PARTE I
LENGUAJE SQL. GESTION DE DATOS

TEMA 1. EL LENGUAJE DE GESTIÓN DE BASES DE DATOS

Paulina Barthelemy

1 – Bases de datos

1.1 - Lenguaje de gestión de bases de datos.

SQL son las siglas de Structured Query Language que significa lenguaje estructurado de consulta

Se trata de un lenguaje definido por el estándar ISO/ANSI_SQL que utilizan para la gestión de las bases de datos los principales fabricantes de Sistemas de Gestión de Bases de Datos Relacionales.

Es un lenguaje estándar no procedimental que se utiliza para definir, gestionar y manipular la información contenida en una Base de Datos Relacional.

En los lenguajes procedimentales se deben especificar todos los pasos que hay que dar para conseguir el resultado. Sin embargo, como ya hemos dicho, SQL es un lenguaje no procedimental en el que tan solo deberemos indicar al sistema qué es lo que queremos obtener, y el sistema decidirá cómo obtenerlo.

Es un lenguaje sencillo y potente que se emplea para la gestión de la base de datos a distintos niveles de utilización: usuarios, programadores y administradores de la base de datos.

1.2 - ¿Qué es una Base de Datos?

Una base de datos está constituida por un conjunto de información relevante para una empresa o entidad, junto con los procedimientos para almacenar, controlar, gestionar y administrar esa información.

Además, la información contenida en una base de datos cumple una serie de requisitos o características:

- Los datos están interrelacionados, sin redundancias innecesarias.
- Los datos son independientes de los programas que los usan.
- Se emplean métodos determinados para recuperar los datos almacenados o para incluir datos nuevos y borrar o modificar los existentes

Una base de datos estará organizada de forma que se cumplan los requisitos para que la información se almacene con las mínimas redundancias, con capacidad de acceso para diferentes usuarios pero con un control de seguridad y privacidad. Debe tener mecanismos que permitan recuperar la información en caso de pérdida y la capacidad de adaptarse fácilmente a nuevas necesidades de

almacenamiento.

1.3 - ¿Qué es un Sistema de Gestión de Bases de Datos?

Un Sistema de Gestión de Bases de Datos (SGBD) es una aplicación formada por un conjunto de programas que permite construir y gestionar bases de datos. Proporciona al usuario de la base de datos las herramientas necesarias para realizar, al menos, las siguientes tareas:

- Definir las estructuras de los datos.
- Manipular los datos. Es decir, insertar nuevos datos, así como modificar, borrar y consultar los datos existentes.
- Mantener la integridad de la información.
- Proporcionar control de la privacidad y seguridad de los datos en la Base de Datos, permitiendo sólo el acceso a los mismos a los usuarios autorizados.

Para realizar las funciones que acabamos de describir, el Sistema Gestor de Bases de Datos necesita un conjunto de programas que gestionen el almacenamiento y la recuperación de dichos datos y un personal informático que maneje dichos programas.

Los componentes principales de un SGBD son:

- **GESTOR DE LA BASE DE DATOS**
  Es un conjunto de programas transparentes al usuario que se encargan de gestionar la seguridad de los datos y el acceso a ellos. Interacciona con el sistema operativo proporcionando una interfaz entre el usuario y los datos. Cualquier operación que se realice ha de estar procesada por este gestor.

- **DICCIONARIO DE LA BASE DE DATOS**
  Es donde se almacena toda la descripción de los diferentes objetos de la base de datos. Esta información se almacena con la misma estructura que los datos de los usuarios. El almacenamiento de esta información lo realiza el sistema gestor y cualquier usuario puede acceder a su contenido con el mismo lenguaje que al resto de los datos almacenados (SQL)

- **LENGUAJES**
  El sistema gestor ha de proporcionar lenguajes que permitan definir la estructura de los datos, almacenar la información y recuperarla. Podrán utilizar estos lenguajes los usuarios y lo administradores de la base de datos.
  Estos lenguajes son:
    - **Lenguaje de definición de datos (DDL)**
      Para definir la estructura con la que almacenaremos los datos.
    - **Lenguaje de manipulación de datos (DML)**
      Para añadir, modificar o eliminar datos, así como recuperar la información almacenada.
    - **Lenguaje de control de datos (DCL)**
      Para controlar el acceso a la información y la seguridad de los datos. Permiten limitar y controlar los accesos a la información almacenada. De esta tarea se ocupa el administrador de la base de datos.

- **ADMINISTRADOR DE LA BASE DE DATOS**
  Es una persona o grupo de personas responsables de la seguridad y la eficiencia de todos los componentes del sistema de bases de datos. Deben conocer el sistema tanto a nivel físico como lógico y a todos los usuarios que interaccionan con él.

- **USUARIOS DE LA BASE DE DATOS**
  Tradicionalmente considerados como un componente más de los sistemas gestores de bases de datos, debido a que estos fueron los primeros en considerarlos una parte importante para el correcto funcionamiento del sistema. Pueden ser usuarios terminales (usuarios no especializados que interaccionan con la base de datos), usuarios técnicos (usuarios que desarrollan programas de aplicación para ser utilizados por otros) y usuarios especializados (usuarios que utilizan el sistema gestor de la base de datos como una herramienta de desarrollo dentro de otros sistemas más complejos).

Algunos de los productos comerciales más difundidos son:

- **ORACLE** de Oracle Corporation.
- **DB2** de I.B.M. Corporation
- **Informix** de Informix Software Inc.
- **SQL Server** de Microsoft Corporation.
- **MySQL** producto Open Source (código abierto)

2 – Modelos de datos y bases de datos relacionales

Un modelo de datos es una filosofía de trabajo que permite realizar una abstracción de la realidad y representarla en el mundo de los datos.

2.1 – Tipos de bases de datos

Existen, tradicionalmente, varios tipos de bases de datos:

Bases de Datos Jerárquicas
Bases de Datos en Red
Bases de Datos Relacionales
Bases de Datos Objeto-Relacionales

Estas dos últimas son, con diferencia, las más difundidas y utilizadas en la actualidad debido a su potencia, versatilidad y facilidad de utilización. Se basan en el Modelo Relacional cuyas principales características veremos a continuación. Para gestionarlas se utiliza el lenguaje SQL.

2.2 - El Modelo de Datos Relacional. Componentes.

Un modelo de datos es un conjunto de reglas y convenciones que nos permiten describir, con los elementos del modelo, los datos y las relaciones entre ellos.

Analizaremos el modelo de datos relacional, en él se basan las bases de datos relacionales.
Sus principales componentes son:

Entidad.
Es un objeto acerca del cual se recoge información relevante.
Ejemplo de entidades: EMPLEADO, CLIENTE, PRODUCTO.

Atributo
Es una propiedad o característica de la entidad.
Por ejemplo pueden ser atributos de la entidad PERSONA los siguientes: DNI, NOMBRE, EDAD, etc.

Relación o Inter-Relación
Representa la relación que puede haber entre dos entidades.

Por ejemplo, una relación **compra** representa la relación entre las entidades CLIENTE y PRODUCTO

CLIENTE -> **compra** -> PRODUCTO..... "Un cliente **compra** un producto"

Y también, una relación **pertenece a** representa la relación entre las entidades EMPLEADO Y DEPARTAMENTO

EMPLEADO -> **pertenece a** -> DEPARTAMENTO....."Un empleado **pertenece** a un departamento"

Con este modelo podemos representar, utilizando los componentes anteriores, la información que deseamos almacenar en la base de datos. Posteriormente este modelo se va a convertir en una base de datos relacional.

2.3 – Bases de datos relacionales

La definición de Bases de Datos Relacionales fue enunciada por *E.F. Codd.* En 1972 estableciendo las reglas que debía cumplir cualquier base de datos para ser una base de datos relacional. Son 12 reglas, llamadas reglas de Codd, más una teoría matemática, llamada cálculo y álgebra relacional, que marcan las características de los sistemas relacionales.

Es un modelo basado en la teoría de las relaciones, álgebra y calculo relacional, con las siguientes características:

- Hay una parte de definición de datos, llamada estática, que nos da la estructura del modelo, donde los datos se encuentran almacenados en forma de relaciones, llamadas generalmente tablas, ya que su estructura es muy similar a las tablas convencionales. Estas tablas son independientes de la forma física de almacenamiento. A estos datos se añaden unas restricciones que son unas reglas que limitan los valores que podemos almacenar en las tablas de la base de datos y nos permiten implementar las relaciones entre las tablas.
- A esta parte se añade otra, llamada dinámica, con las operaciones que se pueden realizar sobre las tablas, anteriormente definidas, para gestionar los datos almacenados

El modelo de datos relacional que acabamos de exponer se almacena en una base de datos relacional. Al realizar la conversión de un modelo relacional a una base de datos relacional las entidades y las relaciones del modelo se transforman en tablas y restricciones de la base de datos. Estas tablas junto con las restricciones forman la clave de las bases de datos relacionales.

2.3.1 - Tablas

Son los objetos de la Base de Datos donde se almacenan los datos. Tienen la forma de una tabla tradicional, de ahí su nombre. Normalmente una tabla representa una entidad aunque también puede representar una asociación de entidades. Cada fila representa una ocurrencia de la entidad y cada columna representa un atributo o característica de la entidad.
Las tablas tienen un nombre que las identifica y están formadas por atributos representados en las columnas, y por tuplas representadas en las filas.

Ejemplos de atributos: para la tabla `departamentos` las columnas con el número de departamento, el nombre del departamento y la localidad donde se encuentra.
La tabla `empleados` puede tener como columnas o atributos: numero de empleado, nombre, fecha de alta, salario,...

Ejemplos de tuplas son: los datos de un empleado si es una tabla de `empleados`, de un departamento si es una tabla de `departamentos`, de un cliente si se trata de una tabla de `clientes`, o de un producto si es una tabla de `productos`.

Ejemplos de tablas:
Tabla ***DEPARTAMENTOS:***

	Columna 1	*Columna 2*	*Columna 3*
	DEP_NO	**DNOMBRE**	**LOCALIDAD**
Fila 1 ->	10	CONTABILIDAD	BARCELONA
Fila 2 ->	20	INVESTIGACION	VALENCIA
Fila 3 ->	30	VENTAS	MADRID
Fila 4 ->	40	PRODUCCION	SEVILLA

Tabla de ***EMPLEADOS:***

EMP_NO	APELLIDO	OFICIO	DIRECTOR	FECHA_ALTA	SALARIO	COMISION	DEP_NO
7499	ALONSO	VENDEDOR	7698	20/02/81	1400.00	400.00	30
7521	LOPEZ	EMPLEADO	7782	08/05/81	1350.00	NUL0	10
7654	MARTIN	VENDEDOR	7698	28/09/81	1500.00	1600.00	30
7698	GARRIDO	DIRECTOR	7839	01/05/81	3850.00	NULO	30
7782	MARTINEZ	DIRECTOR	7839	09/06/81	2450.00	NUL0	10
7839	REY	PRESIDENTE	NULO	17/11/81	6000.00	NULO	10
7844	CALVO	VENDEDOR	7698	08/09/81	1800.00	0.00	30
7876	GIL	ANALISTA	7782	06/05/82	3350.00	NULO	20
7900	JIMENEZ	EMPLEADO	7782	24/03/82	1400.00	NULO	20

2.3.2 - Tablas del curso

A lo largo de este curso utilizaremos, además de las tablas `EMPLEADOS` y `DEPARTAMENTOS` cuya forma y contenido ya hemos visto, las tablas de `CLIENTES`, `PRODUCTOS` y `PEDIDOS` cuya forma y contenido es el siguiente:

TABLA DE CLIENTES

```
mysql> SELECT *
    -> FROM clientes;
+----------+-----------------------+-----------+-----------+-----+------+---------------+
|CLIENTE_NO| NOMBRE                | LOCALIDAD |VENDEDOR_NO|DEBE | HABER| LIMITE_CREDITO|
+----------+-----------------------+-----------+-----------+-----+------+---------------+
|      101 | DISTRIBUCIONES GOMEZ  | MADRID    |      7499 |0.00 |  0.00|       5000.00 |
|      102 | LOGITRONICA S.L       | BARCELONA |      7654 |0.00 |  0.00|       5000.00 |
|      103 | INDUSTRIAS LACTEAS S.A.| LAS ROZAS |      7844 |0.00 |  0.00|      10000.00 |
```

```
|      104 | TALLERES ESTESO S.A.    | SEVILLA   |      7654 |0.00 |  0.00|       5000.00 |
|      105 | EDICIONES SANZ          | BARCELONA |      7499 |0.00 |  0.00|       5000.00 |
|      106 | SIGNOLOGIC S.A.         | MADRID    |      7654 |0.00 |  0.00|       5000.00 |
|      107 | MARTIN Y ASOCIADOS S.L.| ARAVACA   |      7844 |0.00 |  0.00|      10000.00 |
|      108 | MANUFACTURAS ALI S.A.   | SEVILLA   |      7654 |0.00 |  0.00|       5000.00 |
+----------+-------------------------+-----------+-----------+-----+------+---------------
8 rows in set (0.00 sec)
```

TABLA PRODUCTOS

```
mysql> SELECT *
    -> FROM productos;
+-------------+-----------------------------+---------------+------------------+
| PRODUCTO_NO | DESCRIPCION                 | PRECIO_ACTUAL | STOCK_DISPONIBLE |
+-------------+-----------------------------+---------------+------------------+
|          10 | MESA DESPACHO MOD. GAVIOTA  |        550.00 |               50 |
|          20 | SILLA DIRECTOR MOD. BUFALO  |        670.00 |               25 |
|          30 | ARMARIO NOGAL DOS PUERTAS   |        460.00 |               20 |
|          40 | MESA MODELO UNIÓN           |        340.00 |               15 |
|          50 | ARCHIVADOR CEREZO           |       1050.00 |               20 |
|          60 | CAJA SEGURIDAD MOD B222     |        252.00 |               15 |
|          70 | DESTRUCTORA DE PAPEL A3     |        450.00 |               25 |
|          80 | MODULO ORDENADOR MOD. ERGOS |        495.00 |               25 |
+-------------+-----------------------------+---------------+------------------+
8 rows in set (0.03 sec)
```

TABLA PEDIDOS

```
mysql> SELECT *
    -> FROM pedidos;
+-----------+-------------+------------+----------+--------------+
| PEDIDO_NO | PRODUCTO_NO | CLIENTE_NO | UNIDADES | FECHA_PEDIDO |
+-----------+-------------+------------+----------+--------------+
|      1000 |          20 |        103 |        3 | 1999-10-06   |
|      1001 |          50 |        106 |        2 | 1999-10-06   |
|      1002 |          10 |        101 |        4 | 1999-10-07   |
|      1003 |          20 |        105 |        4 | 1999-10-16   |
|      1004 |          40 |        106 |        8 | 1999-10-20   |
|      1005 |          30 |        105 |        2 | 1999-10-20   |
|      1006 |          70 |        103 |        3 | 1999-11-03   |
|      1007 |          50 |        101 |        2 | 1999-11-06   |
|      1008 |          10 |        106 |        6 | 1999-11-16   |
|      1009 |          20 |        105 |        2 | 1999-11-26   |
|      1010 |          40 |        102 |        3 | 1999-12-08   |
|      1011 |          30 |        106 |        2 | 1999-12-15   |
|      1012 |          10 |        105 |        3 | 1999-12-06   |
|      1013 |          30 |        106 |        2 | 1999-12-06   |
|      1014 |          20 |        101 |        4 | 2000-01-07   |
|      1015 |          70 |        105 |        4 | 2000-01-16   |
+-----------+-------------+------------+----------+--------------+
16 rows in set (0.02 sec)
```

2.3.3 – Restricciones

Son una parte importante para la implementación del modelo relacional. Restringen los valores que pueden tomar los datos en cada una de las columnas.

