LENGUAJE SQL. GESTION DE DATOS

TEMA 1. EL LENGUAJE DE GESTIÓN DE BASES DE DATOS

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_AITA	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 Ejemplo

En este ejemplo utilizaremos, además de las tablas EMPLEADOS y DEPARTAMENTOS cuya forma y contenido ya hemos visto, las tablas de CLIENTES, PRODUCTOS y PEDIDOScuya 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 | TALLERESESTESO 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 ASOCIADOSS.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 (0.00sec)
```

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 latabla 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.– 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

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.

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 oliterales

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 oigual

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  |
```

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 VERDADER*O* 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 demáscasos.
- 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 ANDcuando 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 losoperadores

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 eelevado a *num*
LN(*num*)	**Logaritmo neperiano** Devuelve el logaritmo en base ede *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 NULLy 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 NULLsi 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 SQLla 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 |
+----------+------------+---------+--------------------------+
```

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

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 detablas

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.

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,
    -> direccionVARCHAR(20),
    -> codigo_postalINT(5));
Query OK, 0 rows affected (0.30 sec)
```

El campo telefonolo creamos tipo CHARen lugar de VARCHAR porque siempre tendrá 9 caracteres

2. Crear una tabla prestamospara guardar los préstamos hechos a los socios con los datos:

- Número del préstamo Número entero de 2dígitos
- Código del socio Número entero de 4dígitos

```
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 KEYdonde 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) AUTOINCREMENTO

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,
 ->     direccionVARCHAR(20),
 ->     codigo_postalINT(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,
 ->     direccionVARCHAR(20),
 ->     codigo_postalINT(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',
 ->     direccionVARCHAR(20),
 ->     codigo_postalINT(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) PRIMARYKEY,
 ->     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 (0.06sec)

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) PRIMARYKEY,
 ->     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_noINT(4)REFERENCESsocios_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 (0.49sec)
```

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 TABLEdepartamentos
        (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
                         REFERNCESdepartamentos(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_noen 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]NombreColumnaTipoDato      [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 detablas

a) Ejemplo de añadir una columna

Añadir la columna para la dirección de correo electrónico, direccion_correo_2c, a la tabla de socios_2ccon 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_2cla 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_altade 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_1apara 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 apellidospor apellidos_ae incrementar de 14 a 20 caracteres el tamaño.

```
mysql> ALTER TABLE socios_1a
    ->    CHANGE apellidos apellidos_aVARCHAR(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 (0.10sec)
mysql> DROP TABLE departamentos2;
Query OK, 0 rows affected (0.06sec)
```

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 TOsocios2;
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

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 SQLrecibe 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 defilas

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 (0.08sec)

mysql> select * fromdepartamentos;
+--------+---------------+-----------+
| 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. LARGA31',
    ->                          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 row1
```

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. PANADEROS9 |          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 (0.00sec)

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 (0.49sec)
```

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. PANADEROS9 |         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 (0.01sec)
```

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. PANADEROS9 |        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 (0.00sec)
```

4 - Eliminación de filas.

La sentencia DELETEes 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 | GOMEZDURUELO | 918654329 | 2005-01-01 | C. REAL 15     |         28401 |
|     1005 | PEÑA MAYOR   | 918515256 | 2005-01-01 | C. LARGA 31    |         28431 |
+----------+--------------+-----------+------------+----------------+---------------+
6 rows in set (0.00sec)
```

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 DELETECASCADE
    -> );
Query OK, 0 rows affected (0.06sec) 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 COMMITautomático, 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 COMMITejecutado.

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 | SILLAGIRATORIA |
+-----+----------------+

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 | SILLAGIRATORIA |
mysql> INSERT INTO inventario  VALUES (NULL,'IMPRESORA');

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

```
|     1 | ARMARIO BLANCO|
|     2 | MESA MADERA   |
|     3 | ORDENADOR     |
|     4 | SILLAGIRATORIA|
|     5 | IMPRSORA      |
+-----+---------------+
```

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

mysql> SELECT * FROM inventario;

+-----+---------------+
|     1 | ARMARIO BLANCO|
|     2 | MESA MADERA   |
|     3 | ORDENADOR     |
|     4 | SILLAGIRATORIA|
+-----+---------------+
```

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 | SILLAGIRATORIA|
|     6 | ARCHIVADOR    |
+-----+---------------+

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 | SILLAGIRATORIA|
|     6 | ARCHIVADOR    |
+-----+---------------+
4  rows in set (0.00 sec
```

PARTE II LENGUAJE SQL. CONSULTA DE DATOS

TEMA 5. CONSULTAS SENCILLAS

1 - 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.

Las primeras consultas van a ser escritas con un formato inicial de la sentencia SELECT, que se irá completando durante este tema y en temas siguientes.

Vamos a comenzar a utilizar las sentencias del lenguaje SQL. En cada apartado iremos escribiendo los formatos de las sentencias con lo que hemos visto hasta ese momento.

2 – Consultas sencillas

Corresponden al formato mínimo de la instrucción de selección. Las cláusulas que aparecen en ellas,
SELECT y FROM, son obligatorias.

La consulta más sencilla consiste en recuperar una o varias columnas o expresiones relacionadas de una tabla.

2.1 – Formato mínimo de selección

Formato inicial de la sentencia SELECT

```
SELECT [ALL/DISTINCT] ExpresionColumna [, ExpresionColumna....]
FROM NombreTabla;
```

Notación: ALL y DISTINCT aparecen entre corchetes porque son opcionales y separados por la barra / porque hay que elegir entre ambos. La segunda Expresión de columna aparece entre corchetes por es opcional y seguida de puntos suspensivos porque puede repetirse las veces que se quiera.

donde ***ExpresionColumna...*** es un conjunto de nombres de columnas, literales o constantes, operadores, funciones y paréntesis.

NombreTabla..... es el nombre de la tabla de la que queremos seleccionar filas.

ALL..... obtiene los valores de todos los elementos seleccionados en todas las filas, aunque sean repetidos. Es la opción por defecto y no suele escribirse

DISTINCT..... obtiene los valores no repetidos de todos los elementos.

En caso de que queramos mostrar varias expresiones estas irán separadas por comas.

Este es el formato mínimo con las cláusulas SELECT y FROM que son siempre obligatorias. A este formato mínimo le iremos añadiendo otras cláusulas opcionales en los siguientes apartados y siguientes temas.

2.2 - Consultas de todas las filas

En lugar de las expresiones puede parecer el carácter ***** que indica que deben seleccionares todas las columnas de la tabla. No se ha escrito para no complicar el formato inicial y hacerlo más fácil de comprender. El formato sería:

SELECT { * | [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna*]....] }
FROM *NombreTabla*

*Notación: hay que escoger obligatoriamente una de las opciones, entre las de expresiones de columnas y el asterisco *****. Por eso aparecen entre llaves y separadas por una barra vertical | las posibles opciones.*

2.3 – Ejemplos de consultas sencillas

1. Obtener todos los empleados de la tabla empleados con todos sus datos.

```
mysql> SELECT * FROM empleados;
+--------+----------+-----------+----------+-----------+---------+----------+--------+
| EMP_NO | APELLIDO | OFICIO    | DIRECTOR | FECHA_ALTA | SALARIO | COMISION |DEP_NO|
+--------+----------+-----------+----------+-----------+---------+----------+--------+
|   7499 | ALONSO   | VENDEDOR  |    7698 | 1981-02-23 | 1400.00 |   400.00 |   30 |
|   7521 | LOPEZ    | EMPLEADO  |    7782 | 1981-05-08 | 1350.50 |    NULL |   10  |
|   7654 | MARTIN   | VENDEDOR  |    7698 | 1981-09-28 | 1500.00 |  1600.00 |   30 |
|   7698 | GARRIDO  | DIRECTOR  |    7839 | 1981-05-01 | 3850.12 |    NULL |    30 |
|   7782 | MARTINEZ | DIRECTOR  |    7839 | 1981-06-09 | 2450.00 |    NULL |   10   |
|   7839 | REY      | PRESIDENTE |    NULL | 1981-11-17 | 6000.00 |    NULL |    10|
|   7844 | CALVO    | VENDEDOR  |    7698 | 1981-09-08 | 1800.00 |    0.00 |    30|
|   7876 | GIL      | ANALISTA  |    7782 | 1982-05-06 | 3350.00 |    NULL |    20|
|   7900 | JIMENEZ  | EMPLEADO  |    7782 | 1983-03-24 | 1400.00 |    NULL |    20 |
+--------+----------+-----------+----------+-----------+---------+----------+--------+
9 rows in set (0.03 sec)
```

2. Obtener los números de empleados, los apellidos y el número de departamento de todos los empleados de la tabla empleados.

```
mysql> SELECT emp_no, apellido, dep_no
    ->     FROM empleados;
+--------+----------+--------+
| emp_no | apellido | dep_no|
+--------+----------+--------+
|   7499 | ALONSO   |    30 |
|   7521 | LOPEZ    |    10 |
|   7654 | MARTIN   |    30 |
|   7698 | GARRIDO  |    30 |
```

```
|      7782 |  MARTINEZ  |        10 |
|      7839 |  REY       |        10 |
|      7844 |  CALVO     |        30 |
|      7876 |  GIL       |        20 |
|      7900 |  JIMENEZ   |        20 |
+--------+----------+--------+
9 rows in set (0.00 sec)
```

3. Obtener los departamentos diferentes que hay en la tabla

```
empleados mysql> SELECT DISTINCT dep_no
    ->    FROM empleados;
```

```
+--------+
| dep_no |
+--------+
|     10 |
|     20 |
|     30 |
+--------+
```

3- Obtener los diferentes oficios que hay en cada departamento de la tabla

```
empleados mysql> SELECT DISTINCT oficio, dep_no
    -> FROM empleados;
```

```
+------------+--------+
| oficio     | dep_no |
+------------+--------+
| VENDEDOR   |     30 |
| EMPLEADO   |     10 |
| DIRECTOR   |     30 |
| DIRECTOR   |     10 |
| PRESIDENTE |     10 |
| ANALISTA   |     20 |
| EMPLEADO   |     20 |
+------------+--------+
```

2.4 - Utilización de alias

```
SELECT [ALL/DISTINCT] ExpresionColumna [AS] AliasColumna]
                      [, ExpresionColumna [AS] AliasColumna]....]
FROM NombreTabla [AliasTabla] ;
```

Notación: los alias de columna y de tabla aparecen entre corchetes porque son opcionales.

donde ***AliasTabla.....*** SQL permite asignar otro nombre a la misma tabla, dentro de la misma consulta.

AliasColumna..... se escribe detrás de la expresión de columna, separado de ella al menos

por un espacio. Puede ir entre comillas dobles o sin comillas (sino lleva espacios en blanco y sí solo es una palabra)
La cláusula AS que asigna el alias puede omitirse

Usar alias para una tabla es opcional cuando su finalidad consiste en simplificar su nombre original, y obligatorio en consultas cuya sintaxis lo requiera (más adelante lo utilizaremos)

Usar alias para las columnas puede ser necesario porque los títulos o cabeceras que muestra la salida de una consulta para las columnas seleccionadas, se corresponden con los nombres de las columnas de las tablas o las expresiones y esto no siempre es muy visual. Para mejorar su legibilidad y estética se utilizan los alias de columna. También puede ser utilizado, en algunos casos, para renombrar la columna y utilizar este nombre posteriormente

2.5 – Ejemplos de utilización de alias

Ejemplos de alias de tabla

1 - Mostrar el apellido y la fecha de alta de todos los empleados.

```
mysql> SELECT apellido, fecha_alta
    ->     FROM empleados emp;

+----------+------------+
| apellido | fecha_alta |
+----------+------------+
| ALONSO   | 1981-02-23 |
| LOPEZ    | 1981-05-08 |
| MARTIN   | 1981-09-28 |
| GARRIDO  | 1981-05-01 |
| MARTINEZ | 1981-06-09 |
| REY      | 1981-11-17 |
| CALVO    | 1981-09-08 |
| GIL      | 1982-05-06 |
| JIMENEZ  | 1983-03-24 |
+----------+------------+
9 rows in set (0.02 sec)
```

Nota: de momento no podemos ver la utilidad del alias de tabla, pero más adelante veremos su necesidad.

Ejemplos de alias de columna.

1 - Obtener el salario total anual (14 pagas) de los empleados de la empresa mostrando el mensaje Salario total
Vemos, en este ejemplo, las diferentes posibilidades para escribir el alias.

a) Con la cláusula AS

```
mysql> SELECT salario*14        AS Salario_total
FROM empleados;

+--------------+
| Salario_total |
+--------------+
```

```
|          18907.00 |
|          19600.00 |
|          19600.00 |
|          21000.00 |
|          25200.00 |
|          34300.00 |
|          46900.00 |
|          53901.68 |
|          84000.00 |
+---------------+
9 rows in set (0.00 sec)
```

b) Con comillas dobles (admite el carácter blanco en el alias)

```
mysql> SELECT salario*14          "Salario total"
    ->     FROM empleados;
```

```
+----------------+
| Salario total |
+---------------+
|          19600.00 |
|          18907.00 |
|          21000.00 |
|          53901.68 |
|          34300.00 |
|          84000.00 |
|          25200.00 |
|          46900.00 |
|          19600.00 |
+---------------+
9 rows in set (0.00 sec)
```

c) Con comillas simples (admite el carácter blanco en el alias)

```
mysql> SELECT salario*14          'Salario total'
    ->     FROM empleados;
+---------------+
| Salario total |
+---------------+
|          19600.00 |
|          18907.00 |
|          21000.00 |
|          53901.68 |
|          34300.00 |
|          84000.00 |
|          25200.00 |
|          46900.00 |
|          19600.00 |
+---------------+
9 rows in set (0.00 sec)
```

2 - Mostrar el número de empleado, el apellido y el departamento de los empleados de la empresa.

```
mysql> SELECT emp_no "Nº_Empleado", apellido"Apellido", dep_no
                                    "Departamento"
    ->     FROM empleados;
```

```
+-------------+----------+--------------+
| Nº_Empleado | Apellido | Departamento|
+-------------+----------+--------------+
|        7499 | ALONSO   |           30 |
|        7521 | LOPEZ    |           10 |
|        7654 | MARTIN   |           30 |
|        7698 | GARRIDO  |           30 |
|        7782 | MARTINEZ |           10 |
|        7839 | REY      |           10 |
|        7844 | CALVO    |           30 |
|        7876 | GIL      |           20 |
|        7900 | JIMENEZ  |           20 |
+-------------+----------+--------------+
9 rows in set (0.00 sec)
```

3 - Condiciones de selección.

Para seleccionar las filas de la tabla sobre las que realizar una consulta, la cláusula WHERE permite incorporar una condición de selección a la sentencia SELECT.
Muestra todas aquellas filas para las que el resultado de evaluar la condición de selección es VERDADERO

3.1 – Formato de selección con consultas

Formato de consulta con condición de selección

SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna*....]
FROM *NombreTabla*
[WHERE *CondicionSeleccion*];

Notación: la cláusula WHERE es opcional, por eso aparece toda ella entre corchetes

donde ***CondicionSeleccion.....*** es una expresión (conjunto de nombres de columnas, literales, operadores, funciones y paréntesis) cuyo resultado es *VERDADERO/FALSO/NULO*

El funcionamiento es el siguiente: para cada fila se evalúa la condición de selección y si el resultado es VERDADERO se visualizan las expresiones indicadas.

3.2 – Ejemplos de condiciones de selección

1. Seleccionar aquellos empleados cuyo apellido empiece por 'M' y tengan un salario entre 1000 y 200 euros.

