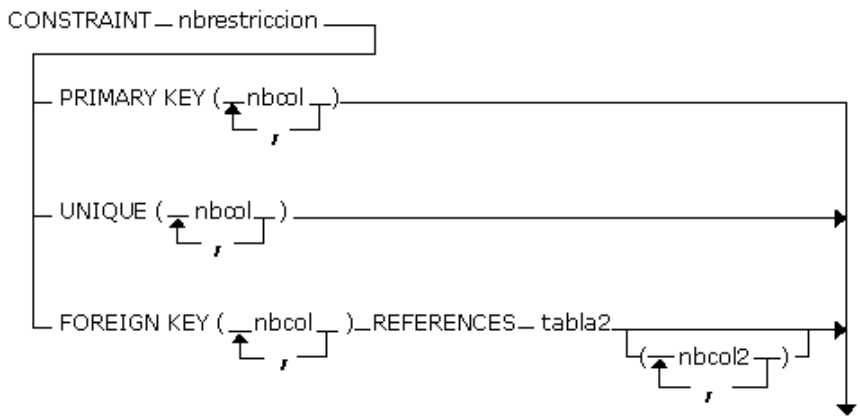
```
CREATE TABLE tab1 (
    col1 INTEGER CONSTRAINT pk PRIMARY KEY,
    col2 CHAR(25) NOT NULL,
    col3 CHAR(10) CONSTRAINT uni1 UNIQUE,
    col4 INTEGER,
    col5 INT CONSTRAINT fk5 REFERENCES tab2 );
```

Con este ejemplo se está creando la tabla *tab1* compuesta por: una columna llamada *col1* de tipo entero definida como clave principal, una columna *col2* que puede almacenar hasta 25 caracteres alfanuméricos y no puede contener valores nulos, una columna *col3* de hasta 10 caracteres que no podrá contener valores repetidos, una columna *col4* de tipo entero sin ninguna restricción, y una columna *col5* de tipo entero clave foránea que hace referencia a valores de la clave principal de la tabla *tab2*

RESTRICCIÓN A NIVEL DEL TABLA (Restricción tipo 2)

Una **restricción de tipo 2** se utiliza **para definir una característica que afecta a una columna o a una combinación de columnas** de la tabla que estamos definiendo, **se escribe después de haber definido todas las columnas** de la tabla.

Tiene la siguiente **sintaxis:**



La sintaxis de una restricción de tipo 2 es muy similar a la **CONSTRAINT** de una restricción 1 la diferencia es que ahora tenemos que indicar sobre qué columnas se quiere definir la restricción. Se utilizan obligatoriamente las restricciones de tipo 2 cuando la restricción afecta a un grupo de columnas o cuando se quieren definir más de una **CONSTRAINT** para una columna (sólo se puede definir una restricción1 en cada columna).

La cláusula **PRIMARY KEY** se utiliza para definir la **clave principal** de la tabla. Después de las palabras **PRIMARY KEY** se indica entre paréntesis el nombre de la columna o las columnas que forman la clave principal. Las columnas que forman la clave principal no pueden contener valores nulos ni pueden haber valores duplicados de la combinación de columnas, por ejemplo la tabla pedidos de los ejemplos tiene una clave principal formada por idfab e idproducto, pues no pueden haber dos filas con la misma combinación de idfab con idproducto (*aci,0001* por ejemplo) pero sí pueden haber dos filas con el valor *aci* en la columna *idfab* si tienen valores diferentes en la columna *idproducto*, y pueden haber dos filas con el mismo idproducto pero distinto idfab.

En una tabla **no puede haber varias claves principales**, por lo que no podemos indicar la cláusula **PRIMARY KEY** más de una vez, en caso contrario la sentencia da un error.

La cláusula **UNIQUE** sirve para definir un **índice único** sobre una columna o sobre una combinación de columnas. Un índice único es un índice que **no permite valores duplicados**. Si el índice es sobre varias columnas no se puede repetir la misma combinación de valores en dos o más filas. Se suele emplear para que el sistema compruebe el mismo que no se añaden valores que ya existen.