Estas restricciones son:

- **Clave primaria (PRIMARY KEY)**
  Una de las características del modelo relacional es que cada fila debe ser única. Ello obliga a la existencia de un identificativo que permita y controle esta unicidad. Este identificativo es la clave primaria. Estará formada por una columna o grupo de columnas y se elegirá de tal forma que su valor en cada fila sea único.
  En una base de datos relacional es obligatoria su existencia en cada una de las tablas.

- **Clave Ajena (FOREIGN KEY)**
  Será la forma de implementar las relaciones entre las tablas. Una columna o grupo de columnas que sea clave ajena referenciará a la tabla con la que está relacionada. Los valores que podrá tomar la clave ajena serán valores que ya existan en la tabla relacionada, o en su defecto un valor nulo.
  La importancia de su definición radica en que será la encargada de implementar las relaciones entre las tablas, para ajustarnos al Modelo Relacional
  Por ejemplo la tabla EMPLEADOS está relacionada con la tabla DEPARTAMENTOS a través de la columna DEP_NO (numero de departamento) que se encuentra en ambas tablas. Esta columna es clave primaria de la tabla DEPARTAMENTOS y en la tabla EMPLEADOS es la clave ajena. Esto quiere decir que los valores que tome el campo DEP_NO en la tabla EMPLEADOS solo pueden ser valores que ya existan en el campo DEP_NO de la tabla DEPARTAMENTOS.

- **Unicidad (UNIQUE)**
  Es una restricción que obliga a que una columna o conjunto de columnas tenga un valor único o un valor nulo. También admite valores nulos.

- **Restricción de valores permitidos (CHECK)**
  Es una restricción que nos permitirá controlar el conjunto de valores que serán válidos en una columna. Solo serán validos los valores que cumplan la condición especificada.
  Nota: MySQL lo acepta dentro del formato pero no lo implementa en la versión actual.

- **Obligación de valor (NOT NULL)**
  Una de las características de las bases de datos relacionales es que se permite la ausencia de valor en los datos. Esta ausencia puede ser debida a que en el momento de introducir los datos se desconoce el valor de un atributo para esa fila o a que ese atributo es inaplicable para esa fila. El valor que se le asigna a ese atributo en esa fila se llama valor nulo (`NULL`). Esta restricción obliga a que una columna tenga que tener siempre valor no permitiéndose que tome el valor nulo.
  En el siguiente capítulo hablaremos de forma más extensa de estos valores nulos.

2.3.4 - Restricciones en las tablas del curso

Para implementar el modelo relacional con las restricciones propias (cada fila debe ser única) y las restricciones de nuestro modelo necesitamos, al menos, las siguientes restricciones en las tablas:

TABLA DEPARTAMENTOS

dep_no	**CLAVE PRIMARIA**

TABLA EMPLEADOS

emp_no	**CLAVE PRIMARIA**
dep_no	**CLAVE AJENA QUE REFERENCIA A dep_no DE DEPARTAMENTOS**
dir	**CLAVE AJENA QUE REFERENCIA A emp_no DE EMPLEADOS**

La tabla `EMPLEADOS` está relacionada con la tabla `DEPARTAMENTOS` a través de la columna DEP_NO (numero de departamento) que se encuentra en ambas tablas. De esta forma podemos saber, por ejemplo que el empleado GIL pertenece al departamento 20. Y si vamos a la tabla departamentos comprobaremos que el departamento 20 es INVESTIGACION y se encuentra en VALENCIA. Por tanto, el empleado GIL pertenece al departamento de INVESTIGACION que está en VALENCIA.
La tabla `EMPLEADOS` también se relaciona consigo misma mediante las columnas EMP_NO y DIRECTOR. Cada empleado tiene un número de empleado (EMP_NO) y suele tener también un DIRECTOR. Esta última columna contiene un número de empleado que, suponemos, es el director del empleado en cuestión. Así podemos saber que REY es el director de GARRIDO y de MARTINEZ; y que el director de JIMENEZ es MARTINEZ, etcétera. El director de un empleado debe ser a su vez empleado de la empresa, de ahí la existencia de esta clave ajena. La columna DIRECTOR deberá contener un valor que se corresponda con un valor de EMP_NO o tener el valor nulo.

TABLA CLIENTES

cliente_no	**CLAVE PRIMARIA**
vendedor_no	**CLAVE AJENA QUE REFERENCIA emp_no DE EMPLEADOS**

La tabla `CLIENTES` se relaciona con `EMPLEADOS` por medio de la columna VENDEDOR_NO de la primera que hace referencia a la columna EMPLEADO_NO de la segunda. Así cada cliente tendrá asignado un vendedor, que será un empleado de la empresa existente en la tabla `EMPLEADOS`.

TABLA PRODUCTOS

producto_no	**CLAVE PRIMARIA**

TABLA PEDIDOS

pedido_no	**CLAVE PRIMARIA**
producto_no	**CLAVE AJENA QUE REFERENCIA A producto_no DE PRODUCTOS**
cliente_no	**CLAVE AJENA QUE REFERENCIA A cliemte_no DE CLIENTES**

La tabla `PEDIDOS` se relaciona con `PRODUCTOS` mediante la columna PRODUCTO_NO y con CLIENTES mediante la columna CLIENTE_NO. De esta forma sabemos que el pedido número 1000 lo ha realizado el cliente INDUSTRIAS LACTEAS S.A. y que el producto solicitado es SILLA DIRECTOR MOD. BUFALO a un precio de 670.00, etcétera.

El SGBD velará porque todas las operaciones que se realicen respeten estas restricciones manteniendo así la integridad de la información (`integridad referencial`)

El resto de las restricciones las estudiaremos en el tema 3 de creación de tablas.

3 – Lenguaje SQL

Como ya hemos dicho al inicio del tema, SQL significa lenguaje estructurado de consulta *(Structured Query Language).*
Es un lenguaje desarrollado sobre un prototipo de gestor de bases de datos relacionales con su primera implementación en el año 1979.
Posteriormente, en 1986, fue adoptado como estándar por el instituto ANSI (*American National Standard Institute*) como estándar y en 1987 lo adopta ISO (*Internacional Standardization Organization).* Aparece así el ISO/ANSI SQL que utilizan los principales fabricantes de Sistemas de Gestión de Bases de Datos Relacionales.
El lenguaje SQL es un lenguaje relacional que opera sobre relaciones (tablas) y da como resultado otra relación.

3.1 - ¿Qué podemos hacer con SQL?

Todos los principales SGBDR incorporan un motor SQL en el Servidor de Base Datos, así como herramientas de cliente que permiten enviar comandos SQL para que sean procesadas por el motor del servidor. De esta forma, todas las tareas de gestión de la Base de Datos (BD) pueden realizarse utilizando sentencias SQL.
Lo que podemos hacer con este lenguaje SQL es:

- Consultar datos de la Base de Datos.
- Insertar, modificar y borrar datos.
- Crear, modificar y borrar objetos de la Base de Datos.
- Controlar el acceso a la información.
- Garantizar la consistencia de los datos.

Actualmente se ha impuesto el almacenamiento de la información en bases de datos. El SQL es, por tanto, un lenguaje muy extendido y muchos lenguajes de programación incorporan sentencias SQL como parte de su repertorio o permiten la comunicación con los motores de SQL.

3.2 - Tipos de sentencias SQL.

Entre los trabajos que se pueden realizar en una base de datos podemos distinguir tres tipos: definición, manipulación y control de datos. Por ello se distinguen tres tipos de sentencias SQL:

- Sentencias de definición de datos. (Lenguaje de Definición de Datos **DDL**)
  Se utilizan para:
  Crear objetos de base de datos ---------- **SENTENCIA CREATE**
  Eliminar objetos de base de datos ---------- **SENTENCIA DROP**
  Modificar objetos de base de datos ---------- **SENTENCIA ALTER**

- Sentencias de manipulación de datos. (Lenguaje de Manipulación de Datos **DML**)
  Se utilizan para:
  Recuperar información ---------- **SENTENCIA SELECT**
  Actualizar la información:
  Añadir filas ---------- **SENTENCIA INSERT**
  Eliminar filas ---------- **SENTENCIA DELETE**
  Modificar filas ---------- **SENTENCIA UPDATE**

- Sentencias de control de datos. (Lenguaje de Control de datos DCL)

Se utilizan para:
Crear privilegios de acceso a los datos ---------- **SENTENCIA GRANT**
Quitar privilegios de acceso a los datos ---------- **SENTENCIA REVOKE**

3.3 - Sentencias SQL

Realizaremos algunas consideraciones sobre las notaciones y formatos utilizados.

a) Formatos de las instrucciones
Estos formatos están recuadrados. Se escriben utilizando una notación que recordamos a continuación:

- Las palabras reservadas de SQL aparecen en mayúsculas.

- Los nombres de objetos (tablas, columnas, etcétera) aparecen en el formato TipoTítulo (las iniciales de las palabras en mayúsculas)

- Las llaves **{ }** indican la elección obligatoria entre varios elementos.

- La barra vertical **|** separa los elementos en una elección.

- Los corchetes **[ ]** encierran un elemento opcional.

- El punto y coma **;** que aparece al final de cada comando es el separador de instrucciones y en realidad no forma parte de la sintaxis del lenguaje SQL, pero suele ser un elemento requerido por las herramientas de cliente para determinar el final del comando SQL y enviar la orden (sin él ;) al servidor.

Los formatos de las instrucciones, cuando sean complejos, se irán viendo por partes. Cada vez que añadamos algo nuevo lo remarcaremos en negrita.

b) Consulta de los datos.
Realizar una consulta en SQL consiste en recuperar u obtener aquellos datos que, almacenados en filas y columnas de una o varias tablas de una base de datos, cumplen unas determinadas especificaciones. Para realizar cualquier consulta se utiliza la sentencia SELECT.

Aunque la sentencia de consulta de datos en las tablas, SELECT, se tratará con profundidad en los temas del 5 al 9, necesitaremos hacer alguna pequeña consulta para poder verificar los datos que tenemos en las tablas. Las primeras consultas van a ser escritas con un formato inicial de la sentencia SELECT, que se completará en el bloque siguiente.

SELECT { * | [*NombreColumna* [, *NombreColumna*]....] }
FROM *NombreTabla*
[WHERE *Condicion] ;

*Notación: hay que escoger obligatoriamente una de las opciones, entre indicar los nombres de las columnas y el asterisco ***** (por eso aparecen las posibles opciones entre llaves y separadas por una barra). En caso de escoger la segunda opción se pueden indicar una o varias columnas (por eso aparece entre corchetes y seguido de puntos suspensivos). La cláusula WHERE es opcional (por eso aparece entre corchetes).*

El funcionamiento de esta sentencia es el siguiente: visualizará las correspondientes filas de la tabla. Si hemos escrito los nombres de las columnas separados por comas visualizará solo los valores de esas columnas y si hemos escrito el signo * visualizará todas las columnas. Si además, hemos escrito una condición con la cláusula `WHERE` visualizará solo las filas que cumplan esa condición.

c) Notación de los ejemplos

Otras directrices importantes de notación. Vamos a escribir cada cláusula de la instrucción en una línea, como los espacios en blanco y saltos de línea dentro de la instrucción son ignorados, hasta que encuentra el ; que es el fin de la instrucción. Ello facilita enormemente la lectura de las instrucciones de selección.

Vamos a ver un ejemplo aunque todavía no conozcamos el significado.

Supongamos que escribimos:

select apellido,(salario+ifnull(comisio,0)) “salario total”,
dept_no from empleados where oficio = ’analista’ order by dept_no;

O bien que escribimos:

```
SELECT apellido, salario+IFNULL(comision,0) “Salario total”, dept_no
FROM empleados
WHERE oficio = ’ANALISTA’
ORDER BY dept_no;
```

Este segundo formato es más claro y visual. Cada cláusula, comenzando con una palabra reservada, aparece en una línea y se han introducido algunos espacios en blancos y saltos de línea para separar.
Utilizaremos este formato en todos los ejemplos y ejercicios.

TEMA 2. ELEMENTOS DEL LENGUAJE

Paulina Barthelemy

1 - Introducción.

Antes de abordar esta unidad hay que tener en cuenta:

1º Se trata de una guía para que sirva de referencia o de consulta cuando se necesite a lo largo del curso. Posteriormente haremos ejemplos utilizando los elementos estudiados en este tema. Permitirá irse familiarizando con el vocabulario y la sintaxis de las sentencias que se explicarán posteriormente.

2º En esta unidad se abordan cuestiones que, aunque están definidas por el estándar ANSI/ISO SQL, no están asumidas al 100% por todos los fabricantes. Por tanto, pueden existir ligeras diferencias de algunos productos con algunas de las especificaciones que aquí se exponen.
Se ha escogido el gestor de bases de datos MySQL para realizar los ejemplos y los ejercicios. Aunque como este es un curso de SQL, no se profundizará en las particularidades de este SGBD hasta los temas de Administración (temas 10, 11 y 12 del curso) que si son específicos de MySQL.

2 – Elementos de SQL

2.1 – Identificadores

Es la forma de dar nombres a los objetos de la base de datos y a la propia base de datos. Un objeto tendrá un nombre (*NombreObjeto*) que lo identifique de forma única dentro de su base de datos.

El estándar define que pueden tener hasta 18 caracteres empezando con un carácter alfabético y continuando con caracteres numéricos y/o alfabéticos.

En la práctica este estándar se ha ampliado y MySQL permite nombres de identificadores de hasta 64 caracteres sin espacios en blanco. Para los nombres de las bases de datos y de las tablas no están permitidos '/', '\' ni '.'

2.2 – Palabras reservadas

Al igual que en el resto de los lenguajes de programación, existen palabras que tiene un significado especial para el gestor de la base de datos y no pueden ser utilizadas como identificadores.
Serán todas las palabras que irán apareciendo en los formatos de las instrucciones, más algunas que se verán en este curso.
Puede encontrarse una lista completa en el manual de MySQL en la dirección: http://dev.mysql.com en el apartado Reserved_words.html

3 - Datos

3.1 – Datos constantes o literales

En SQL podemos utilizar los siguientes tipos de constantes:

Constantes numéricas.
Construidas mediante una cadena de dígitos que puede llevar un punto decimal, y que pueden ir precedidos por un signo + ó -. (Ej. : -2454.67)
También se pueden expresar constantes numéricas empleado el formato de coma flotante, notación científica. (Ej. : 34.345E-8).

Constantes de cadena.
Consisten en una cadena de caracteres encerrada entre comillas simples. (Ej.: 'Hola Mundo')

Constantes de fecha.
En realidad las constantes de fecha, en MySQL y otros productos que soportan este tipo, se escriben como constantes de cadena sobre las cuales se aplicarán las correspondientes funciones de conversión.
Existe una gran cantidad de formatos aplicables a estas constantes (americano, europeo, japonés, etcétera). La mayoría de los productos pueden trabajar también con FECHA Y HORA en distintos formatos.