```
mysql> SELECT emp_no "Nº Empleado", apellido "Apellido", dep_no “Departamento”
    -> FROM empleados
    -> WHERE apellido LIKE 'M%' AND salario BETWEEN 1000 AND 2000;
```

```
+-------------+----------+--------------+
| Nº Empleado | Apellido | Departamento|
+-------------+----------+--------------+
|              7654 | MARTIN          |                  30 |
+-------------+----------+--------------+
1 row in set (0.01 sec)
```

El operador LIKE usado con '%'indica que puede sustituirse por cualquier grupo de caracteres

2. Seleccionar aquellos empleados cuyo apellido incluya una 'A' en el segundocarácter.

```
mysql> SELECT emp_no "Nº Empleado", apellido "Apellido", dep_no
"Departamento"

    -> FROM empleados
      -> WHERE (apellido LIKE '_A%') ;

+-------------+----------+--------------+
| Nº Empleado | Apellido | Departamento|
+-------------+----------+--------------+
|              7654 | MARTIN          |                  30 |
|              7698 | GARRIDO         |                  30 |
|              7782 | MARTINEZ |                       10 |
|              7844 | CALVO           |                  30 |
+-------------+----------+--------------+
4   rows in set (0.00 sec)
```

El operador LIKE usado con '_' indica que ocupa la posición de un carácter.

3. Seleccionar los empleados existentes en los departamentos 10 y30.

```
mysql> SELECT emp_no "Nº Empleado", apellido "apellido", dep_no
"Departamento"
    -> FROM empleados
     -> WHERE dep_no=10 OR dep_no=30;
+-------------+----------+--------------+
| Nº Empleado | apellido | Departamento|
+-------------+----------+--------------+
|              7499  |  ALONSO      |                  30  |
|              7521  |  LOPEZ       |                  10  |
|              7654  |  MARTIN      |                  30  |
|              7698  |  GARRIDO     |                  30  |
|              7782  |  MARTINEZ    |                  10  |
|              7839  |  REY         |                  10  |
|              7844  |  CALVO       |                  30  |
+-------------+----------+--------------+
7 rows in set (0.02 sec)
```

También puede hacerse utilizando el operador IN: El operador IN comprueba si una determinada expresión toma alguno de los valores indicados entre paréntesis.

```
mysql> SELECT emp_no "Nº EMPLEADO", apellido, dep_no Departamento
    ->  FROM empleados
    ->  WHERE dep_no IN(10,30);
```

4. Seleccionar los empleados que tienen de oficio ANALISTA

```
mysql> SELECT emp_no, apellido, oficio
    -> FROM empleados
    -> WHERE oficio = 'ANALISTA';
+--------+----------+----------+
| emp_no | apellido| oficio                    |
+--------+----------+----------+
|      7876 | GIL                | ANALISTA |
+--------+----------+----------+
1 row in set (0.00 sec)
```

Aunque el campo oficio está grabado en mayúsculas obtenemos el mismo resultado si lo escribimos en minúsculas (MySQL no diferencia entre ambas)

```
mysql> SELECT emp_no, apellido, oficio
    -> FROM empleados
    -> WHERE oficio = 'analista';
+--------+----------+----------+
| emp_no | apellido| oficio                    |
+--------+----------+----------+
|      7876 | GIL                | ANALISTA |
+--------+----------+----------+
1 row in set (0.00 sec)
```

NOTA: MySql no diferencia mayúsculas de minúsculas pero otros gestores de bases de datos sí. Por lo que si se trabaja con otro gestor debe tenerse en cuenta esa posibilidad a la hora de escribir las sentencias.

4 – Funciones predefinidas en expresiones y condiciones

Hemos dicho que una expresión es un conjunto de nombres de columnas, literales o constantes, operadores, funciones y paréntesis, y una condición es una expresión cuyo resultado es *VERDADERO/FALSO/NULO.* Estas expresiones y condiciones nos aparecen en diferentes cláusulas de la sentencia select

Dentro de las expresiones y de las condiciones podemos utilizar las funciones predefinidas enumeradas en el tema 2.

Estas funciones pueden ser utilizadas en todas las expresiones y condiciones con la única restricción determinada por los tipos de datos con los que operan y que devuelven.

Vamos a ver algún ejemplo de cada uno de estos tipos de funciones dentro de las expresiones y las condiciones.

4.1 - Funciones numéricas o aritméticas

Operan con datos numéricos y el resultado es un número.

1 – Visualizar los salarios de los empleados redondeados sin decimales

```
mysql> SELECT apellido, ROUND(salario,0) “SALARIO SIN
DECIMALES”  FROM empleados;
```

```
+----------+-----------------------+
| apellido | SALARIO SIN DECIMALES|
+----------+-----------------------+
| ALONSO        |                          1400 |
| LOPEZ         |                          1350 |
| MARTIN        |                          1500 |
| GARRIDO       |                          3850 |
| MARTINEZ |                               2450 |
| REY           |                          6000 |
| CALVO         |                          1800 |
| GIL           |                          3350 |
| JIMENEZ       |                          1400 |
+----------+-----------------------+
9 rows in set (0.23 sec)
```

2 – Mostrar los datos de los empleados en los que su comisión se múltiplo de 100, y no sea cero.

```
mysql> SELECT *
    -> FROM empleados
    -> WHERE MOD(comision,100)=0 AND comision!=0;
```

```
+-------+----------+---------+--------+------------+---------+---------+------+
|EMP_NO | APELLIDO | OFICIO |DIRECTOR| FECHA_ALTA | SALARIO| COMISION |DEP_NO|
+-------+----------+---------+--------+------------+---------+---------+------+
| 7499 | ALONSO       | VENDEDOR |      7698 | 1981-02-23 | 1400.00 |  400.00 |              30 |
| 7654 | MARTIN       | VENDEDOR |      7698 | 1981-09-28 | 1500.00 |1600.00 |              30 |
+------+----------+----------+--------+------------+---------+---------+------+
2 rows in set (0.00sec)
```

4.2 – Funciones de caracteres

Operan con datos alfanuméricos y el resultado puede ser un dato alfanumérico o un valor numérico. 1 - Visualizar los tres primeros caracteres de los apellidos de los empleados seguidos de un punto

```
mysql> SELECT CONCAT(SUBSTR(apellido,1,3),'.') "INICIALES"
    -> FROM empleados;
```

```
+-----------+
| INICIALES |
+-----------+
| ALO.          |
| LOP.          |
| MAR.          |
| GAR.          |
| MAR.          |
| REY.          |
| CAL.          |
| GIL.          |
| JIM.          |
+-----------+
9 rows in set (0.01 sec)
```

2- Visualizar los nombres de los departamentos cuyo nombre tenga de más de 6 caracteres reemplazando las letras ‘A’ por ‘*’

```
mysql> SELECT REPLACE(dnombre,'A','*')
    -> FROM departamentos
    -> WHERE LENGTH(dnombre)>6;

+--------------------------+
| REPLACE(dnombre,'A','*')|
+--------------------------+
| CONT*BILID*D             |
| INVESTIG*CION            |
| PRODUCCION               |
+--------------------------+
3 rows in set (0.00 sec)
```

4.3 – Funciones de fechas

Son funciones que operan con datos de tipo fecha y pueden devolver diferentes tipos de datos. Son las más complejas y que más diversidad presentan. Su manejo es muy importante dentro de los procesos de gestión que se pueden realizar con las bases de datos.

Nota: estas funciones de fechas varían bastante de un sistema gestor a otro.

1 – Visualizar la fecha que será dentro de una semana

```
mysql> SELECT CURDATE() "HOY",
           ADDDATE(CURDATE(),7) "DENTRO DE UNA SEMANA";

+------------+----------------------+
| HOY        | DENTRO DE UNA SEMANA |
+------------+----------------------+
| 2005-04-02 | 2005-04-09           |
+------------+----------------------+
1 row in set (0.00 sec)
```

2 – Visualizar de alta de los empleados con el formato <día de la semana> - <dia> de <mes> de <año>.

```
mysql> SELECT apellido,CONCAT(DAYNAME(fecha_alta),' - ',
                DAYOFMONTH(fecha_alta),' de ',MONTH(fecha_alta),' de',
                YEAR(fecha_alta)) "fecha alta"
    -> FROM empleados;

+----------+----------------------------+
| apellido | fecha alta                 |
+----------+----------------------------+
| ALONSO   | Monday - 23 de 2 de 1981   |
| LOPEZ    | Friday - 8 de 5 de 1981    |
| MARTIN   | Monday - 28 de 9 de 1981   |
| GARRIDO  | Friday - 1 de 5 de 1981    |
| MARTINEZ | Tuesday - 9 de 6de 1981    |
| REY      | Tuesday - 17 de 11 de 1981 |
| CALVO    | Tuesday - 8 de 9 de 1981   |
| GIL      | Thursday - 6 de 5 de 1982  |
```

```
| JIMENEZ          | Thursday - 24 de 3 de 1983 |
+----------+----------------------------+
9 rows in set (0.00 sec)
```

3 - Mostrar los datos de los empleados que entraron en la empresa en lunes

```
mysql> SELECT *
    -> FROM empleados
      -> WHERE DAYOFWEEK(fecha_alta)=2;

+------+----------+----------+--------+------------+---------+---------+------+
|EMP_NO| APELLIDO | OFICIO           |DIRECTOR| FECHA_ALTA | SALARIO | COMISION|DEP_NO|
+------+----------+----------+--------+------------+---------+---------+------+
| 7499 | ALONSO   | VENDEDOR |      7698 | 1981-02-23 | 1400.00 |  400.00 |  30|
| 7654 | MARTIN   | VENDEDOR |      7698 | 1981-09-28 | 1500.00 | 1600.00 |  30|
+------+----------+----------+--------+------------+---------+---------+-----+
2 rows in set (0.00 sec)
```

4 – Mostrar para cada empleado su apellido junto con el número de trienos que tiene (se tiene un trienio por cada tres años en la empresa)

```
mysql> SELECT APELLIDO, TRUNCATE(((DATEDIFF(CURDATE(),fecha_alta)/365)/3),0)
"TRIENIOS"
    -> FROM empleados;

+----------+----------+
| APELLIDO | TRIENIOS|
+----------+----------+
| ALONSO   |        8 |
| LOPEZ    |        7 |
| MARTIN   |        7 |
| GARRIDO  |        7 |
| MARTINEZ |        7 |
| REY      |        7 |
| CALVO    |        7 |
| GIL      |        7 |
| JIMENEZ  |        7 |
+----------+----------+
9 rows in set (0.00 sec)
```

5 – Mostrar los empleados que llevan más de 23 años en la empresa.

```
mysql> SELECT *
    -> FROM empleados
    -> WHERE fecha_alta<DATE_SUB(CURDATE(),INTERVAL 23 YEAR);

+------+----------+----------+--------+------------+---------+----------+----+
|EMP_NO| APELLIDO | OFICIO           |DIRECTOR| FECHA_ALTA | SALARIO | COMISION |DEP |
+------+----------+----------+--------+------------+---------+----------+----+
| 7499 | ALONSO   | VENDEDOR |      7698 | 1981-02-23 | 1400.00 |   400.00 | 30|
| 7521 | LOPEZ    | EMPLEADO |      7782 | 1981-05-08 | 1350.50 |     NULL | 10|
| 7654 | MARTIN   | VENDEDOR |      7698 | 1981-09-28 | 1500.00 | 1600.00 | 30|
| 7698 | GARRIDO  | DIRECTOR |      7839 | 1981-05-01 | 3850.12 |     NULL | 30|
| 7782 | MARTINEZ | DIRECTOR |      7839 | 1981-06-09 | 2450.00 |     NULL | 10|
| 7839 | REY      | PRESIDENTE|     NULL | 1981-11-17 | 6000.00 |     NULL | 10|
| 7844 | CALVO    | VENDEDOR |      7698 | 1981-09-08 | 1800.00 |     0.00 | 30|
+------+----------+----------+--------+------------+---------+----------+----+
7 rows in set (0.00sec)
```

6 - Visualizar la fecha de 4/10/1997 con el formato <día de la semana>, <número de día> de <nombre del mes> de <año>

```
mysql> SELECT DATE_FORMAT('1997-10-04','%W, %e de %M de %Y') ;
+-----------------------------------------------+
| DATE_FORMAT('1997-10-04','%W, %e de %M de %Y')|
+-----------------------------------------------+
| Saturday, 4 de October de 1997                |
+-----------------------------------------------+
1 row in set (0.00 sec)
```

4.4 – Funciones de comparación

Comparan un valor con otro dando y el resultado obtenido dependerá de la función

concreta. 1 - Visualizar para cada empleado el valor que sea mayor entre su salario y su

comisión

```
mysql> SELECT apellido, salario,comision,
           GREATEST(salario,comision)
       -> FROM empleados;

+----------+---------+----------+----------------------------+
| apellido | salario | comision | GREATEST(salario,comision) |
+----------+---------+----------+----------------------------+
| ALONSO   | 1400.00 |   400.00 |                    1400.00 |
| LOPEZ    | 1350.50 |     NULL |                    1350.50 |
| MARTIN   | 1500.00 |  1600.00 |                    1600.00 |
| 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)
```

2 – Mostar los empleados en los que la suma de su salario más su comisión es menor de 2.000 euros

```
mysql> SELECT apellido, salario, comision
       -> FROM empleados
       -> WHERE salario+IFNULL(comision,0)<2000;

+----------+---------+----------+
| apellido | salario | comision|
+----------+---------+----------+
| ALONSO   | 1400.00 |   400.00 |
| LOPEZ    | 1350.50 |     NULL |
| CALVO    | 1800.00 |     0.00 |
| JIMENEZ  | 1400.00 |     NULL |
+----------+---------+----------+
4 rows in set (0.00 sec)
```

4.5 – Otras funciones

Son funciones que nos dan información sobre la base de datos o realizan algunas operaciones con valores o listas de valores de las filas de la tabla.

1- Indicar la versión de MyQL que estamos utilizando

```
mysql> SELECT VERSION();

+--------------------+
| VERSION()          |
+--------------------+
| 5.1.0-alpha-nt-max |
+--------------------+
1 row in set (0.00 sec)
```

2 – Indicar el usuario con el que estamos conectados

```
mysql> SELECT USER();

+----------------+
| USER()         |
+----------------+
| ODBC@localhost |
+----------------+
1 row in set (0.00 sec)
```

5 - Ordenación.

Para obtener la salida de una consulta clasificada por algún criterio o especificación, la sentencia SELECT
dispone de la cláusula ORDER BY para ordenar.

5.1 – Formato de selección con ordenación

Formato de consulta con ordenación

SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna*....]
FROM *NombreTabla*
[WHERE *CondicionSeleccion*]
[ORDER BY *{ExpresionColumna|Posicion}* [ASC|DESC]
[,*{ExpresionColumna|Posicion}* [ASC|DESC].....]] ;

Notación: la cláusula ORDER BY es opcional, por eso aparece toda ella entre corchetes. Dentro de ella expresión de columna y posición van entre llaves porque hay que elegir una de ellas y ASC y DESC entre corchetes porque son opcionales

donde **ASC|DESC.....** ASC (ascendente) o DESC (descendente) indica la forma de ordenación para esa expresión. Por omisión es ASC.

ExpresionColumna..... conjunto de nombres de columna con literales, operadores y/o funciones. También admite alias.

Posicion..... si queremos ordenar por expresiones que se muestran en el select la ExpresionColumna puede ser sustituida por el número**, Posicion,** que corresponde al número de orden que ocupa en la lista de expresiones

visualizadas en la select.

Si existe más de una expresión por la que ordenar estas aparecen separadas por comas y el orden en que se realizan las clasificaciones es de izquierda a derecha, es decir, a igualdad de la expresión más a la izquierda ordena por la siguiente expresión y así sucesivamente.

5.2 – Ejemplos deordenación

1. Obtener relación alfabética de todos los empleados con todos sus datos.