La cláusula **FOREIGN KEY** sirve para definir una **clave foránea** sobre una columna o una combinación de columnas. Una clave foránea es una columna o conjunto de columnas que **contiene un valor que hace referencia a una fila de otra tabla**, en una restricción 1 se puede definir con la cláusula **REFERENCES**. Para definir una clave foránea en una restricción de tipo 2 se debe empezar por las palabras **FOREIGN KEY** después se indica entre paréntesis la/s columna/s que es clave foránea, a continuación la palabra reservada **REFERENCES** seguida del nombre de la tabla a la que hace referencia, opcionalmente se puede indicar entre paréntesis el nombre de la/s columna/s donde tiene que buscar el valor de referencia, por defecto coge la clave principal de la tabla2, si el valor que tiene que buscar se encuentra en otra/s columna/s de tabla2, entonces se debe escribir el nombre de esta/s columna/s entre paréntesis, además sólo se puede utilizar una columna (o combinación de columnas) que esté definida con una restricción de **UNIQUE**, de lo contrario la sentencia **CREATE TABLE** dará un error.

Ejemplo:

```
CREATE TABLE tab1 (col1 INTEGER,
    col2 CHAR(25) NOT NULL,
    col3 CHAR(10),
    col4 INTEGER,
    col5 INT,
    CONSTRAINT pk PRIMARY KEY (col1),
    CONSTRAINT uni1 UNIQUE (col3),
    CONSTRAINT fk5 FOREIGN KEY (col5) REFERENCES tab2);
```

Con este ejemplo se está creando la misma tabla *tab1* del ejemplo de la página anterior pero ahora se han definido las restricciones utilizando restricciones de tipo 2.

2.2.2 Nombres de las Tablas

Deben cumplir las siguientes reglas (reglas de Oracle, en otros SGBD podrían cambiar):

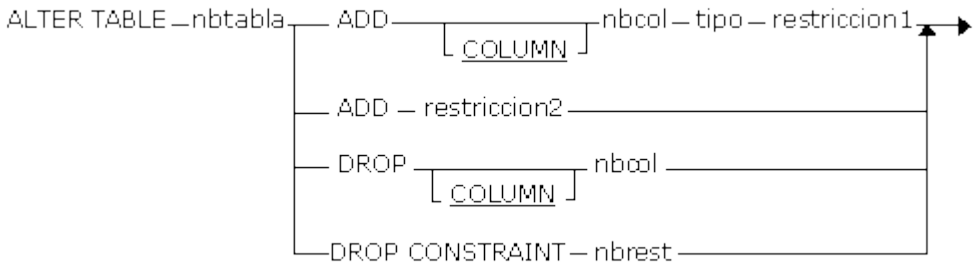
a) Deben comenzar con una letra.
b) No deben tener más de 30 caracteres.
c) Sólo se permiten utilizar letras del alfabeto (inglés), números o el signo de subrayado (también el signo $ y #, pero esos se utilizan de manera especial por lo que no son recomendados)
d) No puede haber dos tablas con el mismo nombre para el mismo esquema (pueden coincidir los nombres si están en distintos esquemas)
e) No puede coincidir con el nombre de una palabra reservada SQL (por ejemplo no se puede llamar SELECT a una tabla)

2.2.3 Modificación de Tablas

La sentencia **ALTER TABLE** sirve para **modificar la estructura de una tabla** que ya existe. Mediante esta instrucción se pueden añadir columnas nuevas o eliminar columnas. Hay que tener en cuenta que cuando se elimina una columna se pierden todos los datos almacenados en ella.

También permite crear nuevas restricciones o borrar algunas existentes. La sintaxis puede parecer algo complicada pero sabiendo el significado de las palabras reservadas la sentencia se aclara bastante; ADD (añade), ALTER (modifica), DROP (elimina), COLUMN (columna), CONSTRAINT (restricción).