3.2 – Datos variables

Las columnas de la base de datos almacenan valores que tendrán diferentes valores en cada fila. Estos datos se definen indicando su nombre (*NombreColumna*) y el tipo de datos que almacenarán. La forma de almacenarlos no es la misma para todos, por lo tanto una parte importante de la definición de un dato es la especificación de su tipo. Al indicar el tipo de datos se suele indicar también el tamaño.

La cantidad de tipos de datos posibles es muy extensa, quedando la enumeración completa fuera de los objetivos de este curso. A continuación se indican algunos de los tipos de datos más utilizados suficientes para este curso (para mayor detalle consultar el manual).

a) Datos numéricos

INT[(*num*)] o INTEGER [(*num*)]
Se utiliza para guardar datos numéricos enteros, siendo *num* el número de dígitos

FLOAT(*escala, precision)*
Se utiliza para guardar datos numéricos en coma flotante. La escala indica el número total de dígitos. La precisión el número de posiciones decimales

NUMERIC(*escala, precisión)*
Se utiliza para guardar datos numéricos. La escala indica el número total de dígitos. La precisión el número de posiciones decimales, y si no se especifica se supone 0 (correspondería a un número entero)

b) Datos alfanuméricos o cadenas de caracteres

CHAR *(long)*
Se utiliza para guardar cadenas de caracteres de longitud fija especificada entre paréntesis. La longitud, *long,* puede ser un número entre 0 y 255 (*ver nota)*

VARCHAR *(long)*
Se utiliza igualmente para almacena cadenas de caracteres de longitud variable cuyo límite máximo es el valor especificado como *long* (*ver nota)*
La longitud puede ser un número entre 0 y 255

TEXT
Un texto de longitud máxima 65.535 caracteres. (2 ^ 16 – 1).
Se almacena como un VARCHAR

LONGTEXT
Un texto de longitud máxima 4 Gigas caracteres. (2 ^ 32 – 1)
Se almacena como un VARCHAR

NOTA:

La diferencia entre ambos tipos, CHAR y VARCHAR, está en la forma de almacenarlos. Teniendo en cuenta que un carácter necesita un byte para su almacenamiento, un dato definido como CHAR(5) se almacena reservando espacio para los 5 caracteres, aunque el datos almacenado tenga menos de 5. Sin embargo, un dato definido como VARCHAR(5) en el que se almacene un datos solo reserva espacio para el número de caracteres que tenga ese dato más uno para indicar el final de la cadena.

VALOR	***CHAR(5)***	***TAMAÑO RESERVADO***	***VARCHAR(5)***	***TAMAÑO RESERVADO***
‘ ‘	*‘ ‘*	*5 BYTES*	*‘ ‘*	*1 BYTE*
‘AB’	*‘AB ‘*	*5 BYTES*	*‘AB’*	*3 BYTES*
‘ABCDE’	*‘ABCDE’*	*5 BYTES*	*‘ABCDE’*	*6 BYTES*

c) Fechas
Estos datos se tratan externamente como cadenas de caracteres por lo que estar entre comillas para su utilización.

DATE
Este tipo de dato permite almacenar fechas, incluyendo en esa información: año, mes y día con la forma ‘YYYY-MM-DD’.

DATETIME
Este tipo de dato permite almacenar fechas y horas, incluyendo en esa información: año, mes, día, horas, minutos y segundos con la forma ‘YYYY-MM-DD HH:MM:SS’.

TIME
Este tipo de dato permite almacenar horas, incluyendo en esa información: horas, minutos y segundos con la forma 'HH:MM:SS'.

d) Binarios

BOOLEAN
Almacena valores binarios formados por combinaciones de los valores 1(verdadero) y 0 (falso).

Nota: hemos escogido los tipos datos más utilizados con la notación de MySQL que se ajustan a las especificaciones del estándar ANSI/ISO SQL estándar. Pero si se utiliza otro sistema gestor debe tenerse en cuenta que puede haber variaciones a la hora de indicar los tipos de datos.

4 - Operadores

4.1 - Operadores aritméticos

Operan entre valores numéricos y devuelven un valor numérico como resultado de realizar los cálculos indicados, algunos de ellos se pueden utilizar también con fechas. Se emplean para realizar cálculos numéricos.
Los operadores aritméticos son:

+	Suma
-	Resta
*	Multiplicación
/	División
Div	División entera (parte entera de la división, sin decimales)

4.2- Operadores de comparación

Las expresiones formadas con operadores de comparación dan como resultado un valor de tipo *Verdadero/Falso/Nulo (True/False/Null).*
Los operadores de comparación son:

=	Igual
!=	Distinto
<>	Distinto
<	Menor
<=	Menor o igual
>	Mayor
>=	Mayor o igual

BETWEEN / NOT BETWEEN
IN / NOT IN
IS NULL / IS NOT NULL
LIKE

Los primeros son los operadores relaciones ya conocidos. Los segundos son pares de un operador y su negación con la siguiente forma de actuar:

BETWEEN valor1 AND valor2
Da como resultado VERDADERO si el valor comparado es mayor o igual que valor1 y menor o igual que valor2 y FALSO en el caso contrario

IN (lista de valores separados por comas)
Da como resultado VERADADERO si el valor comparado está dentro de la lista de valores especificado y FALSO en el caso contrario

IS NULL
Da como resultado VERDADERO si el valor del dato comparado es nulo (`NULL`) y FALSO en el caso contrario

LIKE
Permite comparar dos cadenas de caracteres con la peculiaridad de que admite caracteres comodines. Los caracteres comodines son `'%'` y `'_'`. Estos caracteres permiten utilizar patrones en la comparación. El `%'` puede ser sustituido por un grupo de caracteres (de 0 a cualquier número) y el `'_'` por un carácter cualquiera en esa posición.

Ejemplos:

1. La expresión: APELLIDO = 'JIMENEZ' será verdadera *(*`true`*)* en el caso de que el valor de la columna APELLIDO (suponemos que se trata de una columna) sea 'JIMENEZ' y falsa *(*`false`*)* en caso contrario.

2. La expresión: SALARIO > 300000 será verdadera (`true`*)* en el caso de que SALARIO tenga un valor superior a 300000 y falsa (`false`*)* en caso contrario.

3. La expresión APELLIDO LIKE 'A%' será verdadera (`true`) si el apellido empieza por A y después tiene cualquier grupo de caracteres y falsa (`false`) en caso contrario. La expresión APELLIDO LIKE 'A_B_C' será verdadera (`true`) si el primer carácter es una A, el segundo cualquiera, el tercero una B, el cuarto cualquiera y el quinto una C y falsa (`false`) en caso contrario.
Estos operadores de comparación se utilizan fundamentalmente para construir condiciones de búsqueda en la base de datos. De esta forma se seleccionarán aquellas filas que cumplan la condición especificada (aquellas filas para las que el valor de la expresión sea *true).* Por ejemplo, el siguiente comando seleccionará todas las filas de la tabla empleados que en la columna OFICIO aparezca el valor 'VENDEDOR'.

```
mysql> SELECT emp_no, apellido,oficio,fecha_alta,comision,salario
    -> FROM empleados
    -> WHERE oficio = 'VENDEDOR';

+--------+---------+----------+------------+----------+---------+
| EMP_NO | APELLIDO| OFICIO   | FECHA_ALTA | COMISION | SALARIO |
+--------+---------+----------+------------+----------+---------+
|   7499 | ALONSO  | VENDEDOR | 1981-02-23 |   400.00 | 1400.00 |
|   7654 | MARTIN  | VENDEDOR | 1981-09-28 |  1600.00 | 1500.00 |
|   7844 | CALVO   | VENDEDOR | 1981-09-08 |     0.00 | 1800.00 |
+--------+---------+----------+------------+----------+---------+
3 rows in set (0.00 sec)
```

4.3 - Operadores lógicos

Operan entre datos con valores lógicos *Verdadero/Falso/Nulo* y devuelven el valor lógico *Verdadero/Falso/Nulo.* Los operadores lógicos son:

!	NOT
&&	AND
||	OR
	XOR

A continuación se detallan las tablas de valores de los operadores lógicos NOT, AND y OR, teniendo en cuenta todos los posibles valores, incluida la ausencia de valor (NULL).

NOT	VERDADERO	FALSO	NULO
	FALSO	VERDADERO	NULO

AND	VERDADERO	FALSO	NULO
VERDADERO	VERDADERO	FALSO	NULO
FALSO	FALSO	FALSO	FALSO
NULO	NULO	FALSO	NULO

OR	VERDADERO	FALSO	NULO
VERDADERO	VERDADERO	VERDADERO	VERDADERO
FALSO	VERDADERO	FALSO	NULO
NULO	VERDADERO	NULO	NULO

XOR	VERDADERO	FALSO	NULO
VERDADERO	FALSO	VERDADERO	NULO
FALSO	VERDADERO	FALSO	NULO
NULO	NULO	NULO	NULO

Podemos establecer:

- El operador NOT devuelve VERDADERO cuando el operando es falso, y FALSO cuando el operando es verdadero y NULO cuando el operando es nulo.
- El operador AND devolverá VERDADERO cuando los dos operandos sean verdaderos, FALSO cuando alguno de los dos operandos sea falso y NULO en los demás casos.
- El operador OR devolverá VERDADERO cuando alguno de los operandos sea verdadero, FALSO cuando los dos operandos sean falsos; y NULO en los demás casos.
- El operador XOR devolverá VERDADERO si uno de los operandos es verdadero y el otro falso, FALSO cuando ambos sean verdaderos o ambos falsos y NULO si alguno de ellos es nulo.

Ya hemos indicado que los operadores de comparación devuelven un valor de tipo Verdadero/Falso/Nulo (`True/False/Null`). En ocasiones se necesita trabajar con varias expresiones de comparación (por ejemplo cuando queremos formar una condición búsqueda que cumpla dos condiciones, etcétera) en estos casos debemos recurrir a los operadores lógicos `AND`, `OR`, `XOR` y `NOT`.

Supongamos que queremos consultar los empleados cuyo OFICIO = 'VENDEDOR' y que además su SALARIO > 1500. En este caso emplearemos el operador lógico `AND`. Este operador devolverá el valor *true* cuando los dos operandos o expresiones son verdaderas. Podemos decir que se utiliza el operador `AND` cuando queremos que se cumplan las dos condiciones.

Ejemplo:

```
mysql> SELECT apellido, salario, oficio
    -> FROM empleados
    -> WHERE oficio = 'VENDEDOR' AND salario > 1500;

+----------+---------+----------+
| APELLIDO | SALARIO | OFICIO   |
+----------+---------+----------+
| CALVO    | 1800.00 | VENDEDOR |
+----------+---------+----------+
1 row in set (0.00 sec)
```

Cuando lo que queremos es buscar filas que cumplan alguna de las condiciones que se indican emplearemos el operador OR. Este operador devolverá el valor VERDADERO cuando alguno de los dos operandos o expresiones es verdadero .Podemos decir que se utiliza el operador `OR` cuando queremos que se cumpla la primera condición, o la segunda o ambas.

Ejemplo:

```
mysql> SELECT apellido, salario, oficio
    -> FROM empleados
    -> WHERE oficio = 'VENDEDOR' OR salario > 1500;

+----------+---------+------------+
| APELLIDO | SALARIO | OFICIO     |
+----------+---------+------------+
| ALONSO   | 1400.00 | VENDEDOR   |
| MARTIN   | 1500.00 | VENDEDOR   |
| GARRIDO  | 3850.12 | DIRECTOR   |
| MARTINEZ | 2450.00 | DIRECTOR   |
| REY      | 6000.00 | PRESIDENTE |
| CALVO    | 1800.00 | VENDEDOR   |
| GIL      | 3350.00 | ANALISTA   |
+----------+---------+------------+
7 rows in set (0.00 sec)
```

El operador NOT se utiliza para cambiar el valor devuelto por una expresión lógica o de comparación, tal como se ilustra en el siguiente ejemplo:

```
mysql> SELECT apellido, salario, oficio
    -> FROM empleados
    -> WHERE NOT (oficio = 'VENDEDOR');

+----------+---------+------------+
| APELLIDO | SALARIO | OFICIO     |
+----------+---------+------------+
| LOPEZ    | 1350.50 | EMPLEADO   |
| GARRIDO  | 3850.12 | DIRECTOR   |
| MARTINEZ | 2450.00 | DIRECTOR   |
```

```
| REY      | 6000.00 | PRESIDENTE |
| GIL      | 3350.00 | ANALISTA   |
| JIMENEZ  | 1400.00 | EMPLEADO   |
+----------+---------+------------+
6 rows in set (0.00 sec)
```

Observamos en el ejemplo anterior que han sido seleccionadas aquellas filas en las que no se cumple la condición de que el oficio sea vendedor.

Podemos formar expresiones lógicas en las que intervengan varios operadores lógicos de manera similar a como se haría con expresiones aritméticas en las que intervienen varios operadores aritméticos.

Ejemplos:

```
mysql> SELECT apellido, salario, oficio
    -> FROM empleados
    -> WHERE NOT (oficio = 'VENDEDOR' AND salario > 1500);

+----------+---------+------------+
| APELLIDO | SALARIO | OFICIO     |
+----------+---------+------------+
| ALONSO   | 1400.00 | VENDEDOR   |
| LOPEZ    | 1350.50 | EMPLEADO   |
| MARTIN   | 1500.00 | VENDEDOR   |
| GARRIDO  | 3850.12 | DIRECTOR   |
| MARTINEZ | 2450.00 | DIRECTOR   |
| REY      | 6000.00 | PRESIDENTE |
| GIL      | 3350.00 | ANALISTA   |
| JIMENEZ  | 1400.00 | EMPLEADO   |
+----------+---------+------------+
8 rows in set (0.00 sec)

mysql> SELECT apellido, salario, oficio, dep_no
    -> FROM empleados
    -> WHERE oficio = 'VENDEDOR'
             AND salario>1500 OR dep_no = 20;

+----------+---------+----------+--------+
| APELLIDO | SALARIO | OFICIO   | DEP_NO |
+----------+---------+----------+--------+
| CALVO    | 1800.00 | VENDEDOR |     30 |
| GIL      | 3350.00 | ANALISTA |     20 |
| JIMENEZ  | 1400.00 | EMPLEADO |     20 |
+----------+---------+----------+--------+
3 rows in set (0.00 sec)
```

4.4 – Precedencia o prioridad en los operadores

En todo caso deberemos tener en cuenta la prioridad o precedencia del operador ya que puede afectar al resultado de la operación.
La precedencia u orden de prioridad es un concepto matemático, conocido por todos, que nos indica que operación se realizará primero en una expresión cuando hay varios operadores.

La evaluación de las operaciones en las expresiones se realiza de izquierda a derecha, pero respetando las reglas de prioridad.

Por ejemplo

8 + 4 * 5 = 28

Aunque la suma esté antes (más a la izquierda) que la multiplicación, primero se realiza la multiplicación y luego la suma. Esto es debido a que el operador * tiene mayor prioridad que el operador +.

El orden de precedencia o prioridad de los operadores determina, por tanto, el orden de evaluación de las operaciones de una expresión.