```
mysql> SELECT dep_no, apellido, salario
    -> FROM empleados
     -> ORDER BY apellido;
```

```
+--------+----------+---------+
| dep_no | apellido | salario|
+--------+----------+---------+
|     30 | ALONSO   |  1400.00 |
|     30 | CALVO    |  1800.00 |
|     30 | GARRIDO  |  3850.12 |
|     20 | GIL      |  3350.00 |
|     20 | JIMENEZ  |  1400.00 |
|     10 | LOPEZ    |  1350.50 |
|     30 | MARTIN   |  1500.00 |
|     10 | MARTINEZ |  2450.00 |
|     10 | REY      |  6000.00 |
+--------+----------+---------+
9 rows in set (0.00 sec)
```

2. Obtener clasificación alfabética de empleados por departamentos.

```
mysql> SELECT dep_no, apellido, salario
    -> FROM empleados
    -> ORDER BY dep_no, apellido;
+--------+----------+---------+
| dep_no | apellido | salario|
+--------+----------+---------+
|     10 | LOPEZ    |  1350.50 |
|     10 | MARTINEZ |  2450.00 |
|     10 | REY      |  6000.00 |
|     20 | GIL      |  3350.00 |
|     20 | JIMENEZ  |  1400.00 |
|     30 | ALONSO   |  1400.00 |
|     30 | CALVO    |  1800.00 |
|     30 | GARRIDO  |  3850.12 |
|     30 | MARTIN   |  1500.00 |
+--------+----------+---------+
```

O también podemos escribir:

```
mysql> SELECT dep_no, apellido, salario
    -> FROM empleados
    -> ORDER BY 1,2;
```

3. Obtener los datos de los empleados clasificados por oficios y en orden descendente de salarios.

```
mysql> SELECT emp_no, apellido, oficio, salario
    ->     FROM empleados
->     ORDER BY oficio, salario DESC;
+--------+----------+------------+---------+
| emp_no | apellido | oficio                | salario |
+--------+----------+------------+---------+
|   7876 | GIL      | ANALISTA   |  3350.00 |
|   7698 | GARRIDO  | DIRECTOR   |  3850.12 |
|   7782 | MARTINEZ | DIRECTOR   |  2450.00 |
|   7900 | JIMENEZ  | EMPLEADO   |  1400.00 |
|   7521 | LOPEZ    | EMPLEADO   |  1350.50 |
|   7839 | REY      | PRESIDENTE |  6000.00 |
|   7844 | CALVO    | VENDEDOR   |  1800.00 |
|   7654 | MARTIN   | VENDEDOR   |  1500.00 |
|   7499 | ALONSO   | VENDEDOR   |  1400.00 |
+--------+----------+------------+---------+
```

4 – Obtener los apellidos de los empleados junto con su salario anual (salario + comision en 14 pagas) ordenado de mayor a menor por este salario total.

```
mysql> SELECT apellido,
            (salario+IFNULL(comision,0))*14 "Salario anual"
    -> FROM empleados
    -> ORDER BY (salario+IFNULL(comision,0))*14 DESC;
+----------+---------------+
| apellido | Salario anual |
+----------+---------------+
| REY      |      84000.00 |
| GARRIDO  |      53901.68 |
| GIL      |      46900.00 |
| MARTIN   |      43400.00 |
| MARTINEZ |      34300.00 |
| ALONSO   |      25200.00 |
| CALVO    |      25200.00 |
| JIMENEZ  |      19600.00 |
| LOPEZ    |      18907.00 |
+----------+---------------+
```

Si no queremos volver a escribir la expresión podemos utilizar el alias o la posición de la expresión por la que queremos ordenar

```
mysql> SELECT apellido,
            (salario+IFNULL(comision,0))*14 "Salario anual"
    -> FROM empleados
    -> ORDER BY "salario anual" DESC;

mysql> SELECT apellido,
            (salario+IFNULL(comision,0))*14 "Salario anual"
    -> FROM empleados
      -> ORDER BY 2 DESC;
```

6 - Selección con limitación de filas

Nos va a permitir limitar el número de filas que se visualicen como resultado de una sentencia select.

6.1 – Formato de selección con limitación de filas

Formato de consulta con limitación de las filas que se visualizan dentro de las filas seleccionadas con el WHERE.

```
SELECT [ALL/DISTINCT] ExpresionColumna [,ExpresionColumna....]
FROM NombreTabla
[WHERE CondicionSeleccion]
[ORDER BY {ExpresionColumna|Posicion} [ASC|DESC]
        [,{ExpresionColumna|Posicion} [ASC|DESC].....] ]
[ LIMIT [m, ] n ] ;
```

Notación: la cláusula LIMIT es opcional, por eso aparece toda ella entre corchetes. Dentro de ella también lo es indicar el valor de m

donde **m**.....es el número de fila por el que se comienza la visualización. Las filas se empiezan a numerar por 0.
Es opcional y en caso de omitirse se supone el valor 0 (1ª fila)
n..... indica el número de filas que se quieren visualizar.

6.2 – Ejemplos de limitación de filas

1. Obtener los datos de los 5 empleados con menos salario.

```
mysql> SELECT emp_no, apellido, salario, dep_no
    -> FROM empleados
    -> ORDER BY salario
    -> LIMIT 5;
+--------+----------+---------+--------+
| emp_no | apellido | salario | dep_no|
+--------+----------+---------+--------+
|   7521 | LOPEZ    |  1350.50 |     10 |
|   7499 | ALONSO   |  1400.00 |     30 |
|   7900 | JIMENEZ  |  1400.00 |     20 |
|   7654 | MARTIN   |  1500.00 |     30 |
|   7844 | CALVO    |  1800.00 |     30 |
+--------+----------+---------+--------+
5 rows in set (0.00 sec)
```

2. Obtener clasificación alfabética de empleados según su apellido y mostrar desde el 5º hasta el 7º de la lista

```
mysql> SELECT emp_no, apellido, salario, dep_no
    -> FROM empleados
    -> ORDER BY apellido
    -> LIMIT 4,3;

+--------+----------+---------+--------+
| emp_no | apellido | salario | dep_no|
+--------+----------+---------+--------+
|   7900 | JIMENEZ  | 1400.00 |     20 |
|   7521 | LOPEZ    | 1350.50 |     10 |
|   7654 | MARTIN   | 1500.00 |     30 |
+--------+----------+---------+--------+
5 rows in set (0.00 sec)
```

Si observamos la salida producida al ordenar por apellido comprobamos que se han visualizado

3 filas desde la 5ª (Fila 4 empezando por 0)

```
mysql> SELECT emp_no, apellido, salario, dep_no
    -> FROM empleados
    -> ORDER BY apellido
    -> ;
+--------+----------+---------+--------+
| emp_no | apellido | salario | dep_no|
+--------+----------+---------+--------+
|   7499 | ALONSO   | 1400.00 |     30 |Fila  0
|   7844 | CALVO    | 1800.00 |     30 |Fila  1
|   7698 | GARRIDO  | 3850.12 |     30 |Fila  2
|   7876 | GIL      | 3350.00 |     20 |Fila  3
|   7900 | JIMENEZ  | 1400.00 |     20 |Fila  4
|   7521 | LOPEZ    | 1350.50 |     10 |Fila  5
|   7654 | MARTIN   | 1500.00 |     30 |Fila  6
|   7782 | MARTINEZ | 2450.00 |     10 |Fila  7
|   7839 | REY      | 6000.00 |     10 |Fila  8
+--------+----------+---------+--------+
9 rows in set (0.00 sec)
```

7 - Resumen del formato de selección

Formato de selección con todas las cláusulas vistas hasta el momento.

```
SELECT { * | [ALL/DISTINCT] ExpresionColumna [[AS] AliasColumna]
            [, ExpresionColumna [[AS] AliasColumna]....] }
FROM NombreTabla [AliasTabla]
[WHERE CondicionSeleccion]
[ORDER BY {ExpresionColumna|Posicion} [ASC|DESC]
            [,{ExpresionColumna|Posicion} [ASC|DESC].....] ]
[LIMIT [m, ] n ] ;
```

TEMA 6. CONSULTAS CON AGRUPAMIENTO Y FUNCIONES DE GRUPOS

1- Consultas de selección con agrupamientos

SQL permite agrupar las filas de una tabla, seleccionadas en una consulta formando grupos según el contenido de alguna o algunas expresiones, y obtener salidas, calculadas a partir de los grupos formados.

Las salidas obtenidas son los resultados de agrupar y aplicar las funciones a cada uno de los grupos de las filas seleccionada en la tabla.

1.1 – Formato de la consulta de selección con agrupamiento

Añadimos a las cláusulas que ya conocemos las de agrupamiento y selección de grupos.

```
SELECT { ExpresionColumna [,ExpresionColumna .....] | * }
FROM NombreTabla
[WHERE CondicionSeleccion]
[GROUP BY ExpresionColumnaAgrupacion|Posición
                [,ExpresionColumnaAgrupacion|Posicion... ]
      [HAVING CondicionSeleccionGrupos ] ]
[ORDER BY {ExpresionColumna|Posicion| } [ASC|DESC]
            [,{ExpresionColumna|Posicion| } [ASC|DESC].....] ]
[LIMIT [m, ] n ] ;
```

Notación: la cláusula GROUP BY es opcional, por eso aparece toda ella entre corchetes y en caso de existir debe ir antes de la cláusula ORDER BY. En caso de existir la cláusula GROUP BY dentro de ella puede estar la cláusula HAVING, que es a su vez opcional.

Donde ***ExpresionColumna.....*** es una expresión que contiene columnas de la tabla, literales,
operadores y/o funciones. Además admite alias. Esta expresión solo puede contener columnas de agrupación y/o funciones de grupo.

ExpresionColumnaAgrupacion.... es una expresión que contiene columnas de la tabla, literales,
operadores y/o funciones por la que se formarán los grupos. No admite alias.

Posicion ..... posición que ocupa le expresión por la que queremos
agrupar en la lista de expresiones visualizadas.

CondicionSelecionGrupos…….. *e*s una condición, expresión que devuelve el valor *verdadero*
/falso/nulo, para seleccionar grupos

Si existe más de una expresión por la que agrupar, estas aparecen separadas por comas y el orden en que se realizan las agrupaciones es de izquierda a derecha. Se forman grupos con la expresión más a la izquierda y a igualdad de la expresión más a la izquierda se agrupa por la siguiente expresión y así sucesivamente.

La cláusula HAVING se emplea para controlar qué grupos se seleccionan, una vez realizada la agrupación. Esta asociada a la cláusula GROUP BYy no tiene sentido sin ella.

La salida después de utilizar la cláusula GROUP BY queda ordenada por las expresiones de agrupación. Las cláusulas ORDER BY y LIMIT pueden utilizarse después de agrupar, si se quiere agrupar por otro concepto y/o limitar el número de filas. Solo existe una limitación en la cláusula ORDER BY, ya que no puede contener funciones de grupo. En caso de que quiera ordenarse por

una de estas funciones deberemos hacerlo referenciando la posición que ocupa en la select.

Ni en la expresión de la cláusula GROUP BY ni en condición de la cláusula HAVING pueden utilizarse los alias.

1.2 – Consideraciones de consultas con cláusulas de agrupamiento

El funcionamiento de la sentencia SELECT con cláusulas de agrupamiento es el siguiente:

- primero realiza una selección de filas según la cláusula WHERE
- forma grupos según la cláusula GROUP BY
- hace una selección de grupos según la cláusula HAVING.

Es importante tener en cuenta que, en caso de selección con cláusula de agrupación, **solo pueden mostrarse expresiones que contengan columnas de agrupación y/o funciones de grupo**. Así mimo, si se utilizan funciones de agrupación sin la cláusula GROUP BY no pueden mostrarse el resto de las filas de la tabla.

La cláusula **HAVING actúa como un filtro sobre el resultado de agrupar las filas,** a diferencia de la cláusula WHERE que actúa sobre las filas antes de la agrupación.

2- Funciones de grupo

Estas funciones de grupo actúan sobre las filas previamente seleccionadas y los grupos que se hayan formado en ellas. El criterio para agrupar suele ser una o varias de las columnas o expresiones de la tabla llamadas columnas o expresiones de agrupación. Si no se especifica ningún criterio, las filas de la tabla seleccionadas en la consulta, formarán un grupo único.

2.1- Funciones de grupo

AVG(expr)	Valor medio de “expr” ignorando los valores nulos
COUNT({ * | expr })	Número de veces que “expr” tiene un valor no nulo. La opción “*” cuenta el número de filas seleccionadas.
MAX(expr)	Valor máximo de “expr”
MIN(expr)	Valor mínimo de “expr”
STDDEV(expr)	Desviación típica de “expr” sin tener en cuenta los valores nulos
SUM(expr)	Suma de “expr”
VARIANCE(expr)	Varianza de “expr” sin tener en cuenta los valores nulos.

Descripción de las funciones de grupo

AVG(expr) Calcula el valor medio de la expresión de columna que se indique dentro del paréntesis, teniendo en cuenta que los valores NULL no son incluidos.

COUNT({ * | expr }) Tiene dos posibilidades, la primera con un * cuenta el número filas seleccionadas y la segunda con una expresión de columna cuenta el número de veces que la expresión tiene el valor diferente de NULL

MAX(expr) Devuelve el valor máximo de la expresión de columna que le acompaña.

MIN(expr) Devuelve el valor mínimo de la expresión de columna que le acompaña.

STDDEV(expr) Calcula la desviación típica para los valores de la expresión de columna que le acompaña.

SUM(expr) Calcula la suma de valores de la expresión de columna indicada dentro del paréntesis

VARIANCE(expr) Calcula la varianza para los valores de la expresión de columna que se indique dentro del paréntesis, teniendo en cuenta que los valores NULL no son incluidos.

Por lo general, las funciones de grupos se utilizan sobre más de un grupo de filas. La cláusula GROUP BY establece el criterio o columnas de agrupación y se calculará el valor de la función para cada grupo. Pero también pueden utilizarse sin la cláusula GROUP BY y en ese caso estas funciones actúan sobre un único grupo formado por todas las filas seleccionadas.

Estas funciones pueden ser utilizadas con la cláusula DISTICT.
Por ejemplo COUNT(DISTICT(NombreColumna) cuenta cuantos valores diferentes para esa columna hay entre las filas seleccionadas o agrupadas.

2. 2 – Ejemplos con funciones de grupo.

a) Ejemplos de consulta con funciones de grupo sin criterio de agrupación

1. Obtener la masa salarial mensual de todos los empleados.

```
mysql> SELECT SUM(salario)
    ->     FROM empleados;

+--------------+
| SUM(salario) |
+--------------+
|     23100.62 |
+--------------+
1 row in set (0.00 sec)
```

2. Obtener los salarios máximo, mínimo y la diferencia existente entre ambos.