La **sintaxis** es la siguiente:



- **Añadir Columnas:**

La cláusula **ADD COLUMN** (la palabra **COLUMN** es opcional) permite **añadir una columna nueva** a la tabla. Como en la creación de tabla, hay que definir la columna indicando su nombre, tipo de datos que puede contener, y si lo queremos alguna restricción de valor no nulo, clave primaria, clave foránea, e índice único, **restriccion1 es opcional** e indica una restricción de tipo 1 que afecta a la columna que estamos definiendo

Ejemplo:

ALTER TABLE tab1 ADD COLUMN col3 integer NOT NULL CONSTRAINT c1 UNIQUE

Con este ejemplo se está añadiendo a la tabla tab1 una columna llamada col3 de tipo entero, requerida (no admite nulos) y con un índice sin duplicados llamado c1.

Cuando se añade una columna lo mínimo que se puede poner sería:

ALTER TABLE tab1 ADD col3 integer

En este caso la nueva columna admite valores nulos y duplicados.

- **Añadir una nueva restricción**

Podemos utilizar la cláusula **ADD restriccion2** (**ADD CONSTRAINT.**..).

Ejemplo:

ALTER TABLE tab1 ADD CONSTRAINT c1 UNIQUE (col3)

Con este ejemplo se está añadiendo a la tabla tab1 un índice único (sin duplicados) llamado c1 sobre la columna col3.

- **Borrar una columna**

Basta con utilizar la cláusula **DROP COLUMN** (**COLUMN** es opcional) y el nombre de la columna que se quiere borrar, se perderán todos los datos almacenados en la columna.

Ejemplo:

ALTER TABLE tab1 DROP COLUMN col3

También se puede escribir:

ALTER TABLE tab1 DROP col3

El resultado es el mismo, la columna col3 desaparece de la tabla tab1.

- **Borrar una restricción**

Basta con utilizar la cláusula **DROP CONSTRAINT** y el nombre de la restricción que se desea borrar, en este caso sólo se elimina la definición de la restricción pero los datos almacenados no se modifican ni se pierden.

Ejemplo:

ALTER TABLE tab1 DROP CONSTRAINT c1

Con esta sentencia se borra la restricción c1 creada anteriormente pero los datos de la columna col3 no se ven afectados por el cambio.

2.2.4 Eliminación de Tablas

La sentencia **DROP TABLE** sirve para **eliminar una tabla**. No se puede eliminar una tabla si está abierta, tampoco se puede eliminar si el borrado infringe las reglas de integridad referencial (si interviene como tabla padre en una relación y tiene registros relacionados).

La **sintaxis** es la siguiente:

```
DROP TABLE _ nbtabla
```

Ejemplo:

DROP TABLE tab1

Elimina de la base de datos la tabla tab1.

2.3 Tipos de Datos

A la hora de crear tablas, hay que indicar el tipo de datos de cada campo. Necesitamos pues conocer los distintos tipos de datos, que variarán en virtud del SGBD usado.

Vamos a observar los **Tipos de datos existentes en Oracle y luego los de MySQL:**

A continuación se muestra un listado de los tipos de datos más utilizados en el lenguaje de programación PL/SQL:

NUMÉRICOS

NUMBER (P, S): Puede contener un valor numérico entero o de punto flotante, donde P es la precisión y S la escala. La precisión es el número de dígitos del valor, y la escala es la cantidad de dígitos a la derecha del punto decimal. La precisión máxima es 38 y la escala 127.

BINARY_INTEGER: Debido al formato interno de las variables de tipo NUMBER, las operaciones entre ellas requieren más tiempo de CPU que si utilizamos variables de tipo BINARY_INTEGER. Es recomendable su uso en contadores de bucles. Permite almacenar valores entre el rango -2147482647 y +2147482647.

CARÁCTER

VARCHAR2 (L): Guarda una cadena de longitud variable de tamaño máximo L. En PL/SQL el valor máximo del tamaño de una variable de este tipo es 32.767 bytes, sin embargo las bases de datos Oracle sólo permiten campos de hasta 4.000 bytes.

CHAR (L): Análogo al VARCHAR2 pero guarda cadenas de longitud fija. Si no se especifica L, su valor por defecto es 1. El espacio sobrante de una variable CHAR se rellena con caracteres en blanco. En PLSQL el valor máximo del tamaño de una variable de este tipo es 32.767 bytes, sin embargo las bases de datos Oracle sólo permiten columnas de hasta 2.000 bytes.