La tabla siguiente muestra los operadores disponibles agrupados y ordenados de mayor a menor por su orden de precedencia.

Prioridad	Operador	Operación
1º	*, /, DIV	Multiplicación, división
2º	+, -	Suma, resta.
3º	= , != , < , > , <= , >= , IS, LIKE , BETWEEN, IN	Comparación.
4º	NOT	Negación
5º	AND	Conjunción
6º	OR, XOR	Inclusión, exclusión

Esta es la prioridad establecida por defecto, los operadores que se encuentran en el mismo grupo tienen la misma precedencia. Se puede cambiar utilizando paréntesis.

En la expresión anterior, si queremos que la suma se realice antes que la división, lo indicaremos:

(8 + 4) * 5

En este caso el resultado será: 60

5 - Funciones predefinidas

Son funciones incorporadas por el gestor y son muy utilizadas en SQL y dan mucha potencia al lenguaje. Estas funciones predefinidas devuelven un valor dependiendo del valor de un argumento que se pasa en la llamada.

Cabe subrayar que las funciones no modifican los valores del argumento, indicado entre paréntesis, sino que devuelven un valor creado a partir de los argumentos que se le pasan en la llamada, y ese valor puede ser utilizado en cualquier parte de una sentencia SQL.

Existen muchas funciones predefinidas para operar con todo tipo de datos. A continuación se indican las funciones predefinidas más utilizadas. Estas funciones se tratarán con más detalle y se realizarán ejemplos en el tema 5 (punto 5.4) Y se realizarán ejercicios con estas funciones lo que ayudará a su mejor comprensión.

Vamos a ver estas funciones agrupadas según el tipo de datos con los que operan y/o el tipo de datos que devuelven.

Nota: estas funciones pueden variar según el sistema gestor que se utilice.

5.1 - Funciones numéricas o aritméticas

Función	Valor que devuelve.
ABS(*num*)	**Valor absoluto.** Valor absoluto de *num.*
CEIL(*num)*	**Función "Techo"** Devuelve el entero mas pequeño mayor que *num*
FLOOR(*num*)	**Función "Suelo"** Devuelve el entero mas grande menor que *num*
EXP(*num*)	**Potencia del número e** Devuelve el número e elevado a *num*
LN(*num*)	**Logaritmo neperiano** Devuelve el logaritmo en base e de *num*
LOG(*num*)	**Logaritmo** Devuelve el logaritmo en base 10 de *num*
MOD(*num1, num2).*	**Módulo** Resto de la división entera de *num1* por *num2*
PI()	**Pi** Devuelve el valor de la constante PI

POWER(*num1, num2).*	**Potencia** Devuelve *num1* elevado a *num2.*
RAND*()*	**Número aleatorio** Genera un número aleatorio entre 0 y 1
ROUND(*num1, num2)*	**Redondeo** Devuelve *num1* redondeado a *num2 d*ecimales. Si se omite *num2* redondea a 0 decimales.
SIGN(*num*).	**Signo** Si *num* < 0 devuelve -1, Si *num* = 0 devuelve 0 Si *num* > 0 devuelve 1.
SQRT(*num*).	**Raíz cuadrada** Devuelve la raíz cuadrada de *num*
TRUNCATE(*num1, num2).*	**Truncado** Devuelve *num1* truncado a *num2* decimales. Si se omite *num2* trunca a 0 decimales.

5.2 - Funciones de caracteres

Función	Valor que devuelve.
ASCII(*cad1*).	**ASCII** Código ASCII del carácter *cad1.*
CHAR(*num)*	**Carácter ASCII** Devuelve el carácter cuyo código ASCII es *num*
CONCAT(*cad1,cad2 [,cad3…].*)	**Concatenar** Concatena *cad1* con *cad2. Si* existiesen más cadenas, *cad3…,* las concatenaría a continuación
INSERT(*cad1, pos, ,len, cad2)*	**Insertar** Devuelve *cad1* con *len* caracteres desde *pos* en adelante sustituidos en *cad2*
LENGTH(*cad1*)	**Longitud** Devuelve la longitud de *cad1.*
LOCATE(*cad1,cad2,pos)*	**Localizar** Devuelve la posición de la primera ocurrencia de *cad1* en *cad2* empezando desde *pos*
LOWER(*cad1*)	**Minúsculas** La cadena *cad1* en minúsculas.

LPAD(*cad1, n, cad2)*	**Rellenar (Izquierda)** Añade a *cad1* por la izquierda *cad2*, hasta que tenga n caracteres. Si se omite *cad2,* añade blancos.
LTRIM(*cad1*)	**Suprimir (Izquierda)** Suprime blancos a la izquierda de *cad1.*
REPLACE(*cad1,cad2,cad3)*	**Reemplazar** Devuelve *cad1* con todas las ocurrencias de *cad2* reemplazadas por *cad3*
RPAD(*cad1, n, cad2)*	**Rellenar (Derecha)** Igual que LPAD pero por la derecha.
RTRIM(*c1*)	**Suprimir (Derecha)** Suprime blancos a la derecha de *c1.* Igual que LTRIM pero por la izquierda.
SUBSTR(*c1, n, m)*	**Subcadena** Devuelve una subcadena a partir de *c1* comenzando en La posición *n* tomando *m* caracteres.
UPPER(*cad1*)	**Mayúsculas** La cadena *cad1* en mayúsculas.

5.3 - Funciones de fecha

Función	Valor que devuelve.
ADDDATE(*Fecha, Num*)	**Incremento de días** Devuelve *Fecha* incrementada en *Num* días
SUBDATE(*Fecha, Num*)	**Decremento de días** Devuelve *Fecha* decrementada en *Num* días
DATE_ADD(*Fecha*, INTERVAL Num *Formato*)	**Incremento** Devuelve *Fecha* incrementada en Num veces lo indicado en *Formato* *Formato* puede ser entre otros: DAY, WEEK, MONTH, YEAR, HOUR, MINUTE, SECOND
DATE_SUB(Fecha, INTERVAL num *Formato*)	**Decremento** Devuelve Fecha decrementada en *Num* veces lo indicado en *Formato* *Formato* puede ser entre otros: DAY, WEEK, MONTH, YEAR, HOUR, MINUTE, SECOND
DATEDIFF(*Fecha1,Fecha2*)	**Diferencia de fechas** Devuelve el número de días entre *Fecha1* y *Fecha2*
DAYNAME(*Fecha*)	**Nombre del día de la semana** Devuelve el nombre del día de la semana de *Fecha*
DAYOFMONTH(*Fecha*)	**Día del mes** Devuelve el número del día del mes de *Fecha*

DAYOFWEEK(*Fecha*)	**Día de la semana** Devuelve el número del día de la semana de *Fecha* (1.Domingo, 2:Lunes.....7:Sábado)
DAYOFYEAR(*Fecha*)	**Día del año** Devuelve el número de día del año de *Fecha* (de 1 a 366)
WEEKOFYEAR(*Fecha*)	**Semana** Devuelve el número de semana de *Fecha* (de 1 a 53)
MONTH(*Fecha*)	**Mes** Devuelve el número de mes de *Fecha* (de 1 a 12)
YEAR(*Fecha)*	**Año** Devuelve el número de año con 4 dígitos de *Fecha* (de 0000 a 9999)
HOUR(*Tiempo)*	**Hora** Devuelve la hora de *Tiempo* (de 0 a 23)
MINUTE(*Tiempo*)	**Minutos** Devuelve los minutos de *Tiempo* (de 0 a 59)
SECOND(*Tiempo*)	**Segundos** Devuelve los segundos de *Tiempo* (de 0 a 59)
CURDATE()	Devuelve la fecha actual con el formato 'YYYY-MM-DD'
CURTIME()	Devuelve la hora actual con el formato 'HH:MM:SS
SYSDATE()	Devuelve la fecha y la hora actual con el formato 'YYYY-MM-DD HH:MM:SS'

Para la conversión de fechas a otro tipo de datos:

Función	Valor que devuelve.
DATE_FORMAT(*Fecha*,*Formato*)	Devuelve una cadena de caracteres con la Fecha con el Formato especificado. El formato es una cadena de caracteres que incluye las siguientes máscaras: **Mascara Descripción**

%a	Abreviatura (3 letras) del nombre del día de la semana
%b	Abreviatura (3 letra) del nombre mes
%c	Número del mes (1 a 12)
%e	Número del día del mes (0 a 31)
%H	Número de la hora en formato 24 horas (00 a 23)
%h	Número de la hora en formato 12 horas (01 a 12)
%i	Número de minutos (00 a 59)
%j	Número del día del año (001 a 366)
%M	Nombre del mes
%m	Número de mes (01 a 12)
%p	Am o PM
%r	Hora en formato 12 horas (hh:mm seguido de AM o PM)
%s	Número de segundos (00 a 59)
%T	Hora en formato 24 horas (hh:mm:ss)
%u	Número de semana en el año (00 a 53)
%W	Nombre del día de la semana
%w	Número del día de la semana (0:domingo a 6:Sábado)
%Y	Número de año con cuatro dígitos
%y	Número de año con dos dígitos

5.4 - Funciones de comparación

Función	Valor que devuelve.
GREATEST(*lista de valores)*	**Mayor de la lista** Devuelve el valor más grande de una lista de columnas o expresiones de columna
LEAST(*lista de valores)*	**Menor de la lista** Devuelve el valor más pequeño de una lista de columnas o expresiones de columna
IFNULL*(exp1, exp2)*	**Conversión de nulos** Si *exp1* es nulo devuelve *exp2,* sino devuelve *exp1*
ISNULL(*exp)*	**Comprobación de nulo** Devuelve 1(`True`) si *exp* es `NULL` y 0 (`False`) en caso contrario
STRCMP(*cad1,cad2*)	**Comparación de cadenas** Devuelve 1(`True`) si *cad1* y *cad2* son iguales, 0(`False`) si no lo son y `NULL` si alguna de ellas es nula

5. 5 - Otras funciones

Función	Valor que devuelve.
DATABASE()	**Base de datos** Nombre de la base de datos actual
USER()	**Usuario** Devuelve el usuario y el host de la sesión usuario@host
VERSION()	**Versión** Devuelve una cadena indicando la versión que estamos utilizando

6 – Valores Nulos (NULL)

En SQL la ausencia de valor se expresa como valor nulo (NULL). Es importante ser conscientes de que esta ausencia de valor o valor nulo no equivale en modo alguno al valor 0 o a una cadena de caracteres vacía.

Hay que tener cuidado al operar con columnas que pueden contener valores nulos, pues la lógica cambia. Vamos a ver como se trabaja con estos valores nulos.

- Si realizamos operaciones aritméticas hay que tener en cuenta que cualquier expresión aritmética que contenga algún valor nulo dará como resultado un valor nulo.

Así, por ejemplo, si intentamos visualizar la expresión formada por las columnas SALARIO + COMISION de la tabla empleados la salida será similar a la siguiente:

```
mysql> SELECT apellido, salario, comision, salario + comision
    -> FROM empleados;
+----------+---------+----------+--------------------+
| APELLIDO | SALARIO | COMISION | SALARIO + COMISION |
+----------+---------+----------+--------------------+
| ALONSO   | 1400.00 |   400.00 |            1800.00 |
| LOPEZ    | 1350.50 |     NULL |               NULL |
| MARTIN   | 1500.00 |  1600.00 |            3100.00 |
| GARRIDO  | 3850.12 |     NULL |               NULL |
| MARTINEZ | 2450.00 |     NULL |               NULL |
| REY      | 6000.00 |     NULL |               NULL |
| CALVO    | 1800.00 |     0.00 |            1800.00 |
| GIL      | 3350.00 |     NULL |               NULL |
| JIMENEZ  | 1400.00 |     NULL |               NULL |
+----------+---------+----------+--------------------+
```

En el ejemplo anterior observamos que la expresión SALARIO + COMISION retornará un valor nulo siempre que alguno de los valores sea nulo incluso aunque el otro no lo sea. También podemos observar que el valor 0 en la comisión retorna el valor calculado de la expresión.

- Si comparamos expresiones que contienes el valor nulo con otro valor nulo el resultado no es ni mayor ni menor ni igual. En SQL un valor nulo ni siquiera es igual a otro valor nulo tal como podemos apreciar en el siguiente ejemplo:

```
mysql> SELECT *
    -> FROM empleados
    -> WHERE comision = NULL;
Empty set (0.00 sec)
```

La explicación es que un valor nulo es indeterminado, y por tanto, no es igual ni distinto de otro valor nulo. Cuando queremos comprobar si un valor es nulo emplearemos el operador `IS NULL` (o `IS NOT NULL` para comprobar que es distinto de nulo):

```
mysql> SELECT emp_no, apellido,oficio,fecha_alta,comision,salario
    -> FROM empleados
    -> WHERE comision IS NULL;
+--------+----------+------------+------------+----------+---------+
| EMP_NO | APELLIDO | OFICIO     | FECHA_ALTA | COMISION | SALARIO |
+--------+----------+------------+------------+----------+---------+
|   7521 | LOPEZ    | EMPLEADO   | 1981-05-08 |     NULL | 1350.50 |
```

```
|   7698 | GARRIDO  | DIRECTOR   | 1981-05-01 |     NULL | 3850.12 |
|   7782 | MARTINEZ | DIRECTOR   | 1981-06-09 |     NULL | 2450.00 |
|   7839 | REY      | PRESIDENTE | 1981-11-17 |     NULL | 6000.00 |
|   7876 | GIL      | ANALISTA   | 1982-05-06 |     NULL | 3350.00 |
|   7900 | JIMENEZ  | EMPLEADO   | 1983-03-24 |     NULL | 1400.00 |
+--------+----------+------------+------------+----------+---------+
```

Como acabamos de ver, los valores nulos en muchas ocasiones pueden representar un problema, especialmente en columnas que contienen valores numéricos.
Para evitar estos problemas y asegurarnos de la ausencia de valores nulos en una columna ya vimos que existe una restricción `NOT NULL` (es una orden de definición de datos) que impide que se incluyan valores nulos en una columna.
En caso de que permitamos la existencia de valores nulos hay que evitar estos problemas y utilizar la función `IFNULL` (que veremos en detalle más adelante) que se utiliza para devolver un valor determinado en el caso de que el valor del argumento sea nulo. Así nos aseguramos en las operaciones aritméticas que no hay nulos con los que operar. Por ejemplo si queremos sumar el *salario* + *comision* debemos utilizar `IFNULL` (*comision*,0) retornará 0 cuando el valor de comisión sea nulo y sumaremos un 0 a *salario*.

```
mysql> SELECT apellido, salario, comision,
    ->           salario + IFNULL(comision,0) "SALARIO TOTAL"
    -> FROM empleados;
+----------+---------+----------+---------------+
| APELLIDO | SALARIO | COMISION | SALARIO TOTAL |
+----------+---------+----------+---------------+
| ALONSO   | 1400.00 |   400.00 |       1800.00 |
| LOPEZ    | 1350.50 |     NULL |       1350.50 |
| MARTIN   | 1500.00 |  1600.00 |       3100.00 |
| GARRIDO  | 3850.12 |     NULL |       3850.12 |
| MARTINEZ | 2450.00 |     NULL |       2450.00 |
| REY      | 6000.00 |     NULL |       6000.00 |
| CALVO    | 1800.00 |     0.00 |       1800.00 |
| GIL      | 3350.00 |     NULL |       3350.00 |
| JIMENEZ  | 1400.00 |     NULL |       1400.00 |
+----------+---------+----------+---------------+
9 rows in set (0.00 sec)
```

7 – Expresiones y condiciones

Una expresión es un conjunto variables, constantes o literales, funciones, operadores y paréntesis. Los paréntesis sirven para variar el orden de prioridad y sólo son obligatorios en ese caso.
Un caso especial de expresión es lo que llamamos condición. Condición es un expresión cuyo resultado es V*erdadero/Falso/Nulo (*`True/False/Null`*)*

Las sentencias SQL pueden incluir expresiones y condiciones con nombres de columnas y literales. Ejemplos de expresiones:

- `SALARIO + COMISION ->` Devuelve un valor numérico como resultado de sumar al salario del empleado la comisión correspondiente.