```
mysql> SELECT MAX(salario)"Salario mas alto" ,
    ->        MIN(salario) "Salario mas bajo",
    ->        MAX(salario)- MIN(salario) "Diferencia"
    -> FROM empleados;

+------------------+------------------+------------+
| Salario mas alto | Salario mas bajo | Diferencia|
+------------------+------------------+------------+
|          6000.00 |          1350.50 |    4649.50 |
+------------------+------------------+------------+
1 row in set (0.00 sec)
```

3. Obtener la fecha de alta más reciente.

```
mysql> SELECT MAX(fecha_alta)"Fecha alta"
    -> FROM empleados;

+------------+
| Fecha alta |
+------------+
| 1983-03-24 |
+------------+
1 row in set (0.00 sec)
```

4. Calcular el salario medio de los empleados.

```
mysql> SELECT AVG(salario) "Salario medio"
    -> FROM empleados;

+---------------+
| Salario medio |
+---------------+
|   2566.735569 |
+---------------+
1 row in set (0.00 sec)
```

A veces hacer la media con la función AVG no da el mismo resultado que hacer la suma y dividirla por el número de filas, SUM/CONT. Esto es porque COUNT cuenta el número de valores de datos que hay en una columna, sin incluir los valores NULL, y por el contrario, COUNT(*) cuenta todas las filas de la tabla, sin considerar que en algunas columnas existan valores NULL. Sin embargo la función AVGsi tiene en cuenta las filas con valores NULL.
Veamos un ejemplo:

```
mysql> SELECT AVG(comision)
    -> FROM empleados;
+---------------+
| AVG(comision) |
+---------------+
|     666.666667 |
+---------------+
1 row in set (0.00 sec)

mysql> SELECT SUM(comision)/COUNT(*)
    ->  FROM empleados;
+------------------------+
| SUM(comision)/COUNT(*)|
+------------------------+
|                 222.2222 |
+------------------------+
mysql> SELECT SUM(comision)/COUNT(comision)
    -> FROM empleados;

+--------------------------------+
| SUM(comision)/COUNT(comision)|
+--------------------------------+
|                          666.6667 |
+--------------------------------+
1 row in set (0.00 sec)
```

5. Calcular el salario medio de los empleados que sean ANALISTAS

```
mysql> SELECT AVG(salario) "Salario medio"
    ->     FROM empleados
    ->     WHERE oficio = 'ANALISTA';

+---------------+
| Salario medio |
+---------------+
|       3350.000000 |
+---------------+
1 row in set (0.00 sec)
```

b) Ejemplos de consulta con funciones de grupo con criterio de agrupación GROUPBY

1. Obtener los salarios medios por departamento.

```
mysql> SELECT dep_no "Departamento" , AVG(salario) "Salariomedio"
    ->     FROM empleados
    ->     GROUP BY dep_no;

+--------------+---------------+
| Departamento | Salario medio |
+--------------+---------------+
|                   10 |       3266.833333 |
|                   20 |       2375.000000 |
|                   30 |       2137.530029 |
+--------------+---------------+
3 rows in set (0.00 sec)
```

2. Obtener cuántos empleados hay en cada oficio

```
mysql> SELECT oficio "Oficio", COUNT(*) "Nº de Empleados"
    -> FROM empleados
    -> GROUP BY oficio;

+------------+-----------------+
| Oficio             | Nº de Empleados |
+------------+-----------------+
| ANALISTA           |                       1 |
| DIRECTOR           |                       2 |
| EMPLEADO           |                       2 |
| PRESIDENTE |                       1 |
| VENDEDOR           |                       3 |
+------------+-----------------+
5 rows in set (0.00 sec)
```

c) Ejemplos de consulta con funciones de grupo con criterio de agrupación GROUP BY y cláusula **HAVING**

SQL realiza la selección de grupos en el proceso siguiente:

- A partir de la tabla sobre la que se realiza la consulta, la cláusula WHERE actúa como un primer filtro que da como resultado una tabla interna cuyas filas cumplen la condición especificada en el WHERE.
- La cláusula GROUP BY produce la agrupación de las filas de la segunda tabla, dando como resultado una tercera tabla.

- La cláusula HAVING actúa filtrando las filas de la tercera tabla, según la condición de selección especificada, dando como resultado la salida de la consulta.

1. Seleccionar los oficios que tengan dos o más empleados:

```
mysql> SELECT oficio, COUNT(*)
    -> FROM empleados
    -> GROUP BY oficio
    -> HAVING COUNT(*)>= 2;

+----------+----------+
| oficio   | COUNT(*) |
+----------+----------+
| DIRECTOR |        2 |
| EMPLEADO |        2 |
| VENDEDOR |        3 |
+----------+----------+
3 rows in set (0.00 sec)
```

2. Seleccionar los oficios que tengan dos o más empleados, cuyo salario supere los 1400 euros.

```
mysql> SELECT oficio, COUNT(*)
    -> FROM empleados
    -> WHERE salario > 1400
    -> GROUP BY oficio
    -> HAVING COUNT(*) >= 2;

+----------+----------+
| oficio   | COUNT(*) |
+----------+----------+
| DIRECTOR |        2 |
| VENDEDOR |        2 |
+----------+----------+
2 rows in set (0.00 sec)
```

d) Ejemplos de consulta con funciones de grupo con criterio de agrupación GROUP BY, **incluyendo las cláusulas ORDER BY y LIMIT**

1- Seleccionar los datos del departamento con menor salario medio

```
mysql> SELECT dep_no, AVG (salario)
    -> FROM empleados
    -> GROUP BY dep_no
    -> ORDER BY 2
    -> LIMIT 1;
+--------+---------------+
| dep_no | AVG (salario) |
+--------+---------------+
|     30 |   2137.530029 |
+--------+---------------+
```

2- Seleccionar los datos del departamento con mayor número de empleados.

```
mysql> SELECT dep_no, COUNT(*)
    -> FROM empleados
    -> GROUP BY dep_no
    -> ORDER BY 2 DESC
    -> LIMIT 1;

+--------+----------+
| dep_no | COUNT(*) |
+--------+----------+
|     30 |        4 |
+--------+----------+
1 row in set (0.39 sec)
```

3- Resumen (2) del formato de selección

Formato de selección con todas las cláusulas vistas hasta el momento.

```
SELECT { * | [ALL/DISTINCT] ExpresionColumna [AliasColumna]
                          [,ExpresionColumna [AliasColumna]....] }
FROM NombreTabla [AliasTabla]
[WHERE CondicionSeleccion]
[GROUP BY ExpresionColumna|Posicion [,ExpresionColumna|Posicion ... ]
      [HAVING CondicionSelecciónGrupos ] ]
[ORDER BY {ExpresionColumna|Posicion } [ASC|DESC]
          [,{ExpresionColumna|Posicion } [ASC|DESC].....] ]
[LIMIT [m, ] n ] ;
```

TEMA 7. SUBCONSULTAS

1 - ¿Qué es una subconsulta?.

Una subconsulta en SQL consiste en utilizar los resultados de una consulta dentro de otra, que se considera la principal. Esta posibilidad fue la razón original para la palabra "estructurada" que da el nombre al SQL de Lenguaje de Consultas Estructuradas (Structured Query Language).

Anteriormente hemos utilizado la cláusula WHERE para seleccionar los datos que deseábamos comparando un valor de una columna con una constante, o un grupo de ellas. Si los valores de dichas constantes son desconocidos, normalmente por proceder de la aplicación de funciones a determinadas columnas de la tabla, tendremos que utilizar subconsultas. Por ejemplo, queremos saber la lista de empleados cuyo salario supere el salario medio. En primer lugar, tendríamos que averiguar el importe del salario medio:

```
mysql> SELECT AVG(salario)"Salario Medio"
    ->     FROM empleados;

+---------------+
| Salario Medio |
+---------------+
|     2566.735569 |
+---------------+
1 row in set (0.00 sec)
```

A continuación, anotarlo en un papel o recordarlo para la siguiente sentencia:

```
mysql> SELECT dep_no "Nº Empleado", apellido, salario
    ->     FROM empleados
    ->     WHERE salario> 2566.73;
+-------------+----------+---------+
| Nº Empleado | apellido | salario|
+-------------+----------+---------+
|                  30 | GARRIDO         | 3850.12 |
|                  10 | REY             | 6000.00 |
|                  20 | GIL             | 3350.00 |
+-------------+----------+---------+
3 rows in set (0.00 sec)
```

Pero además de tener que anotar el resultado de otra consulta para ser utilizado en esta, tiene el problema de que si el número de empleados o el salario de estos cambiase, cambiaría el valor del salario medio y habría que modificar esta segunda consulta reemplazando el valor antiguo por el nuevo valor.

Sería mucho más eficiente utilizar una subconsulta:

```
mysql> SELECT dep_no "Nº Empleado", apellido, salario
    ->     FROM empleados
    ->     WHERE salario > (SELECT AVG(salario)
    ->                             FROM empleados);

+-------------+----------+---------+
| Nº Empleado | apellido | salario|
+-------------+----------+---------+
|                  30 | GARRIDO         | 3850.12 |
|                  10 | REY             | 6000.00 |
|                  20 | GIL             | 3350.00 |
+-------------+----------+---------+
3 rows in set (0.00 sec)
```

La subconsulta (comando SELECT entre paréntesis) se ejecuta primero y, posteriormente, el valor extraído es utilizado en la consulta principal, obteniéndose el resultado deseado.

1.1 - Formato de una subconsulta

```
( SELECT [ALL/DISTINCT] ExpresionColumna [,ExpresionColumna .....]
  FROM NombreTabla [ , NombreTabla ]
 [WHERE CondicionSeleccion]
 [GROUP BY ExpresionColumnaAgrupacion [, ExpresionColumnaAgrupacion ... ]
        [HAVING CondicionSeleciónGrupos ] ] )
```

donde el formato de la sentencia SELECT entre paréntesis tiene las siguientes diferencias con la sentencia
SELECT de las consultas:

- No tiene sentido la cláusula ORDER BY ya que los resultados de una subconsulta se utilizan internamente y no son visibles al usuario.
- Los nombres de columna que aparecen las expresiones en una subconsulta pueden referirse a columnas de la tabla de la consulta principal y se conocen como *referencias externas.*

Una subconsulta siempre forma parte de la condición de selección en las cláusulas WHERE o HAVING. Cuando incluimos una subconsulta en una sentencia select el funcionamiento es el siguiente: para cada fila de la consulta ejecuta la subconsulta y con ese resultado se evalúa la fila correspondiente de la consulta, mostrándose si el resultado de la evaluación es VERDADERO.

Las subconsultas habitualmente devuelven una sola expresión pero también pueden devolver más de una. La sentencia select que conecta con la subconsulta deberá recoger estos valores en una o varias columnas, según sea la subconsulta, para poder después compararlos.

Vamos a verlo con unos ejemplos
a) Subconsulta que devuelve una sola expresión
Obtener el nombre del departamento donde trabaja
GARRIDO

```
mysql> SELECT dnombre

    -> FROM departamentos
      -> WHERE dep_no = (SELECT dep_no
    ->                          FROM empleados
    ->                          WHERE apellido = 'GARRIDO');

+---------+
| dnombre |
+---------+
| VENTAS  |
+---------+
1 row in set (0.02 sec)
```

La subconsulta devuelve una sola expresión dep_no, que en este caso el valor del departamento de GARRIDO que la consulta compara con el correspondiente en la tabla departamentos.

b)Subconsulta que devuelven más de una expresión
Obtener los empleados que tengan el mismo oficio y departamento que ALONSO

```
mysql> SELECT emp_no, apellido, oficio, dep_no
    -> FROM empleados
    -> WHERE (dep_no, oficio) = (SELECT dep_no, oficio
    ->                           FROM empleados
    ->                           WHERE apellido = 'ALONSO');

+--------+----------+----------+--------+
| emp_no | apellido | oficio   | dep_no |
+--------+----------+----------+--------+
|   7499 | ALONSO   | VENDEDOR |     30 |
|   7654 | MARTIN   | VENDEDOR |     30 |
|   7844 | CALVO    | VENDEDOR |     30 |
+--------+----------+----------+--------+
3 rows in set (0.02 sec)
```

La subconsulta devuelve dos expresiones, dep_no y oficio (en este caso formadas por una columna cada una) correspondientes al departamento y oficio de ALONSO y la consulta lo compara con dos columnas dep_no y oficio, de cada una de las filas de la tabla.

Lo más habitual es el primer caso por ello comenzaremos con las subconsultas que devuelven una sola expresión y posteriormente, en el apartado siguiente, trataremos las que devuelven más de una expresión.

2 – Subconsultas que devuelven una sola expresión

2.1 - Formato de las subconsultas que devuelven una sola expresión

Es un formato habitual de una sentencia SELECT con la particularidad de que solo debe seleccionarse una expresión de columna.

(SELECT [ALL/DISTINCT] ***ExpresionColumna***
FROM ***NombreTabla [, NombreTabla]***
[WHERE ***CondicionSeleccion*****]**
[GROUP BY ***ExpresionColumnaAgrupacion [,ExpresionColumnaAgrupacion ...*** **]**
[HAVING ***CondicionSelecciónGrupos*** **]])**

2.2 - Valores de retorno y condiciones de selección

Como hemos dicho comenzaremos con la subconsultas que devuelven una sola expresión. El resultado de ejecutar la subconsulta puede devolvernos, en esta expresión, un valor simple o más de un valor. Según el retorno de la subconsulta, el operador de comparación que se utilice en la condición de selección del WHERE o HAVING deberá ser del tipo apropiado según la tabla siguiente:

Retorno de la subconsulta	*Operador comparativo*

Valor simple Más de un valor	De tipo aritmético De tipo lógico

2.2.1 - Condición de selección con operadores aritméticos de comparación

Se utiliza cuando la subconsulta devuelve un único valor a comparar con una expresión, por lo general formada a partir de la fila obtenida en la consulta principal. Si la comparación resulta cierta (TRUE), la condición de selección también lo es. Si la subconsulta no devuelve ninguna fila (NULL), la comparación devuelve también el valor NULL. Si la condición de comparación resulta falsa (FALSE), la condición de selección también lo será.

Formato para la condición de selección con operadores aritméticos de comparación

ExpresionColumna OperadorComparacion (Subconsulta)

donde **OperadorComparacion** puede ser=,<>,<,>,<=,<=

Ejemplos de subconsultas con operadores de comparación

1. Obtener todos los empleados que tienen el mismo oficio que GARRIDO

```
mysql> SELECT emp_no "Nº Empleado", apellido, oficio
    ->    FROM empleados
    ->    WHERE oficio = (SELECT oficio
    ->                    FROM empleados
    ->                    WHERE apellido = 'GARRIDO');

+-------------+----------+----------+
| Nº Empleado | apellido | oficio   |
+-------------+----------+----------+
|        7698 | GARRIDO  | DIRECTOR |
|        7782 | MARTINEZ | DIRECTOR |
+-------------+----------+----------+
2 rows in set (0.00 sec)
```

La subconsulta devuelve el oficio de GARRIDO que es DIRECTOR y se visualizan los trabajadores de oficio DIRECTOR, entre ellos GARRIDO. Más adelante haremos un ejemplo donde no se visualizará el empleado de la subconsulta.

2. Obtener información de los empleados que ganan más que cualquier empleado del departamento 30.