LONG: Este tipo, muy similar al VARCHAR2, se trata de una cadena de longitud variable de hasta 32.760 bytes. Los tipos de datos LONG de una base de datos Oracle son capaces almacenar hasta dos gigabytes.

FECHA E INTERVALO

DATE: Guardar información sobre la fecha, hora, día, mes, año, hora, minuto y segundo. Las variables de este tipo no son capaces de almacenar milisegundos. Su tamaño es de 7 bytes.

TIMESTAMP [P]: Con las características del tipo DATE pero además permite almacenar fracciones de segundo. El parámetro P es la precisión que por defecto es 6.

BOOLEANOS

BOOLEAN: Sólo es capaz de guardar los valores TRUE (1) y FALSE (0). Las bases de datos Oracle no permiten este tipo de dato.

RAW

RAW: Almacena datos binarios de longitud fija. Los datos de tipo RAW no implican conversiones de carácter. La longitud máxima de una variable de este tipo es 32.767 bytes, sin embargo un campo de una tabla de tipo RAW sólo admite 2.000 bytes.

LONG RAW: Análogo al tipo de dato LONG, pero como el anterior no implican conversiones de carácter.

ROWID

ROWID: Este tipo de dato sirve para almacenar identificadores únicos de registros. Este identificador es con el que trabaja internamente la base de datos Oracle para identificar dichos registros.

Estos no son los únicos tipos de datos permitidos en PLSQL, si alguien está interesado en ampliar la información puedes verlo en la página de Oracle.

TIPOS DE DATOS EN MYSQL

Uno de los conceptos básicos que debemos tener presente siempre a la hora de trabajar con bases de datos **mySQL** (o cualquier otra) es el tipo de datos que podemos utilizar para introducir registros en cada una de las tablas de la base de datos.

¿Tipos de Datos?

Creo que este concepto es muy fácil de entender. La información podemos representarla por medio de símbolos numéricos, alfanuméricos, formatos de fecha, hora, binarios, etc. Todas estas clases o divisiones son tipos de datos. Si tenemos un número que nos indica la cantidad de dinero que disponemos en una cuenta bancaria en Suiza diremos que tenemos un tipo de datos numérico. Nuestra fecha de cumpleaños es un tipo de dato de fecha, etc.

mySQL distingue una serie de tipos de datos que podremos utilizar a la hora de crear los campos que formarán nuestras tablas. Veamos a continuación estos tipos:

Tipo Texto (Char(x), Varchar(x), Text, TinyText, MediumText, LongText)

Char(x)	Tipo de datos que admite caracteres alfanuméricos. La longitud de este campo varía entre 1-255 y está delimitado a la longitud especificada entre paréntesis **(x)** en el momento de la creación del campo de la tabla. Al introducir datos en este campo siempre se solicitará el número de caracteres especificados. Si creamos un campo con **Char(5)** deberemos introducir cinco caracteres cada vez que incluyamos un dato en ese campo. Si incluimos menos, **mySQL** rellenará los caracteres que faltan hasta el número indicado con espacios.
Varchar(x)	Tipo de datos que admite caracteres alfanuméricos. Su uso es similar a **Char(x)**. A la hora de definir un campo de datos Varchar deberemos especificar el número máximo de caracteres que podrá aceptar en la entrada de datos, donde x es un número entre 1-255. A diferencia de **Char**, este tipo de datos es variable en su longitud, admitiendo entradas inferiores a la establecida.
Text, TinyText, MediumText, LongText	Mediante la declaración de este tipo de datos se admiten la inclusión de cadenas alfanuméricas "case-insensitive" de longitudes variables. **TinyText** admite un máximo de 255 caractéres, **Text** admite 65.535, **MediumText** permite introducir textos de hasta 16.777.215 caracteres, **LongText** nos ofrece la posibilidad de incluir un máximo de 4.294.967.295 caracteres. Estos campos no necesitan de especificaciones de longitud a la hora de ser declarados.