- `EMPLEADOS.DEPT_NO = DEPARTAMENTOS.DEPT_NO ->` Devuelve verdadero o falso dependiendo de que el número de departamento del empleado seleccionado coincida con el número de departamento del departamento seleccionado.

- `FECHA_AL BETWEEN '1980-01-01' AND '1982-10-01'->` Devuelve verdadero si la fecha de alta del empleado seleccionado se encuentra entre las dos fechas especificadas.

- `COMISION IS NULL ->` dará como resultado verdadero si la comisión del empleado seleccionado no tiene ningún valor.

Por ejemplo la siguiente sentencia visualizará el apellido, la fecha de alta, el salario y la suma del salario más una gratificación de 1.000 euros

```
mysql> SELECT apellido, fecha_alta, salario,
    -> salario + 1000 "SALARIO CON GRATIFICACION"
    -> FROM EMPLEADOS;
+----------+------------+---------+---------------------------+
| APELLIDO | FECHA_ALTA | SALARIO | SALARIO CON GRATIFICACION |
+----------+------------+---------+---------------------------+
| ALONSO   | 1981-02-23 | 1400.00 |                   2400.00 |
| LOPEZ    | 1981-05-08 | 1350.50 |                   2350.50 |
| MARTIN   | 1981-09-28 | 1500.00 |                   2500.00 |
| GARRIDO  | 1981-05-01 | 3850.12 |                   4850.12 |
| MARTINEZ | 1981-06-09 | 2450.00 |                   3450.00 |
| REY      | 1981-11-17 | 6000.00 |                   7000.00 |
| CALVO    | 1981-09-08 | 1800.00 |                   2800.00 |
| GIL      | 1982-05-06 | 3350.00 |                   4350.00 |
| JIMENEZ  | 1983-03-24 | 1400.00 |                   2400.00 |
+----------+------------+---------+---------------------------+
9 rows in set (0.00 sec)
```

Hay unas reglas de sintaxis que se deben respetar. Un error relativamente frecuente consiste en utilizar expresiones del tipo: 1000 >= SALARIO <= 2000

Este tipo de expresiones no es correcto y no producirá el resultado deseado, ya que al evaluar la primera parte de la expresión se sustituirá por un valor lógico de tipo `true/false/null` y este resultado no puede compararse con un valor numérico.
La expresión correcta sería:

SALARIO BETWEEN 1000 AND 2000

O bien:

SALARIO >= 1000 AND SALARIO <= 2000.

TEMA 3. CREACIÓN DE TABLAS

Paulina Barthelemy

1 - Consideraciones previas a la creación de una tabla

Antes de escribir la sentencia para crear una tabla debemos pensar una serie de datos y requisitos que va a ser necesario definir en la creación.
Es importante pensar en la tabla que se quiere crear tomando las decisiones necesarias antes de escribir el formato de la creación.

1.1 – Definición de la tabla

Por una parte unas consideraciones básicas como:

- El nombre de la tabla.
- El nombre de cada columna.
- El tipo de dato almacenado en cada columna.
- El tamaño de cada columna.

1.2 - Restricciones en las tablas

Por otra parte, información sobre lo que se pueden almacenar las filas de la tabla. Esta información serán las restricciones que almacenamos en las tablas. Son una parte muy importante del modelo relacional pues nos permiten relacionar las tablas entre sí y poner restricciones a los valores de que pueden tomas los atributos (columnas) de estas tablas.

Restricciones, llamadas **`CONSTRAINTS`**, son condiciones que imponemos en el momento de crear una tabla para que los datos se ajusten a una serie de características predefinidas que mantengan su integridad.

Se conocen con su nombre en inglés y se refieren a los siguientes conceptos:

a) **`NOT NULL.`** Exige la existencia de valor en la columna que lleva la restricción.

b) **`DEFAULT.`** Proporciona un valor por defecto cuando la columna correspondiente no se le da valor en la instrucción de inserción.
Este valor por defecto debe ser una constante. No se permiten funciones ni expresiones.

c) **`PRIMARY KEY.`** Indica una o varias columnas como dato o datos que identifican unívocamente cada fila de la tabla. Sólo existe una por tabla y en ninguna fila puede tener valor `NULL`, por definición.
Es obligatoria su existencia en el modelo relacional.

d) **FOREIGN KEY.** Indica que una determinada columna de una tabla, va a servir para referenciar a otra tabla en la que está definida la misma columna (columna o clave referenciada). El valor de la clave ajena deberá coincidir con uno de los valores de esta clave referenciada o ser NULL. No existe límite en el número de claves ajenas que pueda tener una tabla. Como caso particular, una clave ajena puede referenciar a la misma tabla en la que está. Para poder crear una tabla con clave ajena deberá estar previamente creada la tabla maestra en la que la misma columna es clave primaria.

e) **UNIQUE.** Indica que esta columna o grupo de columnas debe tener un valor único. También admite valores nulos. Al hacer una nueva inserción se comprobará que el valor es único o NULL. Algunos sistemas gestores de bases de datos relacionales generan automáticamente índices para estas columnas

f) **CHECK.** Comprueba si el valor insertado en esa columna cumple una determinada condición.

2 - Formato genérico para la creación de tablas.

La sentencia SQL que permite crear tablas es CREATE TABLE.

2.1 - Formato básico para la creación de tablas

Comenzaremos con un formato básico de creación de tabla al que iremos añadiendo, posteriormente, otras informaciones.

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
( NombreColumna TipoDato [ , NombreColumna TipoDato ].... );
```

donde **NombreTabla .....** es el identificador elegido para la tabla
NombreColumna ..... es el identificador elegido para cada columna
TipoDato..... indica el tipo de dato que se va a almacenar en esa columna. (Ver apartado 3 del Tema 2).

El nombre de la tabla debe ser único en la base de datos. Los nombres de columnas deben ser únicos dentro de la tabla.

Para el nombre de la tabla y de las columnas se elegirán identificadores de acuerdo con las reglas del gestor de la base de datos (Ver apartado 2.1 del Tema 2). Estos identificadores no pueden coincidir con palabras reservadas.

Existirán tantas definiciones de columna como datos diferentes se vayan a almacenar en la tabla que estamos creando, todas ellas separadas por comas.

La cláusula IF NOT EXISTS previene el posible error generado si existiese una tabla con ese nombre.

2.2 – Ejemplos básicos de creación de tablas

Los siguientes ejemplos de creación de tablas se van a utilizar siempre nuevas tablas para no alterar las tablas anteriormente utilizadas.

Realizaremos con un ejemplo de una biblioteca queremos guardar los datos de los socios en una tabla socios y los préstamos que se realizan en una tabla prestamos. Empezaremos con los formatos básicos e iremos añadiendo cláusulas. En cada apartado le daremos un nombre diferente a las tablas para evitar problemas (si en la instrucción e creación el nombre de la tabla ya existe se produce un error)

1-. Crear una tabla socios con los datos de los socios:

- Numero de socio — Número entero de 4 dígitos
- Apellidos del socios — Cadena de 14 caracteres máximo
- Teléfono — Cadena de 9 caracteres
- Fecha de alta como socio — Fecha
- Dirección — Cadena de 20 carateres máximo

- Codigo postal Número entero de 5 dígitos

```
mysql> CREATE TABLE socios_0
    ->  (socio_no INT(4),
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5));
Query OK, 0 rows affected (0.30 sec)
```

El campo `telefono` lo creamos tipo `CHAR` en lugar de `VARCHAR` porque siempre tendrá 9 caracteres

2. Crear una tabla `prestamos` para guardar los préstamos hechos a los socios con los datos:
   - Número del préstamo Número entero de 2 dígitos
   - Código del socio Número entero de 4 dígitos

```
mysql> CREATE TABLE prestamos_0
    ->   (num_prestamo INT(2),
    ->    socio_no INT(4));
Query OK, 0 rows affected (0.08 sec)
```

2.3 – Formatos para la creación de tablas con la definición de restricciones

Los valores por defecto pueden ser: constantes, funciones `SQL` o las variables `USER` o `SYSDATE`.
La restricciones pueden definirse de dos formas, que llamaremos

1. **Restriccion1** Definición de la restricción a nivel de columna
2. **Restriccion2** Definición de la restricción a nivel de tabla

Estas restricciones , llamadas `CONSTRAINTS` se pueden almacenar con o sin nombre. Si no se lo damos nosotros lo hará el sistema siguiendo una numeración correlativa, que es poco representativa.

Es conveniente darle un nombre, para después podernos referirnos a ellas si las queremos borrar o modificar. Estos nombres que les damos a las `CONSTRAINTS` deben ser significativos para hacer mas fácil las referencias. Por ejemplo:

- *pk_NombreTabla* para `PRIMARY KEY`
- *fk_NombreTabla1_NombreTabla2* `FOREIGN KEY` donde NombreTabla1 es la tabla donde se crea y NombreTabla2 es la tabla a la que referencia.
- *uq_NombreTabla_NombreColumna* para `UNIQUE`

2.3.1 - Formato para la creación de tablas con restricciones definidas a nivel de columna

Definimos cada restricción al mismo tiempo que definimos la columna correspondiente.

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
  ( NombreColumna TipoDato [Restriccion1]
      [ , NombreColumna TipoDato  [Restriccion1] ..... ] );
```

donde ***Restriccion1.....*** es la definición de la restricción a nivel de columna

Las restricciones solo se pueden definir de esta forma si afectan a una sola columna, la que estamos definiendo es ese momento.

Definición de los diferentes tipos de `CONSTRAINTS` a nivel de tabla (Restriccion1)

a) **CLAVE PRIMARIA**
```
PRIMARY KEY
```
b) **POSIBILIDAD DE NULO**
```
NULL | NOT NULL
```
c) **VALOR POR DEFECTO**
```
DEFAULT ValorDefecto
```
d) **UNICIDAD**
```
UNIQUE
```
e) **COMPROBACION DE VALORES**
```
CHECK (Expresion)
```
Nota: Esta cláusula de SQL estándar, en MySQL en la versión 5 está permitida pero no implementada

f) **CLAVE AJENA**
```
REFERENCES NombreTabla [( NombreColumna )]
```
Notación: el nombre de tabla referenciada es el nombre de la tabla a la que se va a acceder con la clave ajena. Si la columna que forma la clave referenciada en dicha tabla no tiene el mismo nombre que en la clave ajena, debe indicarse su nombre detrás del de la tabla referenciada y dentro del paréntesis. Si los nombres de columnas coinciden en la clave ajena y en la primaria, no es necesario realizar esta indicación.

g) **AUTO INCREMENTO**
```
AUTO_INCREMENT
```
Aunque no es propiamente una restricción, pero como parte de la definición de una columnas podemos indicar que sea `AUTO_INCREMENT.` De esta forma el sistema gestor irá poniendo valores en esta columna incrementándolos de 1 en 1 respecto al anterior y empezando por 1 (opción por defecto que se puede modificar cambiando las opciones de las tabla en la creación lo que queda fuera de este curso). Esta definición sólo se puede aplicar sobre columnas definidas como y enteras y que sean claves.

2.3.2 - Ejemplos de definición de restricciones a nivel de columna

Veremos un ejemplo de cada caso donde se van definiendo las columnas con las correspondientes restricciones.

a) **PRIMARY KEY.** El numero de socio en la *tabla socios*

```
mysql> CREATE TABLE socios_1a
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5));
Query OK, 0 rows affected (0.08 sec)
```

b) **NOT NULL.** La columna teléfono es obligatoria en la tabla `socios`, nunca irá sin información.

```
mysql> CREATE TABLE socios_1b
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14),,
    ->  telefono CHAR(9) NOT NULL,
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5));
Query OK, 0 rows affected (0.05 sec)
```

c) **DEAFAULT.** En ausencia de valor el campo fecha _ alta tomará el valor de 1 de enero de 2000.

```
mysql> CREATE TABLE socios_1c
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9) NOT NULL,
    ->  fecha_alta DATE DEFAULT '2000-01-01',
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5));
Query OK, 0 rows affected (0.11 sec)
```

d) **UNIQUE.** La columna apellido será única en la tabla `socios`.

```
mysql> CREATE TABLE socios_1d
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14) UNIQUE,
    ->  telefono CHAR(9) NOT NULL,
    ->  fecha_alta DATE DEFAULT '2000-01-01',
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5) );
Query OK, 0 rows affected (0.06 sec)
```

e) **CHECK.** Se comprobará que la columna cdigo_postal corresponde a Madrid (valores entre 28000 y 28999)

```
mysql> CREATE TABLE socios_1e
    ->  (socio_no INT(4) PRIMARY KEY,
    ->  apellidos VARCHAR(14) UNIQUE,
    ->  telefono CHAR(9) NOT NULL,
    ->  fecha_alta DATE DEFAULT '2000-01-01',
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5)
```

```
            CHECK (codigo_postal BETWEEN 28000 AND 28999) );
Query OK, 0 rows affected (0.06 sec)
```

f) **FOREIGN KEY.** El campo socio_num de la tabla *prestamos* tendrá que tener valores existentes en el campo num_socio de la tabla *socios* o valor nulo.

```
mysql> CREATE TABLE prestamos
    ->  (num_prestamo INT(2) PRIMARY KEY,
    ->  socio_no INT(4) REFERENCES socios_1e(socio_no));
Query OK, 0 rows affected (0.17 sec)
```

g) **AUTO INCREMENTO.** Crearemos una tabla con una columna AUTO_INCREMENT y posteriormente (siguiente tema) insertaremos valores.

```
mysql> CREATE TABLE inventario
    -> (num INT(2) AUTO_INCREMENT PRIMARY KEY,
    ->  descripcion VARCHAR(15));
Query OK, 0 rows affected (0.49 sec)
```

2.3.3 - Formato para la creación de tablas con restricciones definidas a nivel de tabla

En este caso definimos todas las restricciones al final de la sentencia, una vez terminada la definición de las columnas.

```
CREATE TABLE [IF NOT EXISTS  NombreTabla
(  NombreColumna TipoDato [ , NombreColumna TipoDato..... ]
   [Restriccion2 [ , Restrccion2]…..  );
```

donde ***Restriccion2.....*** es la definición de la restricción a nivel de tabla

Las restricciones siempre se pueden definir de esta forma tanto si afectan a una sola columna como a varias columnas y puede darse un nombre a cada una de las restricciones.