```
mysql> SELECT emp_no "Nº Empleado", apellido,salario, dep_no
           "Nº Departamento"
    ->    FROM empleados
```

```
    -> WHERE salario > (SELECT MAX(salario)
    ->                  FROM empleados
    ->                  WHERE dep_no=30);

+-------------+----------+---------+-----------------+
| Nº Empleado | apellido | salario | Nº Departamento |
+-------------+----------+---------+-----------------+
|        7839 | REY      | 6000.00 |              10 |
+-------------+----------+---------+-----------------+
1 row in set (0.00 sec)
```

3. Visualizar el número de VENDEDORES del departamento VENTAS.

```
mysql> SELECT COUNT(*) "Total Empleados"
    -> FROM empleados
    -> WHERE oficio = 'VENDEDOR'
    ->    AND dep_no = (SELECT dep_no
    ->                  FROM departamentos
    ->                  WHERE dnombre = 'VENTAS')
    -> GROUP BY oficio;
+-----------------+
| Total Empleados |
+-----------------+
|               3 |
+-----------------+
```

4. Visualizar la suma de los salarios para cada oficio de los empleados del departamento

VENTAS.

```
mysql> SELECT oficio, sum(salario)"Suma salarios"
    -> FROM empleados
    -> WHERE dep_no = (SELECT dep_no
    ->                 FROM departamentos
    ->                 WHERE dnombre='VENTAS')
    -> GROUP BY oficio;
+----------+---------------+
| oficio   | Suma salarios |
+----------+---------------+
| DIRECTOR |       3850.12 |
| VENDEDOR |       4700.00 |
+----------+---------------+
2 rows in set (0.01 sec)
```

2.2.3 - Condición de selección con operadores lógicos.

Se utiliza cuando la subconsulta puede devolver más de una fila a comparar con la fila actual de la consulta principal. En ese caso los operadores aritméticos dan error.

Formato para la condición de selección con operadores lógicos

ExpresionColumna OperadorLogico (Subconsulta)

Donde **OperadorLogico** puede ser IN, ANY, ALL Y EXISTS

a) Operador lógico IN

Comprueba si valores de la fila actual de la consulta principal coincide con alguno de la lista de valores devueltos por la subconsulta. Si el resultado es afirmativo la comparación resulta cierta (TRUE).

Formato para la condición de selección con el operador lógico IN

ExpresionColumna [NOT] IN (Subconsulta)

Ejemplos del operador lógico IN

1. Listar, en orden alfabético, aquellos empleados que no trabajen ni en Madrid ni en Barcelona.

```
mysql> SELECT emp_no, apellido, dep_no
    -> FROM empleados
    -> WHERE dep_no IN (SELECT dep_no

    ->                  FROM departamentos
    ->                  WHERE localidad NOT LIKE 'MADRID'
    ->                      AND localidad NOT LIKE 'BARCELONA');

+--------+----------+--------+
| emp_no | apellido | dep_no|
+--------+----------+--------+
|   7876 | GIL      |     20 |
|   7900 | JIMENEZ  |     20 |
+--------+----------+--------+
2 rows in set (0.00 sec)
```

La subconsulta selecciona todos los departamentos que no están en Madrid ni en Barcelona, y la consulta principal comprueba, empleado a empleado, si su departamento es uno de los seleccionados en la subconsulta, visualizando sus datos caso de ser cierto.

También podemos resolverla utilizando NOT IN

```
mysql> SELECT emp_no, apellido, dep_no
    ->     FROM empleados
    ->     WHERE dep_no NOT IN (SELECT dep_no
    ->                          FROM departamentos
    ->                          WHERE localidad LIKE 'MADRID'
    ->                            OR localidad LIKE 'BARCELONA');
```

2. Listar los nombres de los departamentos que tengan algún empleado con fecha de alta anterior a 1982.

```
mysql> SELECT dep_no "NºDepartamento", dnombre "Departamento"
    -> FROM departamentos
    -> WHERE dep_no IN (SELECT DISTINCT(dep_no)
    ->                  FROM empleados
    ->                  WHERE fecha_alta < '1982-01-01');

+----------------+--------------+
| NºDepartamento | Departamento|
+----------------+--------------+
|             10 | CONTABILIDAD |
|             30 | VENTAS       |
+----------------+--------------+
2 rows in set (0.00 sec)
```

En este ejemplo, la subconsulta selecciona todos los empleados cuya fecha de alta sea anterior al año 1982, y la consulta principal compara, departamento a departamento, si coincide con el de alguno de los que hayan sido seleccionados en la subconsulta.

3. Obtener los departamentos y sus nombres, siempre que haya más de dos empleados trabajando en ellos.

```
mysql> SELECT dep_no "NºDepartamento", dnombre "Departamento"
    -> FROM departamentos
    -> WHERE dep_no IN (SELECT dep_no
    ->                  FROM empleados
    ->                  GROUP BY dep_no
    ->                  HAVING COUNT(*)>2 );

+----------------+--------------+
| NºDepartamento | Departamento|
+----------------+--------------+
|             10 | CONTABILIDAD |
|             30 | VENTAS       |
+----------------+--------------+
2 rows in set (0.00 sec)
```

La subconsulta selecciona todos los departamentos que tienen más de un empleado trabajando, y la consulta principal comprueba, departamento a departamento, si es uno de los seleccionados en la subconsulta, visualizando sus datos caso de ser cierto.

b) Operadores lógicos ANY y ALL

Se utilizan junto a los operadores aritméticos de comparación para ampliar las posibles comprobaciones de valores obtenidos a partir de la fila seleccionada en la consulta principal con valores obtenidos en la subconsulta.
Su uso a menudo es sustituido por el del operador IN.

Formato para los operadores ANY/ALL

ExpresionColumna OperadorComparacion {ANY|ALL} (Subconsulta)

donde **OperadorComparacion** puede ser =,<>,<,>,<=,<

El operador ANY con uno de los seis operadores aritméticos compara el valor de la expresión formada a partir de la consulta principal con valores producidos por la subconsulta. Si alguna de las comparaciones individuales produce un resultado verdadero (TRUE), el operador ANY devuelve un resultado verdadero (TRUE).

El operador ALL también se utiliza con los operadores aritméticos para comparar un valor de la expresión formada a partir de la consulta principal con cada uno de los valores de datos producidos por la subconsulta. Si todos los resultados de las comparaciones son ciertos (TRUE), el operador ALL devuelve un valor cierto (TRUE).

Ejemplos de ANY

1. Visualizar los nombres de los departamentos que tengan empleados trabajando en ellos..

```
mysql> SELECT dep_no "Nº Departamento", dnombre Departamento
    -> FROM departamentos
    -> WHERE dep_no = ANY (SELECT dep_no
    ->                     FROM empleados);

+-----------------+---------------+
| Nº Departamento | Departamento  |
+-----------------+---------------+
|              10 | CONTABILIDAD  |
|              20 | INVESTIGACION |
|              30 | VENTAS        |
+-----------------+---------------+
3 rows in set (0.00 sec)
```

Lo mismo con el operador IN:

```
mysql> SELECT dep_no "Nº Departamento", dnombre Departamento
    -> FROM departamentos
    -> WHERE dep_no IN (SELECT dep_no
    ->                  FROM empleados);

+-----------------+---------------+
| Nº Departamento | Departamento  |
+-----------------+---------------+
|              10 | CONTABILIDAD  |
|              20 | INVESTIGACION |
|              30 | VENTAS        |
+-----------------+---------------+
4 rows in set (0.00 sec)
```

2. Seleccionar aquellos departamentos en los que al menos exista un empleado con comisión nula.

```
mysql> SELECT dep_no "Nº Departamento", dnombre Departamento
    ->     FROM departamentos
    ->     WHERE dep_no = ANY (SELECT DISTINCT dep_no
    ->                                 FROM empleados WHERE comision ISNULL);

+-----------------+--------------+
| Nº Departamento | Departamento|
+-----------------+--------------+
|                         30 | VENTAS                  |
+-----------------+--------------+
1  row in set (0.00 sec)
```

Ejemplos del operador ALL

1. Listar los empleados con mayor salario que todos los del departamento 20

```
mysql> SELECT dep_no "Nº Departamento", apellido, salario
    -> FROM empleados
    -> WHERE salario >          ALL (SELECT salario
    ->                                 FROM empleados
    ->                                 WHERE dep_no = 20);

+-----------------+----------+---------+
| Nº Departamento | apellido | salario|
+-----------------+----------+---------+
|                         30 | GARRIDO          | 3850.12 |
|                         10 | REY                | 6000.00 |
+-----------------+----------+---------+
2  rows in set (0.00 sec)
```

2. Listar los departamentos que no tienen empleados

```
mysql> SELECT dep_no "Nº Departamento",
                        dnombre "Nombre departamento"
    -> FROM departamentos
    -> WHERE dep_no <>       ALL (SELECT DISTINCT(dep_no)
    ->                                 FROM empleados);

+-----------------+----------------------+
| Nº Departamento | Nombre departamento|
+-----------------+----------------------+
|                         40 | PRODUCCION                    |
+-----------------+----------------------+
1 row in set (0.00 sec)
```

Como vemos <> ALL es equivalente a NOT IN

c) Operador lógico EXISTS

Se utiliza cuando la condición de selección consiste exclusivamente en comprobar que la subconsulta devuelve alguna fila seleccionada según la condición incluida en la propia subconsulta. El operador EXISTS no necesita que la subconsulta devuelva alguna columna porque

no utiliza ninguna expresión de comparación, justificando así la aceptación del * en el formato de la misma.

Formato para la condición de selección con el operador lógico EXISTS

ExpresiónColumna* [NOT] EXISTS *(Subconsulta)

Una subconsulta expresada con el operador EXISTStambién podrá expresarse con el operador IN. Se utiliza ,sobre todo, con consultas correlacionadas que veremos más adelante.

Ejemplos con el operador lógico EXISTS

1 - Visualizar los departamentos en los que hay más de un trabajador.

```
mysql> SELECT dep_no, dnombre
    -> FROM departamentos e
    -> WHERE EXISTS ( SELECT *
    ->                FROM empleados d
    ->                WHERE e.dep_no = d.dep_no
    ->                GROUP BY dep_no
    ->                HAVING COUNT(*) > 1);
```

```
+--------+--------------+
| dep_no | dnombre      |
+--------+--------------+
|     10 | CONTABILIDAD |
|     30 | VENTAS       |
+--------+--------------+
2 rows in set (0.00 sec)
```

2. Listar las localidades donde existan departamentos con empleados cuya comisión supere el 10% del salario.

```
mysql> SELECT localidad
    ->    FROM departamentos d
    ->    WHERE EXISTS (SELECT *
    ->                  FROM empleados e
    ->                  WHEREcomision>10*salario/100
                              ANDe.dep_no=d.dep_no);
```

```
+-----------+
| localidad |
+-----------+
| MADRID    |
+-----------+
1 row in set (0.00 sec)
```

Nota: las tablas de departamentos y de empleados necesitan llevar alias para poder realizar parte de la condición de selección en la subconsulta ya que en ambas existe una columna con el mismo nombre (dep_no). Esto se verá en el apartado posterior de consultas correlacionadas.

La misma subconsulta podemos expresarla con el operador IN de la siguiente manera:

```
mysql> SELECT localidad
    ->    FROM departamentos
    ->    WHERE dep_no IN (SELECT dep_no
    ->                     FROM empleados
    ->                     WHERE comision>10*salario/100);
+-----------+
| localidad |
+-----------+
| MADRID    |
+-----------+
1 row in set (0.00 sec)
```

3 – Subconsultas que devuelven más de una expresión.

Hemos dicho que las subconsultas habitualmente de devuelven una sola expresión pero también pueden devolver más de una.

Son útiles, sobre todo, cuando la clave ajena de la tabla de la consulta, que se corresponde con una clave primaria de la tabla de la subconsulta, es compuesta.

3.1 – Formato de consultas que devuelven más de una expresión

Es un formato equivalente, pero la select de la subconsulta devuelve más de una expresión.

(SELECT [ALL/DISTINCT] ***ExpresionColumna*** **[,*ExpresionColumna .....]**
FROM ***NombreTabla [, NombreTabla]***
[WHERE ***CondicionSeleccion*****]**
[GROUP BY ***ExpresionColumnaAgrupacion [,ExpresionColumnaAgrupacion…]***
[HAVING ***CondicionSelecionGrupos*** **]])**

3.2 - Valores de retorno y condiciones deselección

La condición de selección debe tener en cuenta el número de valores de retorno de la subconsulta y compararlos con el mismo número de expresiones.

(ExpresionColumna [,ExpresionColumna ....]) OperadorComparacion (Subconsulta)

donde **OperadorComparacion** tiene que ser el operador = o bien el operador IN dependiendo de si devuelve un valor simple o puede devolver varios valores.

El número de **ExpresionColumna** en la consulta será igual al número de ExpresionColumna que nos proporcione el select de la subconsulta

3.3 – Ejemplos de consultas que devuelven más de una expresión

1- Obtener los empleados que pertenecen al mismo departamento y entraron en la empresa el mismo día que GARRIDO

```
mysql> SELECT emp_no, apellido
    -> FROM empleados
    -> WHERE (dep_no,fecha_alta) = (SELECT dep_no, fecha_alta
    ->                              FROM empleados
    ->                              WHERE apellido='GARRIDO');
+--------+----------+
| emp_no | apellido |
+--------+----------+
|   7698 | GARRIDO  |
+--------+----------+
1 row in set (0.00 sec)
```

2 - Obtener el empleado que pertenece al mismo departamento que JIMENEZ y tiene el máximo salario

```
mysql> SELECT emp_no, apellido
    -> FROM empleados
    -> WHERE (dep_no,salario)=(SELECT dep_no, MAX(salario)
    ->                         FROM empleados
    ->                         WHERE dep_no= (SELECT dep_no
    ->                                        FROM empleados
    ->                                        WHERE apellido='JIMENEZ')
    ->                         GROUP BY dep_no);
+--------+----------+
| emp_no | apellido |
+--------+----------+
|   7876 | GIL      |
+--------+----------+
```

3 - Listar el empleado que tiene el mayor salario de cada departamento

```
mysql> SELECT dep_no, emp_no, apellido
```

```
    -> FROM empleados
    -> WHERE (dep_no, salario) IN (SELECT dep_no, MAX(salario)
    ->                                 FROM empleados
    ->                                 GROUP BY dep_no)
    -> ORDER BY dep_no;
+--------+--------+----------+
| dep_no | emp_no | apellido|
+--------+--------+----------+
|     10 |   7839 | REY      |
|     20 |   7876 | GIL      |
|     30 |   7698 | GARRIDO  |
+--------+--------+----------+
3 rows in set (0.00 sec)
```

4 - Visualizar los empleados que tienen el mismo jefe y departamento que ALONSO excluido el mismo.

```
mysql> SELECT emp_no, apellido, director, dep_no
    -> FROM empleados
    -> WHERE (director, dep_no) = (SELECT director, dep_no
    ->                                 FROM empleados
    ->                                 WHERE apellido = 'ALONSO')
    -> AND apellido != 'ALONSO';

+--------+----------+----------+--------+
| emp_no | apellido | director | dep_no|
+--------+----------+----------+--------+
|   7654 | MARTIN   |     7698 |     30 |
|   7844 | CALVO    |     7698 |     30 |
+--------+----------+----------+--------+
2 rows in set (0.00 sec)
```

Ahora para que no suceda lo que en el ejemplo anterior y no se visualice ALONSO, que tiene igual jefe y departamento que el mismo, hemos añadido en la consulta la condición de que el apellido no sea ALONSO

4 - Subconsultas en la selección de grupos.

Aunque las subconsultas suelen encontrarse sobre todo en la cláusula WHERE, también pueden usarse en la HAVING formando parte de la selección del grupo de filas efectuada por dicha cláusula.

4.1 – Formato de subconsultas en la cláusula HAVING

Todos los formatos son exactamente iguales a cuando son utilizados en la cláusula WHERE

4.2 – Ejemplos de subconsultas en la cláusula HAVING

1. Visualizar los departamentos en los que el salario medio de sus empleados sea mayor o igual que la media de todos los salarios de la empresa.