Tipo Binario (Blob, TinyBlob, MediumBlob, LongBlob)

Blob	Un tipo de datos **Blob** es un objeto binario que puede almacenar cualquier tipo de datos o información, desde un archivo de texto con todo su formato (se diferencia en esto de el tipo **Text**) hasta imágenes, archivos de sonido o video, etc. Al igual que el tipo **Text**, **Blob** admite hasta 65.535 caracteres.
TinyBlob, MediumBlob, LongBlob	Son datos del mismo tipo que el anterior pero que varían en cuanto a su tamaño, así **TinyBlob** admite hasta 255 caracteres máximo, **MediumBlob** acepta tamaños de hasta 16.777.215 de caracteres y **LongBlob** 4.294.967.295 caracteres (como vemos estos tamaños se corresponden con los de **TinyText**, **MediumText** y **LongText**).

Tipo numérico (TinyInt, SmallInt, MediumInt, Int, BigInt, Float, Double, Decimal)

Int	Este es un tipo de datos numéricos de tipo entero. Este tipo de datos guarda valores enteros (no decimales) entre -2.147.483.648 y 2.147.483.647.
TinyiInt, SmallInt, MediumInt, BigInt	Son tipos de datos numéricos enteros (no decimal). **TinyiInt** agrupa un rango de números entre -128 y 127. **SmallInt** alcanza desde -32.768 hasta 32.767. **MediumInt** tiene un rango comprendido entre -8.388.608 y 8.388.607. Finalmente el tipo de datos **BigInt** ocupa un rango numérico entre -9.223.372.036.854.775.808 hasta 9.223.372.036.854.775.807.
Float (M,D)	Número de coma flotante de precisión simple. El valor del argumento M nos indica el número de digitos decimales que se van a utilizar para representar el número. Así, un valor de 5 nos permitirá representar números comprendidos entre -99 y 99 (Numeros expresados en binario con 5 dígitos y signo). El valor del argumento D nos indica el número de posiciones decimales que se van a utilizar en la representación del número. Así, una representación tipo **Float (5,2)** nos permitirá incluir números entre -99,99 y 99,99. El rango de los números de coma flotante de precisión simple es de -3,402823466E+38 a -1,175494351E-38, 0, y 1,175494351E-38 hasta 3,402823466E+38.
Double (M,D)	Número de coma flotante de precisión doble. Es un tipo de datos igual al anterior cuya única diferencia es el rango numérico que abraca, siendo este el comprendido entre 1,7976931348623157E+308 hasta -2,.2250738585072014E-308, 0, y 2,2250738585072014E-308 to 1,7976931348623157E+308
Decimal (M,D)	Su uso es similar al de los anteriores, pero, en este caso, D puede tener valor 0. El rango de este número es el mismo que el de número con coma flotante de precisión doble.

Tipo Fecha-Hora (Date, DateTime, TimeStamp, Time, Year)

Date	Formato de Fecha. Su representación es en formato de fecha numérica del tipo '**YYYY-MM-DD**' (Año con cuatro dígitos, Mes con dos dígitos, día con dos dígitos). Su rango es '1000-01-01' (1 de enero del año 1000, en el cual yo era aún muy pequeño) hasta '9999-12-31' (31 de diciembre del 9999, que ya veremos que pasa después de las uvas)
DateTime	Es una combinación de formato de fecha y hora conjuntamente. Su representación es '**YYYY-MM-DD HH:MM:SS**' (Año con cuatro dígitos, Mes con dos dígitos, día con dos dígitos, hora con dos dígitos, minutos con dos dígitos, segundos con dos dígitos). El rango que soporta este formato es de '1000-01-01 00:00:00' (las 00 horas, 00 minutos, 00 segundos del 1 de enero del año 1000, que no se yo con que reloj podían medir esto) hasta '9999-12-31 23:59:59' (las 23 horas, 59 minutos, 59 segundos del 31 de diciembre del año 9999, es decir, justo antes de las campanadas y una vez que han acabado los cuartos).
TimeStamp(N)	Este es un tipo de datos muy particular. Necesita de un argumento N que puede ser uno de estos números; 14, 12, 10, 8, 6, 4, 2. N representa el número de dígitos que se utilizarán para repsentar un valor de fecha y hora comprendido desde el inicio del año 1970 hasta algún momento del año 2037. Así: TimeStamp(14): YYYYMMDDHHMMSS (Año 4 dígitos + mes + día + hora + minutos + segundos 2 dígitos) TimeStamp(12): YYMMDDHHMMSS (Año 2 dígitos + mes + día + hora + minutos + segundos 2 dígitos) TimeStamp(10): YYMMDDHHMM (Año + mes + día + hora + minutos 2 dígitos) TimeStamp(8): YYMMDDHH (Año + mes + día + hora 2 dígitos) TimeStamp(6): YYMMDD (Año + mes + día 2 dígitos) TimeStamp(4): YYMM (Año + mes 2 dígitos) TimeStamp(2): YY (Año 2 dígitos)
Time	Tipo de datos con formato de Hora. mySQL muestra valores de hora con