Definición de los diferentes tipos de `CONSTRAINTS` a nivel de tabla (Restriccion2)

a) **`PRIMARY KEY`**

```
[CONSTRAINT [NombreConstraint]]
          PRIMARY KEY (Nombrecolumna [,NombreColumna....] )
```

b) **`UNIQUE`**

```
[CONSTRAINT [NombreConstraint]] UNIQUE (NombreColumna
                                        [,NombreColumna...] )
```

c) **`CHECK`**

```
[CONSTRAINT [NombreConstraint]] CHECK (Expresion)
```

d) **`FOREIGN KEY`**

```
[CONSTRAINT [NombreConstraint ]]
```

```
FOREIGN KEY (NombreColumna[, NombreColumna...])
REFERENCES ( NombreTabla [NombreColumna [, NombreColumna.....]])
```

Notación: los nombres de columna o columnas que siguen a la cláusula FOREIGN KEY es aquella o aquellas que están formando la clave ajena. Si hay más de una se separan por comas. El nombre de tabla referenciada es el nombre de la tabla a la que se va a acceder con la clave ajena. Si la columna o columnas que forman la clave referenciada en dicha tabla no tienen el mismo nombre que en la clave ajena, debe indicarse su nombre detrás del de la tabla referenciada y dentro de paréntesis. Si son más de una columna se separan por comas. Si los nombres de columnas coinciden en la clave ajena y en la primaria, no es necesario realizar esta indicación.

2.3.4 - Ejemplos de definición de restricciones a nivel de tabla

Veremos un ejemplo de cada caso donde se van definiendo la columna con la correspondiente restricción. Algunos ejemplos con nombre de constraint y otros sin él.
En un ejemplo, igual que el apartado anterior, de una biblioteca queremos guardar los datos de los socios en una tabla *socios* y los préstamos que se realizan en una tabla *prestamos*

a)**PRIMARY KEY.** El numero de socio será la clave primaria en la *tabla socios.* Será obligatoriamente no nulo y único.

```
mysql> CREATE TABLE socios_2a
    ->  (socio_no INT(4),
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5),
    ->  CONSTRAINT PK2_DEPARTAMENTOS PRIMARY KEY (socio_no));
Query OK, 0 rows affected (0.08 sec)
```

b) **UNIQUE.** El campo apellido es único. Tendrá valores diferentes en cada fila o el valor nulo

```
mysql> CREATE TABLE socios_2b
    ->  (socio_no INT(4),
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5),
    ->  CONSTRAINT PK_SOCIOS PRIMARY KEY(socio_no),
    ->  CONSTRAINT UQ_UNIQUE UNIQUE (apellidos));
Query OK, 0 rows affected (0.06 sec)
```

c)**CHECK.** La columna codigo_postal no admitirá como válidas aquellas filas en las que el código postal no tenga valores entre 28.000 y 28.999 (correspondientes a Madrid)

```
mysql> CREATE TABLE socios_2c
    ->  (socio_no INT(4),
    ->  apellidos VARCHAR(14),
    ->  telefono CHAR(9),
```

```
    ->  fecha_alta DATE,
    ->  direccion VARCHAR(20),
    ->  codigo_postal INT(5),
    ->  CONSTRAINT PK_DEPARTAMENTOS PRIMARY KEY(socio_no),
    ->  CONSTRAINT UQ_UNIQUE UNIQUE(apellidos),
    ->  CONSTRAINT CK_CODIGO
    ->                    CHECK (codigo_postal BETWEEN 28000 AND
28999) );
Query OK, 0 rows affected (0.17 sec)
```

d) **FOREIGN KEY.** El número de socio en una *tabla prestamos* será clave ajena referenciando a la columna correspondiente de la tabla `socios`.

```
mysql> CREATE TABLE prestamos_2
    ->  (num_prestamo INT(2),
    ->  socio_no INT(4) ,
    ->  CONSTRAINT PK_PRESTAMOS PRIMARY KEY(num_prestamo),
    ->  CONSTRAINT FK_SOCIO_PRESTAMOS   FOREIGN KEY (socio_no)
        REFERENCES socios_2c(socio_no) );
Query OK, 0 rows affected (0.13 sec)
```

2.4 - Integridad referencial

La definición de claves ajenas nos permiten mantener la integridad referencial en una base de datos relacional. Hemos dicho que la columna o columnas definidas como clave ajena deben tomar valores que se correspondan con un valor existente de la clave referenciada. La pregunta es: ¿qué sucede si queremos borrar o modificar un valor de la clave primaria refenciada? Pues que el sistema debe impedirnos realizar estas acciones pues dejaría de existir la integridad referencial.

Por ejemplo si tenemos

```
CREATE TABLE departamentos
      (dep_no INT(4)
             CONSTRAINT pk_departamentos PRIMARY KEY.....);

CREATE TABLE empleados
      (....dep_no INT(4) CONSTRAINT fk_empleados_departamentos
                         REFERENCES departamentos(dep_no).... );
```

En este caso empleados.dep_no solo puede tomar valores que existan en departamentos.dep_no pero ¿que sucede si queremos borrar o modificar un valor de departamentos.dep_no.? El sistema no nos lo permitirá si existen filas con ese valor en la tabla de la clave ajena.

Sin embargo, en ocasiones, será necesario hacer estas operaciones. Para mantener la integridad de los datos, al borrar `(DELETE)`, modificar `(UPDATE)` una fila de la tabla referenciada, el sistema no nos permitirá llevarlo a cabo si existe una fila con el valor referenciado. También se conoce como `RESTRICT`. Es la opción por defecto, pero existen las siguientes opciones en la definición de la clave ajena:

- **CASCADE**. El borrado o modificación de una fila de la tabla referenciada lleva consigo el borrado o modificación en cascada de las filas de la tabla que contiene la clave ajena. Es la más utilizada.
- **SET NULL**. El borrado o modificación de una fila de la tabla referenciada lleva consigo

poner a NULL los valores de las claves ajenas en las filas de la tabla que referencia.

- **SET DEFAULT**. El borrado o modificación de una fila de la tabla referenciada lleva consigo poner un valor por defecto en las claves ajenas de la tabla que referencia.
- **NO ACTION**. El borrado o modificación de una fila de la tabla referenciada solo se produce si no existe ese valor en la tabla que contiene la clave ajena. Tiene el mismo efecto que RESTRICT.

Formato de la definición de clave ajena con opciones de referencia

```
REFERENCES NombreTabla [ ( NombreColumna [ , NombreColumna ] ) ]
      [ON DELETE {CASCADE| SET NULL|NO ACTION | SET DEFAULT| RESTRICT}]
      [ON UPDATE {CASCADE| SET NULL|NO ACTION | SET DEFAULT| RESTRICT}]
```

por ejemplo

```
CREATE TABLE empleados
      (....
       dep_no NUMBER(4) CONSTRAINT FK_EMPLEADOS_DEPARTAMENTOS
                        REFERNCES departamentos(dep_no)
                        ON DELETE SET NULL
                        ON UPDATE CASCADE .... );
```

Esto quiere decir que el sistema pondrá nulos en dep_no de la tabla empleados si se borra el valor correspondiente en departamentos y modificará el valor de dep_no en la tabla empleados con el nuevo valor si se modifica la columna dep_no en la tabla departamentos.

En el tema siguiente veremos con más detalle los borrados y modificaciones en los casos en los que existan estas cláusulas en las definiciones de las tablas, realizando algún ejemplo.

2.5 - Formato completo de la creación de tablas

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
       ( NombreColumna TipoDato [Restriccion1 ]
              [ , NombreColumna TipoDato [Restriccion1..... ]
        [Restriccion2 [ , Restrccion2....]    );
```

Notación: los diferentes formatos de definición de restricciones, restricción1 y restricción2, están entre corchetes porque son opcionales, pudiéndose elegir entre ambos sólo si la restricción afecta a una sola columna.

Vamos a crear la tabla socios y prestamos con el formato completo. Algunas CONSTRAINTS las creamos a nivle de columna y otras de tabla:

```
mysql> CREATE TABLE socios
    ->      (socio_no INT(4),
    ->       apellidos VARCHAR(14),
    ->       telefono CHAR(9) NOT NULL,
    ->       fecha_alta DATE DEFAULT '2000-01-01',
    ->       direccion VARCHAR(20),
```

```
    ->       codigo_postal INT(5),
    ->       CONSTRAINT PK_DEPARTAMENTOS PRIMARY KEY(socio_no),
    ->       CONSTRAINT UQ_UNIQUE UNIQUE(apellidos),
    ->       CONSTRAINT CK_CODIGO
    ->                     CHECK (codigo_postal BETWEEN 28000 AND
28999) );
Query OK, 0 rows affected (0.53 sec)

mysql> CREATE TABLE prestamos
    ->     (num_prestamo INT(2) PRIMARY KEY,
    ->      socio_no INT(4) ,
    ->      CONSTRAINT FK_SOCIO_PRESTAMOS FOREIGN KEY (socio_no)
    ->         REFERENCES socios(socio_no) );
Query OK, 0 rows affected (0.17 sec)
```

Utilizaremos estas tablas en el tema siguiente para realizar inserciones, modificaciones y borrados de filas.

3 - Modificación de la definición de una tabla.

Una vez que hemos creado una tabla, a menudo se presenta la necesidad de tener que modificarla. La sentencia SQL que realiza esta función es **ALTER TABLE.**

3.1 - Formato general para la modificación de tablas

La especificación de la modificación es parecida a la de la sentencia CREATE pero varía según el objeto SQL del que se trate.

```
ALTER TABLE NombreTabla
EspecificacionModificacion [ , EspecificacionModificacion..... ]
```

donde **NombreTabla.....** nombre de la tabla se desea modificar.
EspecificacionModificacion..... las modificaciones que se quieren realizarse sobre la tabla

Las modificaciones que se pueden realizar sobre una tabla son:

- Añadir una nueva columna
- Añadir un nueva restricción
- Borrar una columna
- Borrar una restricción
- Modificar una columna sin cambiar su nombre
- Modificar una columna y cambiar su nombre
- Renombrar la tabla

a) AÑADIR UNA NUEVA COLUMNA

```
ADD [COLUMN] NombreColumna TipoDato  [ Restriccion1 ]
```

Puede añadirse una nueva columna y todas las restricciones, salvo NOT NULL. La razón es que esta nueva columna tendrá los valores NULL al ser creada.

b) AÑADIR UNA CONSTRAINT

```
ADD [CONSTRAINT [NombreConstraint] ]
      PRIMARY KEY (NombreColumna [, NombreColumna...] )
ADD [CONSTRAINT [NombreConstraint] ]
      FOREIGN KEY (NombreColumna [, NombreColumna...] )
      REFERENCES NombreTabla[(NombreColumna[,NombreColumna...] )]
ADD [CONSTRAINT [NombreConstraint] ]
      UNIQUE (NombreColumna [, NombreColumna...  ] )
```

c) BORRAR UNA COLUMNA

```
DROP [COLUMN] NombreColumna
```

d) BORRAR UNA CONSTRAINT

```
DROP PRIMARY KEY
```

```
DROP FOREIGN KEY NombreConstraint
```

e) MODIFICAR UNA COLUMNA SIN CAMBIAR SU NOMBRE

```
MODIFY [COLUMN] NombreColumna TipoDato  [Restriccion1]
```

f) MODIFICAR LA DEFINCION DE UNA COLUMNA Y SU NOMBRE

```
CHANGE [COLUMN] NombreColumnaAntiguo
                NombreColumnaNuevo TipoDatos [Restriccion1]
```

f) RENOMBRAR LA TABLA

```
RENAME [TO] NombreTablaNuevo
```

3.2 – Ejemplos de modificaciones de tablas

a) Ejemplo de añadir una columna

Añadir la columna para la dirección de correo electrónico, `direccion_correo_2c`, a la tabla de `socios_2c` con un tipo de dato alfanumérico de 20 caracteres.

```
mysql> ALTER TABLE socios_2c ADD (direccion_correo_e varchar(9));
Query OK, 0 rows affected (0.23 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

b) Ejemplo de añadir una restriccion

Añadir en la tabla `socios_2c` la restricción `UNIQUE` para la columna `telefono`.

```
mysql> ALTER TABLE socios_2c
    -> ADD CONSTRAINT uq_socios_telefono UNIQUE(telefono);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

c) Ejemplo de borrar una columna

Borrar la columna `fecha_alta` de la tabla `socios_a`

```
mysql> ALTER TABLE socios_1a
    -> DROP COLUMN fecha_alta;
Query OK, 0 rows affected (0.32 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

d) Ejemplos de borrado de restricciones

Borrar la clave primaria de la tabla `socios_1a`

```
mysql> ALTER TABLE socios_1a
    -> DROP PRIMARY KEY;
Query OK, 0 rows affected (0.66 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

e) Ejemplos de modificación de una columna
Modificar la tabla `socios_1a` para poner `NOT NULL` en la columna *`apellidos`*.

```
mysql> ALTER TABLE socios_1a
    -> MODIFY apellidos VARCHAR(14) NOT NULL;
Query OK, 0 rows affected (0.59 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

f) Ejemplo de cambio de definición de una columna
Modificar la tabla `socios_1`*`a`* cambiándole el nombre a la columna `apellidos` por `apellidos_a` e incrementar de 14 a 20 caracteres el tamaño.

```
mysql> ALTER TABLE socios_1a
    ->  CHANGE apellidos apellidos_a VARCHAR(20);
Query OK, 0 rows affected (0.36 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

f) Renombrar una tabla
Cambiar el nombre de la tabla `socios_1`**`a`** por `socios1a`

```
mysql> ALTER TABLE socios_1a
    -> RENAME TO socios1a;
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT * FROM socios_1a;
ERROR 1146 (42S02): Table 'test.socios_1a' doesn't exist
```

4 - Eliminación de una tabla.

Para borrar una tabla y su contenido de la base de datos se utiliza la sentencia `DROP TABLE.`

4.1 - Formato para eliminar una tabla.

DROP TABLE [IF EXISTS] NombreTabla
[CASCADE | RESTRICT] ;

La cláusula `IF EXISTS` previene los errores que puedan producirse si no existe la tabla que queremos borrar.
Nota: las cláusulas `CASCADE Y RESTRICT` para borrado en cascada y borrado de las restricciones están permitidas pero no implementadas en esta versión.

4.2 – Ejemplos de borrado de una tabla

1- Borraremos las tablas `departamentos2 y empleados2` enlazadas con una clave ajena. Hay que tener cuidado con el orden:

```
mysql> DROP TABLE  departamentos2;
ERROR 1217 (23000): Cannot delete or update a parent row: a
foreign key constraint fails
```

Debemos borrar primero `empleados2:`

```
mysql> DROP TABLE empleados2;
Query OK, 0 rows affected (0.10 sec)
mysql> DROP TABLE  departamentos2;
Query OK, 0 rows affected (0.06 sec)
```

2 - Borraremos una tabla `empleados7,` que no existe, con la cláusula `IF EXISTS` y vemos que no nos da error.