```
mysql> SELECT dep_no "Nº Departamento",AVG(salario)"Salario Medio"
    ->     FROM empleados
    ->     GROUP BY dep_no
    ->     HAVING AVG(salario)>=(SELECT AVG(salario)
    ->                                     FROM empleados);

+----------------+--------------+
| Nº Departamento        | Salario Medio|
+----------------+--------------+
|                        10 |3266.833333 |
+----------------+--------------+
1 row in set (0.00 sec)
```

2. Visualizar los departamentos que tengan mayor media salarial total (salario + comision) que la mitad de la media salarial total de la empresa.

```
mysql> SELECT dep_no, AVG(salario+IFNULL(comision,0))
    -> FROM empleados
    -> GROUP BY dep_no
    -> HAVING AVG(salario+IFNULL(comision,0)) >
    ->                         (SELECT AVG(salario+IFNULL(comision,0))/2
    ->                                 FROM empleados);

+--------+---------------------------------+
| dep_no | AVG(salario+IFNULL(comision,0))|
+--------+---------------------------------+
|          10  |                                3266.833333  |
|          20  |                                2375.000000  |
|          30  |                                2637.530029  |
+--------+---------------------------------+
3 rows in set (0.03 sec)
```

3. Visualizar el departamento con menos presupuesto asignado para pagar el salario de sus empleados

```
mysql> SELECT dep_no "Departamento",
                    SUM(salario) "Mayor Presupuesto"
    -> FROM empleados

    -> GROUP BY dep_no
    -> HAVING SUM(salario) = (SELECT SUM(salario)
    ->                                 FROM empleados
    ->                                 GROUP BY dep_no
    ->                                 ORDER BY 1
    ->                                 LIMIT 1);

+--------------+-------------------+
| Departamento | Mayor Presupuesto|
+--------------+-------------------+
|                  20 |                    4750.00 |
+--------------+-------------------+
1 row in set (0.00 sec)
```

5 - Subconsultas anidadas.

5.1 – Anidación de subconsultas

Cuando una subconsulta forma parte de una condición de selección en una cláusula WHERE o HAVING
de otra subconsulta se dice que es una *subconsulta anidada.*

Las subconsultas se pueden anidar en varias cláusulas a la vez y en varios niveles. Esta posibilidad de anidamiento es lo que le da potencia a la instrucción select.

5.2 – Ejemplos de subconsultasanidadas

1- Visualizar el número y el nombre del departamento con más personal de oficio VENDEDOR.

```
mysql> SELECT dep_no "Nº Departamento", dnombre Departamento
    ->       FROM departamentos
    ->       WHERE dep_no=(SELECT dep_no
    ->                            FROM empleados
    ->                            WHERE oficio = 'VENDEDOR'
    ->                            GROUP BY dep_no
    ->                            HAVING COUNT(*)=(SELECT COUNT(*)
    ->                                                 FROM empleados
    ->                                                 WHERE oficio =
    'VENDEDOR'
    ->                                                 GROUP BY dep_no
    ->                                                 ORDER BY 1 DESC
    ->                                                 LIMIT 1));
+-----------------+--------------+
| Nº Departamento | Departamento|
+-----------------+--------------+
|                         30 | VENTAS                |
+-----------------+--------------+
1 row in set (0.03 sec)
```

2 - Visualizar los datos, número, nombre y localidad, del departamento donde trabaja el empleado más antiguo con el mismo oficio que GIL

```
mysql> SELECT dep_no, dnombre, localidad
    -> FROM departamentos
    -> WHERE dep_no=(SELECTdep_no
    ->                       FROMempleados
    ->                       WHERE (oficio,fecha_alta)=
    ->                                  (SELECT oficio, MIN(fecha_alta)
    ->                                     FROM empleados
    ->                                     WHERE oficio=(SELECT oficio
    ->                                                 FROM empleados
    ->                                                 WHERE apellido='GIL')
    ->                                     GROUP BY oficio));
```

```
+--------+---------------+-----------+
| dep_no | dnombre                   | localidad |
+--------+---------------+-----------+
|          20 | INVESTIGACION | VALENCIA                  |
+--------+---------------+-----------+
1 row in set (0.00 sec)
```

6 – Subconsultas correlacionadas

En una subconsulta podemos hacer referencias a las columnas de la tabla de la consulta. Cuando los nombres de columnas que aparecen en una subconsulta son nombres de columnas de la consulta principal o de otra subconsulta más externa, caso de las anidadas, se dice que son *referencias externas* y la *subconsulta* que es *correlacionada.*

6.1 – Correlación entre consultas

Las subconsultas correlacionadas funcionan de la siguiente forma: cada vez que se procesa una nueva fila en la consulta principal para decir si esa fila se selecciona o no, se ejecuta la subconsulta. En esa subconsulta podemos hacer referencia a las columnas de la consulta y los valores serán los de la fila con la que estamos trabajando en ese momento

Por ejemplo:
Visualizar los empleados que ganan más salario que la media de la empresa

```
SELECT dep_no "Nº Departamento", oficio,salario
FROM empleados
WHERE salario>(SELECT AVG(salario)
                    FROM empleados);
```

La tabla empleados se utiliza en la subconsulta para hallar el salario medio de la empresa y en la consulta para comprobar las filas que cumplen que el salario de ese empleado sea mayor que el salario medio calculado en la subconsulta.

Ahora supongamos que lo queremos modificar para que se visualicen los empleados que ganan más que la media de **su departamento**. En la subconsulta queremos hallar la media del departamento de cada empleado, es decir del departamento correspondiente al valor del campo dep_no en esa fila en la consulta. Por ejemplo si el primer empleado es del departamento 20 debemos calcular la media del del dep_no=20 para saber si el empleado gana mas que esa media, y si el siguiente empleado es del departamento 10 ahora deberemos calcular la media del dep_no=10.
Debemos referirnos a los valores de las columnas de la consulta dentro de la subconsulta y como son sobre la misma tabla tenemos que poner un alias para diferenciarlas.

En una subconsulta correlacionada si coincide el nombre de una columna de una referencia externa con el nombre de alguna columna de la tabla que está siendo seleccionada en la subconsulta, se deberá anteponer el nombre de la tabla externa para diferenciarlas. Si las tablas son la misma se deberá asignar un alias para diferenciarlas.

Hay dos posibilidades:

a) Poner un alias enambas

```
SELECT e1.dep_no "Nº Departamento", e1.oficio,salario
FROM empleados e1
WHERE e1.salario>(SELECT AVG(e2.salario)
                  FROM empleados e2
                  WHERE e2.dep_no = e1.dep_no);
```

b) Poner un alias en la tabla de la consulta

```
SELECT dep_no "Nº Departamento", oficio,salario
FROM empleados e1
WHERE salario>(SELECT AVG(salario)

                  FROM empleados
                  WHERE dep_no = e1.dep_no);
```

Esta segunda opción es posible porque dentro de una subconsulta, si no se le indica nada, supone que los nombres de las columnas corresponden a esa subconsulta. Para referenciar nombre externos es necesario anteponer el nombre de la tabla de la consulta y si ambas, la consulta y la subconsulta son sobre la misma tabla, es imprescindible el alias.

6.2 – Ejemplos de subconsultascorrelacionadas

1. Visualizar el número de departamento, el oficio y el salario de los empleados con mayor salario de cada departamento.

```
mysql> SELECT dep_no "Nº Departamento", oficio, salario
    ->    FROM empleados e1
    ->    WHERE salario = (SELECTMAX(salario)
    ->                          FROM empleados e2
    ->                          WHERE e1.dep_no=e2.dep_no);

+-----------------+------------+---------+
| Nº Departamento | oficio                | salario |
+-----------------+------------+---------+
|                 30 | DIRECTOR          | 3850.12 |
|                 10 | PRESIDENTE | 6000.00|
|                 20 | ANALISTA          | 3350.00 |
+-----------------+------------+---------+
3 rows in set (0.00 sec)
```

2 - Visualizar el empleado más antiguo de cada oficio

```
mysql> SELECT oficio, emp_no, apellido, fecha_alta
    ->    FROM empleados e1
    ->    WHERE fecha_alta = (SELECT MAX(fecha_alta)
 ->                          FROM empleados
    ->                          WHERE oficio=e1.oficio);
```

```
+------------+--------+----------+------------+
| oficio     | emp_no | apellido | fecha_alta|
+------------+--------+----------+------------+
| VENDEDOR   |   7654 | MARTIN   | 1981-09-28 |
| DIRECTOR   |   7782 | MARTINEZ | 1981-06-09|
| PRESIDENTE |   7839 | REY      | 1981-11-17 |
| ANALISTA   |   7876 | GIL      | 1982-05-06 |
| EMPLEADO   |   7900 | JIMENEZ  | 1983-03-24 |
+------------+--------+----------+------------+
5 rows in set (0.02 sec)
```

3 - Visualizar los empleados que tienen el menor salario de cada departamento

```
mysql> SELECT oficio, emp_no, apellido, salario
    ->     FROM empleados e1
    ->     WHERE salario = (SELECT MIN(salario)
    ->                      FROM empleados
    ->                      WHERE dep_no=e1.dep_no);
```

```
+----------+--------+----------+---------+

| oficio   | emp_no | apellido | salario|
+----------+--------+----------+---------+
| VENDEDOR |   7499 | ALONSO   | 1400.00 |
| EMPLEADO |   7521 | LOPEZ    | 1350.50 |
| EMPLEADO |   7900 | JIMENEZ  | 1400.00 |
+----------+--------+----------+---------+
3 rows in set (0.00 sec)
```

TEMA 8. CONSULTAS MULTITABLA

1 - Multiplicaciones de tablas (Producto cartesiano).

Hasta ahora, las órdenes SQL que hemos utilizado están basadas en una única tabla, pero a menudo es necesario utilizar datos procedentes de dos o más tablas de la base de datos.

Para poder acceder a dos o más tablas de una base de datos, SQL genera internamente una tabla en la que cada fila de una tabla se combina con todas y cada una de las filas de las demás tablas indicadas. Esta operación es el producto cartesiano de las tablas que se están accediendo y la tabla resultante contiene todas las columnas de todas las tablas que se han multiplicado.

Pueden unirse tantas tablas como se desee (aunque la experiencia aconseja que no sean muchas por optimización).

Comenzaremos con el formato más sencillo de multiplicación de tablas, reseñando solo las cláusulas significativas para realizar el producto.
Pueden ser utilizadas todas las cláusulas de selección vistas anteriormente y al final indicaremos el formato completo.

1.1 – Formato de la multiplicación de tablas

El formato más senillo para realizar un producto de tablas es:

SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna .....]*
FROM *NombreTabla [AliasTabla] [, NombreTabla [AliasTabla] ....]*

Notación: el nombre de tabla puede aparecer una o varias veces, separados por comas.

En la SELECT pueden seleccionarse columnas de ambas tablas. Si hay columnas con el mismo nombre en las distintas tablas de la FROM, deben identificarse como NombreTabla.NombreColumna o AliasTabla.NombreColumna. Para no tener que escribir siempre el nombre de la tabla, por comodidad, se suele utilizar un alias para cada tabla eligiendo un nombre corto (1 o 2 caracteres) que identifique a cada tabla. En algún caso, que veremos más adelante, este alias será imprescindible.

1.2 – Ejemplos de la multiplicación de tablas

Veremos la salida que produce la multiplicación, o producto cartesiano, de dos tablas con un ejemplo para obtener todos los empleados, indicando su número de empleado, su apellido, el nombre de su departamento y su localidad de este.

```
mysql> SELECT emp_no "Nº EMPLEADO", apellido "APELLIDO",
    ->        dnombre "DEPARTAMENTO" , localidad "LOCALIDAD"
    -> FROM empleados, departamentos;

+-------------+----------+---------------+-----------+
| Nº EMPLEADO | APELLIDO | DEPARTAMENTO  | LOCALIDAD |
+-------------+----------+---------------+-----------+
|        7499 | ALONSO   | CONTABILIDAD  | BARCELONA |
|        7499 | ALONSO   | INVESTIGACION | VALENCIA  |
|        7499 | ALONSO   | VENTAS        | MADRID    |
|        7499 | ALONSO   | PRODUCCION    | SEVILLA   |
|        7521 | LOPEZ    | CONTABILIDAD  | BARCELONA |
|        7521 | LOPEZ    | INVESTIGACION | VALENCIA  |
|        7521 | LOPEZ    | VENTAS        | MADRID    |
|        7521 | LOPEZ    | PRODUCCION    | SEVILLA   |
|        7654 | MARTIN   | CONTABILIDAD  | BARCELONA |
|        7654 | MARTIN   | INVESTIGACION | VALENCIA  |
|        7654 | MARTIN   | VENTAS        | MADRID    |
|        7654 | MARTIN   | PRODUCCION    | SEVILLA   |
|        7698 | GARRIDO  | CONTABILIDAD  | BARCELONA |
|        7698 | GARRIDO  | INVESTIGACION | VALENCIA  |
|        7698 | GARRIDO  | VENTAS        | MADRID    |
|        7698 | GARRIDO  | PRODUCCION    | SEVILLA   |
|        7782 | MARTINEZ | CONTABILIDAD  | BARCELONA |
|        7782 | MARTINEZ | INVESTIGACION | VALENCIA  |
|        7782 | MARTINEZ | VENTAS        | MADRID    |
```

```
|          7782 | MARTINEZ | PRODUCCION      | SEVILLA    |
|          7839 | REY      | CONTABILIDAD    | BARCELONA  |
|          7839 | REY      | INVESTIGACION   | VALENCIA   |
|          7839 | REY      | VENTAS          | MADRID     |
|          7839 | REY      | PRODUCCION      | SEVILLA    |
|          7844 | CALVO    | CONTABILIDAD    | BARCELONA  |
|          7844 | CALVO    | INVESTIGACION   | VALENCIA   |
|          7844 | CALVO    | VENTAS          | MADRID     |
|          7844 | CALVO    | PRODUCCION      | SEVILLA    |
|          7876 | GIL      | CONTABILIDAD    | BARCELONA  |
|          7876 | GIL      | INVESTIGACION   | VALENCIA   |
|          7876 | GIL      | VENTAS          | MADRID     |
|          7876 | GIL      | PRODUCCION      | SEVILLA    |
|          7900 | JIMENEZ  | CONTABILIDAD    | BARCELONA  |
|          7900 | JIMENEZ  | INVESTIGACION   | VALENCIA   |
|          7900 | JIMENEZ  | VENTAS          | MADRID     |
|          7900 | JIMENEZ  | PRODUCCION      | SEVILLA    |
+-------------+----------+---------------+-----------+
36 rows in set (0.00 sec)
```

Cada empleado de la tabla de empleadosaparece tantas veces como departamentos hay en la tabla de
departamentos, con los correspondientes valores de cada de las filas de la tabla departamentos. Se obtienen, por tanto, 9 x 4= 36 filas.

2 - Composiciones o combinaciones simples (JOIN)

Acabamos de ver, en el ejemplo anterior, que la salida producida por la multiplicación de las tablas de
empleadosy de departamentos no tiene mucho sentido y poca aplicación.

La solución más adecuada del ejemplo sería enlazar la tabla empleados con la tabla departamentos de forma que para cada empleado de la tabla empleados solo tengamos de la tabla departamentosla fila correspondiente a su departamento.
Es decir:

```
mysql> SELECT emp_no "Nº EMPLEADO", apellido
                  "APELLIDO", dnombre DEPARTAMENTO,
                  localidad"LOCALIDAD"
    -> FROM empleados, departamentos
    -> WHERE empleados.dep_no=departamentos.dep_no;

+-------------+----------+---------------+-----------+
| Nº EMPLEADO | APELLIDO| DEPARTAMENTO              | LOCALIDAD |
+-------------+----------+---------------+-----------+
|          7521 | LOPEZ    | CONTABILIDAD    | BARCELONA  |
|          7782 | MARTINEZ | CONTABILIDAD    | BARCELONA  |
|          7839 | REY      | CONTABILIDAD    | BARCELONA  |
|          7876 | GIL      | INVESTIGACION   | VALENCIA   |
|          7900 | JIMENEZ  | INVESTIGACION   | VALENCIA   |
|          7499 | ALONSO   | VENTAS          | MADRID     |
|          7654 | MARTIN   | VENTAS          | MADRID     |
|          7698 | GARRIDO  | VENTAS          | MADRID     |
|          7844 | CALVO    | VENTAS          | MADRID     |
+-------------+----------+---------------+-----------+
9 rows in set (0.00 sec)
```

Hemos visto que la salida que produce el producto de las tablas es cada fila de una tabla combinadas con todas las de otra tabla. Esta información no suele ser la deseada. Aparecen las ***composiciones o combinaciones de tablas*** que nos proporcionan la misma información pero filtrada. Una composición o combinación (join) consiste en aplicar una condición de selección a las filas obtenidas de la multiplicación de las tablas sobre las que se está realizando una consulta.

2.1 – Formato de la combinación detablas

SELECT [ALL/DISTINCT] *ExpresionColumna* [,*ExpresionColumna .....]*

FROM *NombreTabla [AliasTabla] [, NombreTabla [AliasTabla].....]*

[WHERE *CondicionComposicion*]

Notación: la cláusula WHERE es opcional y por eso aparece entre corchetes.

donde ***CondicionComposicion*** .....es una condición que selecciona las filas de la composición de las tablas.

La condición de selección o criterio de emparejamiento para las tablas también se denomina *condición o criterio de composición.*

Dependiendo de la condición de composición tendremos las combinaciones naturales, basadas en la igualdad, o las combinaciones basadas en la desigualdad.

Existen varios tipos según sea esta condición de composición

- ***Composición natural***: es la más sencilla y natural y es aquella en que la condición de selección se establece con el operador de igualdad entre las columnas que deban coincidir exactamente en tablas diferentes.
- ***Composición basada en desigualdad***: es menos utilizada y consiste en que la condición de selección no sea una igualdad.

3- Composición o combinación natural

Como hemos dicho la composición o combinación natural es la más sencilla y natural. Es aquella en que la condición de selección se establece con el operador de igualdad entre las columnas que deben coincidir exactamente en tablas diferentes.

Suele utilizarse para unir tablas en las que hay una relación a través de las claves ajenas, uniendo una tabla con su referenciada.

Veamos con el ejemplo porqué es la forma más natural:

```
mysql> SELECT emp_no "Nº EMPLEADO", apellido
                    "APELLIDO", dnombre DEPARTAMENTO,
                    localidad"LOCALIDAD"
    ->  FROM empleados, departamentos
    ->  WHERE empleados.dep_no=departamentos.dep_no;

+-------------+----------+---------------+-----------+
| Nº EMPLEADO | APELLIDO| DEPARTAMENTO              | LOCALIDAD |
+-------------+----------+---------------+-----------+
|        7521 | LOPEZ    | CONTABILIDAD  | BARCELONA |
|        7782 | MARTINEZ | CONTABILIDAD  | BARCELONA |
|        7839 | REY      | CONTABILIDAD  | BARCELONA |
|        7876 | GIL      | INVESTIGACION | VALENCIA  |
|        7900 | JIMENEZ  | INVESTIGACION | VALENCIA  |
|        7499 | ALONSO   | VENTAS        | MADRID    |
|        7654 | MARTIN   | VENTAS        | MADRID    |
|        7698 | GARRIDO  | VENTAS        | MADRID    |
|        7844 | CALVO    | VENTAS        | MADRID    |
+-------------+----------+---------------+-----------+
9 rows in set (0.00 sec)
```

Si obtenemos el producto cartesiano de las tablas empleados por departamentos para cada empleado obtenemos la combinación con todas las filas de departamento. Pero la única que nos interesará será la del departamento al que pertenezca el empleado, es decir la que cumpla la condición: **departamento.dep_no = empleado.dep_no**

SQL resuelve el anterior ejercicio con el siguiente proceso:

- La cláusula FROM genera todas las combinaciones posibles de filas de la tabla de *empleados* (9 filas) por las de la tabla de *departamentos* (4 filas), resultando una tabla producto de 4x9=36 filas.
- La cláusula WHERE selecciona únicamente aquellas filas de la tabla producto donde coinciden los números de departamento, que necesitan el nombre de la tabla o el alias por tener el mismo nombre en ambas tablas. En total se han seleccionado 9 filas y las 27 restantes se eliminan.
- La sentencia SELECT visualiza las columnas especificadas de las tablas producto para las filas seleccionadas.

SQL no exige que las columnas de emparejamiento estén relacionadas como clave primaria y clave ajena, aunque suele ser lo habitual. Pueden servir cualquier par de columnas de dos tablas, siempre que tengan tipos de datos comparables.

3.1 – Formato de la combinación natural detablas

Es el mismo de la combinación o composición pero donde la condición de composición es una igualdad entre campos de diferentes tablas:

```
[NombreTabla1.] NombreColumna1 = [NombreTabla2.] NombreColumna2
```

Los nombres de las columnas necesitarán anteponer el nombre de la tabla o el alias si el nombre es el mismo en ambas tablas.

3.2 – Ejemplos de combinación natural detablas

1. Obtener los distintos departamentos existentes en la tabla de *empleados.*

```
mysql> SELECT DISTINCT d.dep_no "Nº Departamento",
                        d.dnombre Departamento
    -> FROM empleados e, departamentos d
    ->     WHERE e.dep_no = d.dep_no;

+-----------------+---------------+
| Nº Departamento | Departamento  |
+-----------------+---------------+
|              10 | CONTABILIDAD  |
|              20 | INVESTIGACION |
|              30 | VENTAS        |
+-----------------+---------------+
3 rows in set (0.02 sec)
```

2. Mostrar los siguientes datos relativos a empleados: número, apellido, nombre de departamento y localidad.

```
mysql> SELECT emp_no "Nº Empleado", apellido, dnombre,localidad
   ->                 FROM empleados e, departamentos d
      ->                 WHERE e.dep_no = d.dep_no;

+-------------+----------+----------------+-----------+
| Nº Empleado | apellido | dnombre        | localidad |
+-------------+----------+----------------+-----------+
|        7521 | LOPEZ    | CONTABILIDAD   | BARCELONA |
|        7782 | MARTINEZ | CONTABILIDAD   | BARCELONA |
|        7839 | REY      | CONTABILIDAD   | BARCELONA |
|        7876 | GIL      | INVESTIGACION  | VALENCIA  |
|        7900 | JIMENEZ  | INVESTIGACION  | VALENCIA  |
|        7499 | ALONSO   | VENTAS         | MADRID    |
|        7654 | MARTIN   | VENTAS         | MADRID    |
|        7698 | GARRIDO  | VENTAS         | MADRID    |
|        7844 | CALVO    | VENTAS         | MADRID    |
+-------------+----------+----------------+-----------+
9 rows in set (0.02 sec)
```

4 - Composiciones o combinaciones basadas en desigual

La condición de selección que establezcamos para componer o combinar tablas no tiene porqué ser siempre mediante el operador aritmético de igualdad, aunque su uso sea el más frecuente. SQLpermite utilizar cualquier operador aritmético de comparación.
Estas composiciones son aquellas en las que el operador de la condición de selección NO es la igualdad.

4.1 – Formato de combinaciones basadas endesigualdad

Es el mismo de la combinación o composición pero la condición de composición es diferente de la igualdad entre campos de diferentes tablas.

4.2 – Ejemplosde combinaciones basadas en desigualdad

1. Listar los empleados de los departamentos diferentes al de VENTAS.

```
mysql> SELECT e.emp_no "NºEmpleado",e.apellido "Apellido"
    -> FROM empleados e,departamentos d
-> WHERE d.dnombre = 'VENTAS' AND e.dep_no != d.dep_no;

+------------+----------+
| NºEmpleado | Apellido|
+------------+----------+
|       7521 | LOPEZ    |
|       7782 | MARTINEZ|
|       7839 | REY      |
|       7876 | GIL      |
|       7900 | JIMENEZ  |
+------------+----------+
5 rows in set (0.00 sec)
```

2. Listar los empleados de departamentos con códigos mayores que el código del departamento de contabilidad.

```
mysql> SELECT e.emp_no "NºEmpleado",e.apellido
    -> FROM empleados e,departamentos d
-> WHERE d.dnombre = 'CONTABILIDAD' AND e.dep_no>d.dep_no;

+------------+----------+
| NºEmpleado | apellido|
+------------+----------+
|       7499 | ALONSO   |
|       7654 | MARTIN   |
|       7698 | GARRIDO  |
|       7844 | CALVO    |
|       7876 | GIL      |
|       7900 | JIMENEZ  |
+------------+----------+
6 rows in set (0.00 sec)
```

3. Listar los empleados de departamentos con códigos menores que el código del departamento de

Barcelona

```
mysql> SELECT e.emp_no "NºEmpleado",e.apellido
    -> FROM empleados e,departamentos d
    -> WHERE d.localidad =
                'BARCELONA' AND
                e.dep_no > d.dep_no;

+------------+----------+
| NºEmpleado | apellido |
+------------+----------+
|       7499 | ALONSO   |
|       7654 | MARTIN   |
|       7698 | GARRIDO  |
|       7844 | CALVO    |
|       7876 | GIL      |
|       7900 | JIMENEZ  |
+------------+----------+
6 rows in set (0.00 sec)
```

5 - Composiciones o combinaciones de una tabla consigo misma

Si desde una fila de una tabla podemos acceder a otra fila de la misma tabla, duplicando la tabla, estamos realizando una composición o combinación de una tabla consigo misma.

5.1 - Formato de una combinación de una tabla consigo misma.

Es el mismo de la combinación o composición pero las tablas del producto son las mismas tablas.

En este caso, como ambas tablas se llaman igual, es necesario obligatoriamente un alias para diferenciar las columnas de cada tabla.

5.2 - Ejemplos de una combinación de una tabla consigomisma

1. Obtener la lista de los empleados con los nombres de sus directores.

```
mysql>  SELECT e1.emp_no "NºEmpleado",
            e1.apellido "Nombre
            Empleado", e1.director
            "NºDirector", e2.apellido
            "Nombre Director"
    ->  FROM empleados e1,empleados e2
    ->  WHERE e1.director=e2.emp_no;
```

```
+------------+-----------------+------------+-----------------+
| NºEmpleado | Nombre Empleado | NºDirector | Nombre Director |
+------------+-----------------+------------+-----------------+
|       7499 | ALONSO          |       7698 | GARRIDO         |
|       7521 | LOPEZ           |       7782 | MARTINEZ        |
|       7654 | MARTIN          |       7698 | GARRIDO         |
|       7698 | GARRIDO         |       7839 | REY             |
|       7782 | MARTINEZ        |       7839 | REY             |
|       7844 | CALVO           |       7698 | GARRIDO         |
|       7876 | GIL             |       7782 | MARTINEZ        |
|       7900 | JIMENEZ         |       7782 | MARTINEZ        |
+------------+-----------------+------------+-----------------+
8 rows in set (0.00 sec)
```

Cada empleado de la tabla tiene una columna para su número de empleado *(emp_no)* y otra para el número de empleado de su director *(director).* A partir del dato de la columna *director* de un empleado se puede acceder a otro empleado que contenga el mismo dato en su columna *emp_no.* Las dos filas de la tabla se están relacionando a través de las columnas *director* y *emp_no.*
El uso de alias es obligado por tratarse de la misma tabla y coincidir los nombres de las columnas.

2. Obtener los jefes de los empleados cuyo oficio sea el de VENDEDOR.

```
mysql> SELECT e1.emp_no "NºEmpleado", e1.apellido, e1.oficio,
    ->        e1.director "NºDirector", e2.apellido "Nombre Director"
    -> FROM empleados e1, empleados e2
    -> WHERE e1.director = e2.emp_no AND e1.oficio = 'VENDEDOR';
```

```
+------------+----------+----------+------------+-----------------+
| NºEmpleado | apellido | oficio   | NºDirector | Nombre Director |
+------------+----------+----------+------------+-----------------+
|       7499 | ALONSO   | VENDEDOR |       7698 | GARRIDO         |
|       7654 | MARTIN   | VENDEDOR |       7698 | GARRIDO         |
|       7844 | CALVO    | VENDEDOR |       7698 | GARRIDO         |
+------------+----------+----------+------------+-----------------+
3 rows in set (0.00 sec)
```

6 - Composiciones o combinaciones externas (OUTER JOIN)

Cuando se realiza una composición o combinación de tablas estableciendo una determinada relación entre sus columnas, puede ocurrir que no se emparejen todas las filas que debieran por faltar correspondencia entre algunas de ellas. Esto se debe a que existen filas en alguna tabla que no tienen correspondencia en la otra tabla y al aplicar la condición de selección de la composición no se seleccionarán.

Por ejemplo

```
mysql> SELECT dnombre Departamento, localidad, emp_no,apellido
    ->    FROM empleados e,departamentos d
    ->    WHERE e.dep_no=d.dep_no;
+---------------+-----------+--------+----------+
| Departamento  | localidad | emp_no | apellido |
+---------------+-----------+--------+----------+
| CONTABILIDAD  | BARCELONA |   7521 | LOPEZ    |
| CONTABILIDAD  | BARCELONA |   7782 | MARTINEZ |
```

```
| CONTABILIDAD    | BARCELONA  |      7839 | REY      |
| INVESTIGACION   | VALENCIA   |      7876 | GIL      |
| INVESTIGACION   | VALENCIA   |      7900 | JIMENEZ  |
| VENTAS          | MADRID     |      7499 | ALONSO   |
| VENTAS          | MADRID     |      7654 | MARTIN   |
| VENTAS          | MADRID     |      7698 | GARRIDO  |
| VENTAS          | MADRID     |      7844 | CALVO    |
+---------------+-----------+--------+----------+
9 rows in set (0.00 sec)
```

Aquellos departamentos que no tengan empleados no aparecerán porque para esos departamentos no se cumplirá la igualdad empleados.dep_no = departamentos.dep_no

Es aconsejable que la salida, obtenida por una consulta en la que se pudiera presentar esta posibilidad, muestre todas las filas, aunque algunas con falta de información. Para conseguir este resultado se utiliza la composición o combinación externa (OUTER JOIN).

6.1 – Formato de combinaciones externas

```
SELECT [ALL/DISTINCT] ExpresionColumna [,ExpresionColumna .....]
FROM NombreTabla [AliasTabla]
        {LEFT|RIGHT [OUTER] JOIN NombreTabla [AliasTabla].....]
ON CondicionComposicion
```

donde **LEFT|RIGHT [OUTER] JOIN……..** indica que es un join externo y si la extensión del

producto de las tablas se quiere realizar por la izquierda o por la derecha

CondicionComposicion …….. es la misma condición de composición anterior, pero

escrita aquí en lugar de en la cláusula WHERE

El funcionamiento de un join externo es el siguiente:

- LEFT JOIN: join donde se obtienen todas las filas de la tabla de la izquierda, aunque no

  tenga correspondencia en la tabla de la derecha.
  Realiza el producto cartesiano de las tablas que se le indican, aplica la condición de composición (expresada en la cláusula ON) al resultado de este producto cartesiano y añade, por cada fila de la tabla de la izquierda que no tenga correspondencia en la tabla de la derecha, una fila con los valores de la tabla de la izquierda y en la tabla de la derecha valores NULL en todas las columnas.
- RIGHT JOIN: join donde se obtienen todas las filas de la tabla de la derecha, aunque no tengan correspondencia en la tabla de la izquierda.
  Realiza el producto cartesiano de las tablas que se le indican, aplica la

condición de composición (expresada en la cláusula ON) al resultado de este producto cartesiano y añade, por cada fila de la tabla de la derecha que no tenga correspondencia en la tabla de la izquierda, una fila con los valores de la tabla de la derecha y en la tabla de la izquierda valores NULL en todas las columnas.

Por ejemplo si queremos visualizar los datos de los departamentos y de sus empleados, visualizando también los departamentos que no tengan empleados.