	formato '**HH:MM:SS**'
Year(D)	Tipo de datos con formato de año. Su representación puede ser '**YYYY**' (año con formato de 4 dígitos) o '**YY**' (año con formato de 2 dígitos) donde el valor del argumento D puede ser 4 o 2 respectivamente.

Este ha sido un breve repaso a los tipos de datos que podemos utilizar a la hora de crear campos de datos en tablas de bases de datos mySQL. Recomiendo (yo y todos) analizar profundamente el tipo d e datos que se van a utlizar en cada campo ya que de esta manera podemos mejorar el rendimiento de nuestra base de datos en un porcentaje muy grande.

2.4 Definición y Creación de Índices. Claves Primarias y Externas

Los índices son objetos que forman parte del esquema que hacen que las bases de datos aceleren las operaciones de consulta y ordenación sobre los campos a los que el índice hace referencia.

Lo que realizan es una lista ordenada por la que el SGBD puede acceder para facilitar la búsqueda de los datos. Cada vez que se añade un nuevo registro, los índices involucrados se actualizan a fin de que su información esté al día. De ahí que cuantos más índices haya, más le cuesta al SGBD añadir registros, pero más rápidas se realizan las instrucciones de consulta.

La mayoría de los índices se crean de manera implícita, como consecuencia de las restricciones PRIMARY KEY, UNIQUE y FOREIGN KEY. Estos son índices obligatorios, por los que los crea el propio SGBD.

Creación de Índices:

Aparte de los índices obligatorios comentados anteriormente, se pueden crear índices de forma explícita. Éstos se crean para aquellos campos sobre los cuales se realizarán búsquedas e instrucciones de ordenación frecuente.

Sintaxis:

CREATE INDEX nombre
ON tabla (columna1 [,columna2...])

Ejemplo:

CREATE INDEX nombre_completo
ON clientes (apellido1, apellido2, nombre);

El ejemplo crea un índice para los campos apellido1, apellido2 y nombre. Esto no es lo mismo que crear un índice para cada campo, este índice es efectivo cuando se buscan u ordenan clientes usando los tres campos (apellido1, apellido2, nombre) a la vez.

Se aconseja crear índices en campos que:

- Contengan una gran cantidad de valores
- Contengan una gran cantidad de nulos
- Sean parte habitual de cláusulas WHERE, GROUP BY u ORDER BY

No se aconseja en campos que:

- Pertenezcan a tablas pequeñas
- No se usen a menudo en las consultas
- Pertenecen a tablas que se actualizan frecuentemente

Borrado de Índices

La instrucción **DROP INDEX** seguida del nombre del índice permite eliminar el índice en cuestión.

Para ver la lista de índices en Oracle se utiliza la vista USER_INDEXES. Mientras que la vista USER_IND_COLUMNS Muestra la lista de columnas que son utilizadas por índices.

Claves Primarias y Externas

En el apartado 2.2.1 del presente tema se vio cómo deben de crear las restricciones para la creación de claves primarias y externas (ajenas).