```
mysql> DROP TABLE IF EXISTS empleados7;
Query OK, 0 rows affected, 1 warning (0.00 sec)
```

5 – Renombrado de una tabla

Permite cambiar de nombre una tabla, asignándole un nombre nuevo y el nombre antiguo desaparece.

5.1 - Formato para renombrar una tabla

```
RENAME TABLE NombreTablaAntiguo TO NombreTablaNuevo
  [, NombreTablaAntiguo TO NombreTablaNuevo .....]
```

Notación: se pueden renombrar varias tablas en una sentencia por eso van entre corchetes las siguientes tablas a renombrar.

donde **NombreTablaAntiguo .....** es el nombre antiguo de la tabla, que debe existir
NombreTablaNuevo..... es el nombre nuevo de la tabla, que no debe existir.

Permite renombrar en la misma instrucción una o varias tablas.

5.2 – Ejemplos de renombrado de una tabla

1- Renombrar la tabla *socios* para que su nuevo nombre sea socios2

```
mysql> RENAME TABLE socios TO socios2;
Query OK, 0 rows affected (0.05 sec)

mysql> select * from socios;
ERROR 1146 (42S02): Table 'test.socios' doesn't exist
```

TEMA 4. ACTUALIZACION DE TABLAS

Paulina Barthelemy

1 - Introducción

Como ya hemos explicado el lenguaje SQL incluye un conjunto de sentencias que permiten modificar, borrar e insertar nuevas filas en una tabla. Este subconjunto de comandos del lenguaje SQL recibe la denominación de *Lenguaje de Manipulación de Datos* (DML)

Las órdenes que no permiten actualizar los valores de las tablas son:

- INSERT para la inserción de filas
- UPDATE para la modificación de filas
- DELETE para el borrado de filas.

2 - Inserción de nuevas filas en la base de datos

Para añadir nuevas filas a una tabla utilizaremos la sentencia INSERT.

2.1 – Formato de la inserción una fila

La sentencia para inserción de filas es INSERT cuyo formato típico es:

```
INSERT INTO NombreTabla [( NombreColumna [,NombreColumna...] ) ]
VALUES (Expresion [, Expresión ...] );
```

Notación: la lista de columnas en las que insertamos va es opcional, por lo que va entre corchetes. Si no se especifica se esperan valores para todas las columnas.

Donde **NombreTabla**..... es el nombre de la tabla en la que queremos insertar una fila.
NombreColumna..... incluye las columnas en las que queremos insertar.
Expresion..... indica las expresiones cuyos valores resultado se insertarán, existiendo una correspondencia con la lista de columnas anterior

Como se ve en el formato también se pueden insertar filas en una tabla sin especificar la lista de columnas pero en este caso la lista de valores a insertar deberá coincidir en número y posición con las

columnas de la tabla. El orden establecido para las columnas será el indicado al crear la tabla (el mismo que aparece al hacer una descripción de la tabla DESC NombreTabla*).*

2.2 – Ejemplos de inserción de filas

Por ejemplo, para insertar una nueva fila en la tabla *departamentos* introduciremos la siguiente instrucción:

```
mysql> INSERT INTO departamentos(dep_no, dnombre, localidad)
    -> VALUES (50, 'MARKETING','BILBAO');
Query OK, 1 row affected (0.08 sec)

mysql> select * from departamentos;
+--------+---------------+-----------+
| DEP_NO | DNOMBRE       | LOCALIDAD |
+--------+---------------+-----------+
|     10 | CONTABILIDAD  | BARCELONA |
|     20 | INVESTIGACION | VALENCIA  |
|     30 | VENTAS        | MADRID    |
|     40 | PRODUCCION    | SEVILLA   |
|     50 | MARKETING     | BILBAO    |
+--------+---------------+-----------+
5 rows in set (0.00 sec)
```

Este formato de inserción tiene, además, las siguientes características:

- En la lista de columnas se pueden indicar todas o algunas de las columnas de la tabla. En este último caso, aquellas columnas que no se incluyan en la lista quedarán sin ningún valor en la fila insertada, es decir, se asumirá el valor `NULL` o el valor por defecto para las columnas que no figuren en la lista. En aso de una columna definida como `NOT NULL` tomará el valor 0 si es numérica o blancos sino.
- Los valores incluidos en la lista de valores deberán corresponderse posicionalmente con las columnas indicadas en la lista de columnas, de forma que el primer valor de la lista de valores se incluirá en la columna indicada en primer lugar, el segundo en la segunda columna, y así sucesivamente.
- Se puede utilizar cualquier expresión para indicar un valor siempre que el resultado de la expresión sea compatible con el tipo de dato de la columna correspondiente.

Algunos ejemplos de inserciones válidas son:
1- Inserción de un socio con socio_no=1000

```
mysql> INSERT INTO Socios
(socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
    ->    VALUES ( 1000,'LOPEZ SANTOS','916543267', '2005-01-08',
    ->            'C. REAL 5',28400);
Query OK, 1 row affected (0.05 sec)
```

2 - Inserción de un socio con socio_no=1001

```
mysql> INSERT INTO Socios
 (socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
```

```
    ->     VALUES ( 1001,'PEREZ CERRO','918451256', '2005-01-12',
    ->             'C. MAYOR 31',28400);
Query OK, 1 row affected (0.06 sec)
```

3 - Inserción de un socio con socio_no=1002

```
mysql> INSERT INTO Socios
 (socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
    ->     VALUES ( 1002,'LOPEZ PEREZ','916543267', '2005-01-18',
    ->             'C. REAL 5',28400);
Query OK, 1 row affected (0.06 sec)
```

4 - Inserción de un socio con socio_no=1003 sin valor en teléfono (como es un campo `NOT NULL` en lugar de nulos pondrá blancos)

```
mysql> INSERT INTO socios
    -> (socio_no,apellidos,fecha_alta,direccion,codigo_postal)
    ->     VALUES ( 1003,'ROMA LEIVA', '2005-01-21',
    ->             'C. PANADEROS 9 ',28431);
Query OK, 1 row affected (0.06 sec)
```

5- Inserción de un préstamo para el socio con socio_no=1000

```
mysql> INSERT INTO prestamos
    ->     (num_prestamo, socio_no)
    ->     VALUES ( 1,1000);
Query OK, 1 row affected (0.05 sec)
```

6 - Inserción de un préstamo para el socio con socio_no=1002

```
mysql> INSERT INTO prestamos
    ->     (num_prestamo, socio_no)
    ->        VALUES ( 2,1002);
Query OK, 1 row affected (0.07 sec)
```

7 - Inserción de un socio con socio_no=1004, con una instrucción INSERT sin lista de columnas:

```
mysql> INSERT INTO socios
    -> VALUES ( 1004,'GOMEZ DURUELO','918654329', '2005-01-31',
    ->             'C. REAL 15',28400);
Query OK, 1 row affected (0.43 sec)
```

8 - Inserción de un socio con socio_no=1005, con una instrucción INSERT sin valor en el campo fecha que tiene un valor por defecto (pondrá ese valor 2000-01-01)

```
mysql> INSERT INTO socios
    ->   (socio_no,apellidos,telefono,direccion,codigo_postal)
    ->      VALUES ( 1005,'PEÑA MAYOR','918515256','C. LARGA 31',
    ->                 28431);
Query OK, 1 row affected (0.07 sec)
```

9 - Inserción de un socio con socio_no=1004, con una instrucción INSERT sin lista de columnas:

```
mysql> INSERT INTO prestamos
    ->      VALUES ( 4,1004);
Query OK, 1 row affected (0.40 sec)
```

Algunos intentos de inserciones erróneas:

1 – Inserción en la tabla `socios` en la que falta un valor para la columna dirección:

```
mysql> INSERT INTO Socios
    ->   (socio_no,apellidos,telefono,fecha_alta,direccion)
    ->    VALUES ( 1005,'LOPEZ SANTOS','916543267', '2005-01-08',
    ->            'C. REAL 5',28400);
ERROR 1136(21S01):Column count doesn't match value count at row 1
```

2 – Inserción en la tabla `socios` de una columna con clave primaria duplicada:

```
mysql> INSERT INTO Socios
 (socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
    ->    VALUES ( 1000,'LOPEZ SANTOS','916543267', '2005-01-08',
    ->            'C. REAL 5',28400);
ERROR 1062 (23000): Duplicate entry '1000' for key 1
```

3 - Inserción en la tabla `socios` de una fila con valores duplicados en la columna apellidos:

```
mysql> INSERT INTO Socios
 (socio_no,apellidos,telefono,fecha_alta,direccion,codigo_postal)
    ->    VALUES ( 1009,'LOPEZ SANTOS','916543267', '2005-01-18',
    ->            'C. REAL 5',28400);
ERROR 1062 (23000): Duplicate entry 'LOPEZ SANTOS' for key 2
```

4 - Inserción de una fila en la tabla `prestamos` con un valor en la columna socios_no que no existe en la tabla `socios`

```
mysql> INSERT INTO prestamos
    ->     (num_prestamo, socio_no)
    ->        VALUES ( 3,1009);
ERROR 1216 (23000): Cannot add or update a child row: a foreign key
constraint fails
```

Comprobación de los valores insertados:

```
mysql> SELECT *
    -> FROM socios;
```

socio_no	apellidos	telefono	fecha_alta	direccion	codigo_postal
1000	LOPEZ SANTOS	916543267	2005-01-08	C. REAL 5	28400
1001	PEREZ CERRO	918451256	2005-01-12	C. MAYOR 31	28400
1002	LOPEZ PEREZ	916543267	2005-01-18	C. REAL 5	28400
1003	ROMA LEIVA		2005-01-21	C. PANADEROS 9	28431
1004	GOMEZ DURUELO	918654329	2005-01-31	C. REAL 15	28400
1005	PEÑA MAYOR	918515256	2000-01-01	C. LARGA 31	28431

```
   6 rows in set (0.00 sec)

SELECT *
FROM prestamos;

       mysql> SELECT *
           -> FROM prestamos;
       +--------------+----------+
       | num_prestamo | socio_no |
       +--------------+----------+
       |            1 |     1000 |
       |            2 |     1002 |
       |            4 |     1004 |
       +--------------+----------+
       3 rows in set (0.00 sec)
```

Ejemplo de inserción con un campo autonumérico, de paso creamos una tabla con una columna autonumérica.

```
       mysql> CREATE TABLE inventario
           -> (num INT(2) AUTO_INCREMENT PRIMARY KEY,
           ->  descripcion VARCHAR(15));
       Query OK, 0 rows affected (0.49 sec)
```

1 – Inserción sin enumerar la columna autonumérica

```
       mysql> INSERT INTO inventario (descripcion)
           -> VALUES ('ARMARIO BLANCO');
       Query OK, 1 row affected (0.42 sec)

       mysql> INSERT INTO inventario (descripcion)
           -> VALUES ('MESA MADERA');
       Query OK, 1 row affected (0.05 sec)
```

2 – Inserción de toda la fila (en este caso debe ponerse NULL en la columna correspondiente)

```
       mysql> INSERT INTO inventario
           -> VALUES (NULL,'ORDENADOR');
       Query OK, 1 row affected (0.07 sec)

       mysql> INSERT INTO inventario
           -> VALUES (NULL,'SILLA GIRATORIA');
       Query OK, 1 row affected (0.06 sec)
```

Comprobación de los valores insertados:

```
       mysql> SELECT *
           -> FROM inventario;
       +-----+-----------------+
       | num | descripcion     |
```

```
+-----+-----------------+
|   1 | ARMARIO BLANCO  |
|   2 | MESA MADERA     |
|   3 | ORDENADOR       |
|   4 | SILLA GIRATORIA |
+-----+-----------------+
4 rows in set (0.00 sec)
```

3 - Modificación de filas.

En ocasiones necesitaremos modificar alguno de los datos de las filas existentes de una tabla. Por ejemplo, cambiar el salario o el departamento de uno o varios empleados, etcétera. En estos casos utilizaremos la sentencia UPDATE.

3.1 – Formato de la modificación de filas

La sentencia de modificación de filas es UPDATE cuyo formato genérico es el siguiente:

```
UPDATE NombreTabla
SET NombreColumna = Expresion  [, NombreColumna = Expresion....]
[WHERE Condición];
```

Notación: puede actualizarse una o varias columnas, por lo que la segunda actualización va entre corchetes La cláusula WHERE aparece entre corchetes porque es opcional. En el caso de que no se utilice, la actualización afectará a toda la tabla.

Donde **NombreTabla……** indica la tabla destino donde se encuentran las filas que queremos borrar.
NombreColumna……..... indica el nombre de la columna cuyo valor queremos modificar
Expresión…… es una expresión cuyo valor resultado será el nuevo valor de la columna
Condición…… es la condición que deben cumplir las filas para que les afecte la modificación.

Recordamos que una expresión es un conjunto de datos, constantes y variables, operadores y funciones. Y una condición es una expresión cuyo resultado es `verdadero/falso/nulo`
Para aquellas filas en las que al evaluar la condición el resultado es `verdadero` se modificará el valor de la columna poniendo como nuevo valor el resultado de evaluar la expresión.
Si se quiere modificar más de una columna de la misma tabla se indicarán separadas por comas.

3.2 – Ejemplos de modificación de filas

Partimos de la tabla socios

```
mysql> SELECT  * FROM socios;
+----------+--------------+-----------+------------+----------------+---------------+
| socio_no | apellidos    | telefono  | fecha_alta | direccion      | codigo_postal |
+----------+--------------+-----------+------------+----------------+---------------+
|     1000 | LOPEZ SANTOS | 916543267 | 2005-01-08 | C. REAL 5      |         28400 |
|     1001 | PEREZ CERRO  | 918451256 | 2005-01-12 | C. MAYOR 31    |         28400 |
|     1002 | LOPEZ PEREZ  | 916543267 | 2005-01-18 | C. REAL 5      |         28400 |
|     1003 | ROMA LEIVA   |           | 2005-01-21 | C. PANADEROS 9 |         28431 |
|     1004 | GOMEZ DURUELO | 918654329 | 2005-01-31 | C. REAL 15     |         28400 |
|     1005 | PEÑA MAYOR   | 918515256 | 2000-01-01 | C. LARGA 31    |         28431 |
+----------+--------------+-----------+------------+----------------+---------------+
6 rows in set (0.01 sec)
```

1 - Supongamos que se desea cambiar la dirección del socio de número 1000 y la nueva dirección es :

'C.CUESTA 2'.

```
mysql> UPDATE socios
    -> SET direccion = 'C.CUESTA 2'
    -> WHERE socio_no = 1000;
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

2 - Supongamos que se desea cambiar el teléfono del socio de número 1000 y el nuevo teléfono es 918455431

```
mysql> UPDATE socios
    -> SET telefono = '918455431'
    -> WHERE socio_no = 1000;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

También podíamos haber modificado las dos columnas con una sola sentencia:

```
mysql> UPDATE socios
    -> SET telefono = '918455431',direccion='C.CUESTA 2'
    -> WHERE socio_no = 1000;
Query OK, 0 rows affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

Las actualizaciones anteriores afectan a una única fila pero podemos escribir comandos de actualización que afecten a varias filas:

```
mysql> UPDATE socios
    -> SET codigo_postal = 28401
    -> WHERE codigo_postal = 28400;
Query OK, 4 rows affected (0.07 sec)
Rows matched: 4  Changed: 4  Warnings: 0
```

Si no se incluye la cláusula WHERE la actualización afectará a todas las filas. El siguiente ejemplo modifica la fecha de alta de todos los empleados al valor 1 de enero de 2005.

```
mysql> UPDATE socios
    -> SET fecha_alta = '2005-01-01';
Query OK, 6 rows affected (0.06 sec)
Rows matched: 6  Changed: 6  Warnings: 0
```

Podemos comprobar las modificaciones:

```
mysql> SELECT *
    -> FROM socios;
+----------+--------------+-----------+------------+----------------+--------------+
| socio_no | apellidos    | telefono  | fecha_alta | direccion      | codigo_postal |
+----------+--------------+-----------+------------+----------------+--------------+
|     1000 | LOPEZ SANTOS | 918455431 | 2005-01-01 | C.CUESTA 2     |         28401 |
|     1001 | PEREZ CERRO  | 918451256 | 2005-01-01 | C. MAYOR 31    |         28401 |
|     1002 | LOPEZ PEREZ  | 916543267 | 2005-01-01 | C. REAL 5      |         28401 |
|     1003 | ROMA LEIVA   |           | 2005-01-01 | C. PANADEROS 9 |         28431 |
|     1004 | GOMEZ DURUELO | 918654329 | 2005-01-01 | C. REAL 15     |         28401 |
|     1005 | PEÑA MAYOR   | 918515256 | 2005-01-01 | C. LARGA 31    |         28431 |
+----------+--------------+-----------+------------+----------------+--------------+
6 rows in set (0.00 sec)
```

4 - Eliminación de filas.

La sentencia `DELETE` es la que nos permite eliminar una o más filas de una tabla.

4.1 – Formato de la eliminación de filas

Para eliminar o suprimir filas de una tabla utilizaremos la sentencia `DELETE`. Su formato genérico es el siguiente:

```
DELETE FROM NombreTabla
[WHERE Condicion];
```

Notación: la cláusula WHERE es opcional por eso aparece entre corchetes. Si no se especifica se borraran todas las filas de la tabla

donde **NombreTabla**..... indica la tabla destino donde se encuentran las filas que queremos borrar
Condición..... es la condición que deben cumplir las filas a borrar

4.2 – Ejemplos de la eliminación de filas

A continuación se muestran algunos ejemplos de instrucciones DELETE

```
mysql> DELETE
    -> FROM socios
    -> WHERE socio_no = 1001;
Query OK, 1 row affected (0.08 sec)
```

Hay que tener cuidado con las claves ajenas. Como `prestamos` tiene una clave ajena que referencia a `socios` si pretendemos borrar un socio que tiene préstamos nos encontraremos con un error.

```
mysql> DELETE
    -> FROM socios
    -> WHERE socio_no = 1002;
ERROR 1217 (23000): Cannot delete or update a parent row: a
foreign key constraint fails
```

Podemos comprobar las modificaciones:

```
mysql> SELECT *
    -> FROM socios;
+----------+--------------+-----------+------------+----------------+--------------+
| socio_no | apellidos    | telefono  | fecha_alta | direccion      | codigo_postal |
+----------+--------------+-----------+------------+----------------+--------------+
|     1000 | LOPEZ SANTOS  | 918455431 | 2005-01-01 | C.CUESTA 2     |         28401 |
```

```
|     1002 | LOPEZ PEREZ   | 916543267 | 2005-01-01 | C. REAL 5      |         28401 |
|     1003 | ROMA LEIVA    |           | 2005-01-01 | C. PANADEROS 9 |         28431 |
|     1004 | GOMEZ DURUELO | 918654329 | 2005-01-01 | C. REAL 15     |         28401 |
|     1005 | PEÑA MAYOR    | 918515256 | 2005-01-01 | C. LARGA 31    |         28431 |
+----------+---------------+-----------+------------+----------------+---------------+
6 rows in set (0.00 sec)
```

5 - Restricciones de integridad y actualizaciones.

Como ya hemos vistos los gestores de bases de datos relacionales permiten especificar ciertas condiciones que deban cumplir los datos contenidos en las tablas, como por ejemplo:

- Restricción de clave primaria (PRIMARY KEY) : columnas que identifican cada fila
- Restricción de clave ajena (FOREIGN KEY): columnas que hacen referencia a otras de la misma o de otras tablas
- Restricción de comprobación de valores (CHECK): conjunto de valores que puede o debe tomar una columna.
- Restricción de no nulo (NOT NULL): columnas que tienen que tener necesariamente un valor no nulo.
- Restricción de unicidad (UNIQUE): columnas que no pueden tener valores duplicados.

5.1 - Control de las restricciones de integridad referencial

Estas restricciones dan lugar a que no podamos hacer cualquier modificación en los datos de las tablas. No nos estará permitido hacer inserciones ni modificaciones con valores no permitidos en las columnas ni borrar filas a las que referencien otras filas de la misma o de otra tabla

En nuestras tablas de ejemplo están definidas las siguientes restricciones:
a) **Claves primarias** (PRIMARY KEY). Sirven para referirse a una fila de manera inequívoca. No se permiten valores repetidos.

TABLA DEPARTAMENTOS: PRIMARY KEY (DEP_NO)
TABLA EMPLEADOS: PRIMARY KEY (EMP_NO)
TABLA CLIENTES: PRIMARY KEY(CLIENTE_NO)
TABLA PRODUCTOS: PRIMARY KEY (PRODUCTO_NO)
TABLA PEDIDOS: PRIMARY KEY (PEDIDO_NO)

Como ya hemos visto no podemos hacer inserciones con valores repetidos de las claves primarias.

b) **Claves ajenas** (FOREIGN KEY): Se utilizan para hacer referencia a columnas de otras tablas. Cualquier valor que se inserte en esas columnas tendrá su equivalente en la tabla referida. Opcionalmente se pueden indicar las acciones a realizar en caso de borrado o cambio de las columnas a las que hace referencia .

```
TABLA EMPLEADOS: FOREIGN KEY (DEP_NO)
                        REFERENCES DEPARTAMENTOS(DEP NO)
TABLA CLIENTES: FOREIGN KEY (VENDEDOR_NO)
                        REFERENCES EMPLEADOS(EMP_NO)
TABLA PEDIDOS: FOREIGN KEY (PRODUCTO_NO)
                        REFERENCES PRODUCTOS(PRODUCTO NO)
TABLA PEDIDOS: FOREIGN KEY (CLIENTE_NO)
                        REFERENCES CLIENTES(CLIENTE NO)
```

Las claves ajenas sirven para relacionar dos tablas entre sí y limitan los valores que puede tomar esa columna a los valores existentes en ese momento en la columna a la que referencian, pudiendo tomar valores existentes en esa columna o nulos.

- Esto nos limita los valores de las claves ajenas no podrán tomar valores que no existan en las columnas referenciadas
- Los valores de las claves primarias no podrán actualizarse si existen filas que los referencian.

Lo vemos con un ejemplo:
La tabla EMPLEADOS tiene una clave ajena DEP_NO que referencia la clave primaria DEP_NO de la tabla DEPARTAMENTOS.
Si existe un departamento con dep_no = 20 y existen filas de empleados con este valor de dep_no

1. No podremos borrar ni modificar el valor de dep_no = 20 de la tabla DEPARTAMENTOS
2. No podremos modificar el valor del campo dep_no de la tabla EMPLEADOS a un valor que no exista en la tala DEPARTAMENTOS.

Estas condiciones se determinaron al diseñar la base de datos y se especificaron e implementaron mediante el lenguaje de definición de datos (DDL). Por lo tanto, todos los comandos de manipulación de datos deberán respetar estas restricciones de integridad ya que en caso contrario el comando fallará y el sistema devolverá un error.

5.2 - Ejemplos de borrados y modificaciones en cascada

Vamos a ver con un ejemplo como funciona el borrado en cascada.
Creamos las tablas `departamentos2` y `empleados2` con una clave ajena con borrado en cascada e insertamos los valores desde las tablas `departamentos` *y* `empleados`

```
mysql> CREATE TABLE departamentos2
    ->  (dep_no INT(4),
    ->    dnombre VARCHAR(14),
    ->    localidad VARCHAR(10),
    ->  CONSTRAINT PK2_DEP PRIMARY KEY (DEP_NO)
    ->  );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO departamentos2
    ->  SELECT dep_no, dnombre, localidad
    ->          FROM departamentos;
Query OK, 4 rows affected (0.03 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE empleados2
    -> (emp_no          INT(4),
    ->   apellido       VARCHAR(8),
    ->   oficio         VARCHAR(15),
    ->   director       INT(4),
    ->   fecha_alta     DATE,
    ->   dep_no         INT (2),
    -> CONSTRAINT PK_EMPLEADOS_EMP_NO2 PRIMARY KEY (emp_no),
    -> CONSTRAINT FK_EMP_DEP_NO2 FOREIGN KEY (dep_no)
    ->      REFERENCES departamentos2(dep_no) ON DELETE CASCADE
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO empleados2
```

```
    -> SELECT emp_no, apellido, oficio, director,
    ->           fecha_alta, dep_no
    -> FROM empleados;
Query OK, 9 rows affected (0.03 sec)
Records: 9  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM departamentos2;
+--------+---------------+-----------+
| dep_no | dnombre       | localidad |
+--------+---------------+-----------+
|     10 | CONTABILIDAD  | BARCELONA |
|     20 | INVESTIGACION | VALENCIA  |
|     30 | VENTAS        | MADRID    |
|     40 | PRODUCCION    | SEVILLA   |
+--------+---------------+-----------+
4 rows in set (0.00 sec)

SELECT * FROM empleados2;

mysql> SELECT * FROM empleados2;
+--------+----------+------------+----------+------------+--------+
| emp_no | apellido | oficio     | director | fecha_alta | dep_no |
+--------+----------+------------+----------+------------+--------+
|   7499 | ALONSO   | VENDEDOR   |     7698 | 1981-02-23 |     30 |
|   7521 | LOPEZ    | EMPLEADO   |     7782 | 1981-05-08 |     10 |
|   7654 | MARTIN   | VENDEDOR   |     7698 | 1981-09-28 |     30 |
|   7698 | GARRIDO  | DIRECTOR   |     7839 | 1981-05-01 |     30 |
|   7782 | MARTINEZ | DIRECTOR   |     7839 | 1981-06-09 |     10 |
|   7839 | REY      | PRESIDENTE |     NULL | 1981-11-17 |     10 |
|   7844 | CALVO    | VENDEDOR   |     7698 | 1981-09-08 |     30 |
|   7876 | GIL      | ANALISTA   |     7782 | 1982-05-06 |     20 |
|   7900 | JIMENEZ  | EMPLEADO   |     7782 | 1983-03-24 |     20 |
+--------+----------+------------+----------+------------+--------+
9 rows in set (0.00 sec)
```

Ahora vamos a borrar una fila en la tabla departamentos. Si no existiese borrado en cascada, debido a la integridad referencial, no podríamos borrar ningún departamento que tuviese empleados. De esta forma al borrar un departamento se borrarán todos los empleados de ese departamento.

```
mysql> DELETE FROM departamentos2
    -> WHERE dep_no=10;
Query OK, 1 row affected (0.05 sec)

SELECT * FROM departamentos2;

mysql> SELECT * FROM departamentos2;
+--------+---------------+-----------+
| dep_no | dnombre       | localidad |
+--------+---------------+-----------+
|     20 | INVESTIGACION | VALENCIA  |
|     30 | VENTAS        | MADRID    |
|     40 | PRODUCCION    | SEVILLA   |
+--------+---------------+-----------+
3 rows in set (0.00 sec)
```

```
SELECT * FROM empleados2;

mysql> SELECT * FROM empleados2;
+--------+----------+----------+----------+------------+--------+
| emp_no | apellido | oficio   | director | fecha_alta | dep_no |
+--------+----------+----------+----------+------------+--------+
|   7499 | ALONSO   | VENDEDOR |     7698 | 1981-02-23 |     30 |
|   7654 | MARTIN   | VENDEDOR |     7698 | 1981-09-28 |     30 |
|   7698 | GARRIDO  | DIRECTOR |     7839 | 1981-05-01 |     30 |
|   7844 | CALVO    | VENDEDOR |     7698 | 1981-09-08 |     30 |
|   7876 | GIL      | ANALISTA |     7782 | 1982-05-06 |     20 |
|   7900 | JIMENEZ  | EMPLEADO |     7782 | 1983-03-24 |     20 |
+--------+----------+----------+----------+------------+--------+
6 rows in set (0.00 sec)
```

6 - Control de transacciones: COMMIT y ROLLBACK.

Los gestores de bases de datos disponen de dos comandos que permiten confirmar o deshacer los cambios realizados en la base de datos:

- COMMIT: confirma los cambios realizados haciéndolos permanentes .
- ROLLBACK: deshace los cambios realizados.

Cuando hacemos modificaciones en las tablas estas no se hacen efectivas (llevan a disco) hasta que no ejecutamos la sentencia `COMMIT`. Cuando ejecutamos comandos `DDL` (definición de datos) se ejecuta un `COMMIT` automático, o cuando cerramos la sesión.

El comando `ROLLBACK` no permite deshacer estos cambios sin que lleguen a validarse. Cuando ejecutamos este comando se deshacen todos los cambios hasta el último `COMMIT` ejecutado.

Hay dos formas de trabajar con `AUTO_COMMIT` activado (validación automática de los cambios) o desactivado. Si está activado se hace `COMMIT` automáticamente cada sentencia y no es posible hacer `ROLLBACK`. Si no lo está, tenemos la posibilidad de hacer `ROLLBACK` después de las sentencias `DML (INSERT, UPDATE, DELETE)` dejando sin validar los cambios. Es útil para hacer pruebas.

Existe una variable, `AUTO_COMMIT`, que indica la forma de trabajo y tiene el valor 1 si está en modo `AUTO_COMMIT` y 0 si no lo está. Por defecto el MySQL está en modo `AUTO_COMMIT`. Su valor se puede cambia con la sentencia:

```
mysql> SET AUTOCOMMIT = 0;
Query OK, 0 rows affected (0.41 sec)
```

Vamos a verlo con un ejemplo:

```
mysql> SELECT *
    -> FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
+-----+----------------+
4 rows in set (0.00 sec)

mysql> SET AUTOCOMMIT = 0;
Query OK, 0 rows affected (0.41 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
+-----+----------------+
```

```
4 rows in set (0.00 sec)

mysql> INSERT INTO inventario
    -> VALUES (NULL,'IMPRESORA');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
|   5 | IMPRSORA       |
+-----+----------------+
4 rows in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
+-----+----------------+
4 rows in set (0.00 sec)
```

Para hacer los ejercicios y los ejemplos puede ser interesante modificar esta variable y así disponer de la posibilidad de hacer `ROLLBACK`. Cada vez que se inicie una sesión estará en modo `AUTO_COMMIT` que es el valor por defecto y el `ROLLBACk` no será aplicable.

```
mysql> SET AUTOCOMMIT = 1;
Query OK, 0 rows affected (0.41 sec)

mysql> INSERT INTO inventario
    -> VALUES (NULL,'ARCHIVADOR');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
|   6 | ARCHIVADOR     |
+-----+----------------+
```

```
5 rows in set (0.00 sec)

mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM inventario;
+-----+----------------+
| num | descripcion    |
+-----+----------------+
|   1 | ARMARIO BLANCO |
|   2 | MESA MADERA    |
|   3 | ORDENADOR      |
|   4 | SILLA GIRATORIA|
|   6 | ARCHIVADOR     |
+-----+----------------+
5 rows in set (0.00 sec)
```