```
mysql> SELECT dnombre Departamento,
           localidad, emp_no "Nº empleado",
           apellido
    ->   FROM departamentos d LEFT JOIN empleados e
      ->   ON d.dep_no=e.dep_no;

+---------------+-----------+------------+----------+
| Departamento  | localidad | Nºempleado | apellido|
+---------------+-----------+------------+----------+
| CONTABILIDAD  | BARCELONA |       7521 | LOPEZ    |
| CONTABILIDAD  | BARCELONA |       7782 | MARTINEZ |
| CONTABILIDAD  | BARCELONA |       7839 | REY      |
| INVESTIGACION | VALENCIA  |       7876 | GIL      |
| INVESTIGACION | VALENCIA  |       7900 | JIMENEZ  |
| VENTAS        | MADRID    |       7499 | ALONSO   |
| VENTAS        | MADRID    |       7654 | MARTIN   |
| VENTAS        | MADRID    |       7698 | GARRIDO  |
| VENTAS        | MADRID    |       7844 | CALVO    |
| PRODUCCION    | SEVILLA   |       NULL | NULL     |
+---------------+-----------+------------+----------+
10 rows in set (0.00 sec)
```

Aparecerán todas las filas de la tabla DEPARTAMENTOS, tanto si tienen correspondencia en la tabla EMPLEADOS como si no la tienen. Los departamentos que no tengan empleados también aparecerían. El departamento de PRODUCCIÓN no tiene ningún empleado asignado y se añade una fila en la tabla empleados con todos los campo con valor NULL en correspondencia.

Obtendremos el mismo resultado si cambiamos LEFT por RIGHT el orden de las tablas

```
mysql> SELECT dnombre Departamento, localidad,
    ->          emp_no "Nº empleado",apellido
    ->     FROM empleados e RIGHT JOIN departamentos d
    ->     ON d.dep_no=e.dep_no;

+---------------+-----------+-------------+----------+
| Departamento  | localidad | Nº empleado | apellido|
+---------------+-----------+-------------+----------+
| CONTABILIDAD  | BARCELONA |        7521 | LOPEZ    |
| CONTABILIDAD  | BARCELONA |        7782 | MARTINEZ |
| CONTABILIDAD  | BARCELONA |        7839 | REY      |
| INVESTIGACION | VALENCIA  |        7876 | GIL      |
| INVESTIGACION | VALENCIA  |        7900 | JIMENEZ  |
| VENTAS        | MADRID    |        7499 | ALONSO   |
| VENTAS        | MADRID    |        7654 | MARTIN   |
```

```
| VENTAS          | MADRID    |          7698 | GARRIDO   |
| VENTAS          | MADRID    |          7844 | CALVO     |
| PRODUCCION      | SEVILLA   |          NULL | NULL      |
+-----------------+-----------+---------------+-----------+
10 rows in set (0.00 sec)
```

Hay casos en los que hacer el OUTER JOIN obtendremos el mismo resultado con una combinación natural, ya que no hay filas sin correspondencia en la tabla de la correspondencia.

```
mysql> SELECT emp_no "Nºempleado",
            apellido, dnombre
            "Departamento",localidad
    -> FROM departamentos d RIGHT JOIN empleados e
    -> ON d.dep_no=e.dep_no;

+------------+----------+---------------+-----------+
| Nºempleado | apellido| Departamento          | localidad |
+------------+----------+---------------+-----------+
|       7499 | ALONSO   | VENTAS        | MADRID    |
|       7521 | LOPEZ    | CONTABILIDAD  | BARCELONA |
|       7654 | MARTIN   | VENTAS        | MADRID    |
|       7698 | GARRIDO  | VENTAS        | MADRID    |
|       7782 | MARTINEZ | CONTABILIDAD  | BARCELONA |
|       7839 | REY      | CONTABILIDAD  | BARCELONA |
|       7844 | CALVO    | VENTAS        | MADRID    |
|       7876 | GIL      | INVESTIGACION | VALENCIA  |
|       7900 | JIMENEZ  | INVESTIGACION | VALENCIA  |
+------------+----------+---------------+-----------+
9 rows in set (0.00 sec)
```

Aparecerán todas las filas de la tabla EMPLEADOS, tanto si tienen correspondencia en la tabla DEPARTAMENTOS como si no. Los empleados que no tuviesen departamento asignado también aparecerían. En este ejemplo como todos los empleados tienen departamento asignado el resultado es el mismo.

6.2 – Ejemplos de combinaciones externas

1. Obtener los departamentos con su nombre y localidad y el número de empleados trabajando en ellos, incluyendo los que no tienen empleados.

```
mysql> SELECT dnombre Departamento, localidad, COUNT(emp_no)
    -> FROM departamentos d LEFT JOIN empleados e
    -> ON d.dep_no=e.dep_no
    -> GROUP BY dnombre, localidad;

+---------------+-----------+---------------+
| Departamento  | localidad | COUNT(emp_no)|
+---------------+-----------+---------------+
| CONTABILIDAD  | BARCELONA |             3 |
| INVESTIGACION | VALENCIA  |             2 |
| PRODUCCION    | SEVILLA   |             0 |
| VENTAS        | MADRID    |             4 |
+---------------+-----------+---------------+
```

4 rows in set (0.00 sec)

2. Obtener la lista de empleados con los nombres de sus directores, incluyendo al PRESIDENTE. (Ejemplo en autocomposiciones) .

```
mysql> SELECT e1.apellido "Nombre
                    Empleado", e2.apellido
                    "Nombre Director"
    -> FROM empleados e1 LEFT JOIN empleados e2
    -> ON e1.director = e2.emp_no;
```

```
+----------+-----------------+
| apellido | Nombre Director|
+----------+-----------------+
| ALONSO       | GARRIDO              |
| LOPEZ        | MARTINEZ             |
| MARTIN       | GARRIDO              |
| GARRIDO      | REY                  |
| MARTINEZ | REY                      |
| REY          | NULL                 |
| CALVO        | GARRIDO              |
| GIL          | MARTINEZ             |
| JIMENEZ      | MARTINEZ             |
+----------+-----------------+
9 rows in set (0.00 sec)
```

7 - Composiciones y subconsultas

Hay ocasiones en que una consulta puede resolverse con una composición o combinación (join) de tablas o con una subconsulta.

Si puede solucionarse de ambas formas será preferible hacerlo con una subconsulta. El producto cartesiano es muy costoso pues hay que multiplicar todas las filas de una tabla por todas las de la otra tabla para después seleccionar solo algunas. Con tablas con miles o millones de registros esto es un trabajo muy costoso en tiempo y memoria, que puede resolver con una subconsulta.

En general, si no se necesita visualizar columnas de más de una tabla, se debe utilizar una subconsulta. Solamente si se necesita visualizar columnas de más de una tabla, se usará una composición o combinación.

Vamos a verlo con unos ejemplos:

1- Con subconsulta:

Obtener apellido y oficio de los empleados que tienen el mismo oficio y mismo número de departamento que el de INVESTIGACIÓN.

```
mysql> SELECT apellido, oficio
    ->    FROM empleados
    ->    WHERE oficio IN (SELECT oficio
    ->                     FROM empleados
    ->                     WHERE dep_no IN (SELECT dep_no
```

```
    ->                                  FROM departamentos
    ->                                  WHERE
    dnombre='INVESTIGACION'));

+----------+----------+
| apellido | oficio                |
+----------+----------+
| LOPEZ          | EMPLEADO |
| GIL            | ANALISTA |
| JIMENEZ        | EMPLEADO |
+----------+----------+
3 rows in set (0.00 sec)
```

Puede solucionarse con una subconsulta porque solo nos piden visualizar campos de la tabla empleados. Debe solucionarse, por tanto, utilizarse una subconsulta.

2- Con composición de las tablas:

Obtener apellido, el oficio y la localidad del departamento de los empleados que tienen el mismo oficio y mismo número de departamento que el de INVESTIGACIÓN.

```
mysql> SELECT apellido, oficio,localidad
    ->    FROM empleados e, departamentosd
    ->    WHERE oficio IN (SELECT oficio
    ->                                FROM empleados
    ->                                WHERE e.dep_no = d.dep_no
                                            AND d.dnombre = 'INVESTIGACION');

+----------+----------+-----------+
| apellido | oficio                | localidad |
+----------+----------+-----------+
| GIL            | ANALISTA | VALENCIA            |
| JIMENEZ        | EMPLEADO | VALENCIA            |
+----------+----------+-----------+
2 rows in set (0.00 sec)
```

Es el mismo ejercicio pero, en este ejemplo, nos piden visualizar campos de la tabla empleados y de la tabla departamentos. No puede solucionarse con una subconsulta y debe, por tanto, solucionarse con un producto de ambastablas.

8 - Formato completo de las consultas

Aunque hasta ahora no hemos utilizado todas las cláusulas estudiadas, en la composición de tablas está permitidas todas ellas. Podemos hacer selección de filas, agrupamientos, selección de grupo, ordenación y limitar el número de filas de igual forma que lo hacíamos para consultas con una sola tabla

```
SELECT { * | [ALL/DISTINCT] ExpresionColumna [AliasColumna]
                          [, ExpresionColumna [AliasColumna ]....] }
FROM NombreTabla  [AliasTabla]  [,NombreTabla  [AliasTabla]....]
[WHERE { CondicionSeleccion | CondicionComposicion }]
[GROUP BY ExpresionColumna|Posicion [, ExpresionColumna|Posicion ... ]
        [HAVING CondicionSelecionGrupos ] ]
[ORDER BY {ExpresionColumna|Posicion } [ASC|DESC]
            [,{ExpresionColumna|Posicion } [ASC|DESC].....] ]
[LIMIT [m, ] n ] ;
```

TEMA 9. CONSULTAS DENTRO DE OTRAS INSTRUCCIONES

1 - Creación de una tabla a partir de una selección de otra tabla.

En el momento de crear una tabla podemos utilizar la definición de otra ya creada, utilizando toda la definición o aquella parte que no interese.

1.1 – Formato para la creación de una tabla a partir de una selección de otra tabla.

```
CREATE TABLE [IF NOT EXISTS] NombreTabla
   [( DefinicionColumna [, DefinicionColumna ...] )] [ IGNORE|REPLACE ]
 SentenciaSelect
```

Notación: las definiciones de las columnas son opcionales, por lo que están entre corchetes.

donde **NombreTabla .....** es el identificador de la nueva tabla que se crea
SentenciaSelect ..... es cualquier sentencia select
IGNORE|REPLACE ..... opciones de duplicados en campos únicos

Los nombres de las columnas de la nueva tabla son opcionales. En caso de que no se especifique tomarán los valores de la otra tabla o de los alias correspondientes. Pero debe tenerse cuidado con ello pues si hay expresiones o funciones en la lista del select y no tienen alias ni nuevo nombre, luego estas columnas no pueden referenciarse.

Las opciones de duplicados en campos únicos (claves primarias o unique) permiten indicar que hacer si hay un valor repetido en ese campo. Si no se indica nada se producirá un error. Para evitar esto errores podemos especificar una de las dos opciones IGNORE que ignora la fila y no la graba y REPLACE que reemplaza la fila por la anterior.

La nueva tabla creada no hereda las CONSTRAINTSque tenga asignada la tabla origen. Esto es para dar más flexibilidad al sistema. Si se desea que la nueva tabla tenga constraints deben indicarse en la creación o añadirse posteriormente con ALTERTABLE. Es importante recordarlo para que la nueva tabla

quede con la definición completa.

Al crear la nueva tabla además se insertan las filas correspondientes de la tabla resultado de la sentencia select en la instrucción de creación.

1.2 – Ejemplos de la creación de una tabla a partir de una selección de otra tabla.

1. Crear una tabla de vendedores seleccionando éstos de la tabla de empleados.

```
mysql> CREATE TABLE vendedores
    -> SELECT *
    -> FROM empleados
    -> WHERE oficio = 'VENDEDOR';
Query OK, 3 rows affected (0.13 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM vendedores;
+--------+----------+---------+---------+------------+---------+---------+--------+
| EMP_NO | APELLIDO | OFICIO  |DIRECTOR | FECHA_ALTA | SALARIO | COMISION | DEP_NO|
+--------+----------+---------+---------+------------+---------+----------+-------+
|   7499 | ALONSO   | VENDEDOR|    7698 | 1981-02-23 | 1400.00 | 650.00 |   30 |
|   7654 | MARTIN   | VENDEDOR|    7698 | 1981-09-28 | 1500.00 | 1850.00|   30 |
|   7844 | CALVO    | VENDEDOR|    7698 | 1981-09-08 | 1800.00 |  250.00|   30 |
+--------+----------+---------+---------+------------+---------+---------+--------+
3 rows in set (0.00sec)
```

2. Crear una nueva tabla sólo con los nombres y números de los departamentos a partir de la tabla ya creada con los mismos.

```
mysql> CREATE TABLE nombres_dep (depart_no INT(4),
                                 nombre VArchar(10))
    ->    SELECT dep_no, dnombre
    ->    FROM departamentos;
Query OK, 4 rows affected (0.23 sec)
Records: 4  Duplicates: 0
            Warnings: 0

mysql> SELECT * FROM nombres_dep;
+--------+---------------+
| dep_no | dnombre       |
+--------+---------------+
|     10 | CONTABILIDAD  |
|     20 | INVESTIGACION |
|     30 | VENTAS        |
|     40 | PRODUCCION    |
+--------+---------------+
4 rows in set (0.00 sec)
```

2 - Actualización de una tabla a partir de una subconsulta

Todas las instrucciones de actualización de datos que hemos visto pueden ampliarse con la utilización de subconsultas dentro de ellas.

2.1- Inserciones con subconsultas

Podemos insertar en una tabla el resultado de una consulta sobre otra tabla. En este caso normalmente se

insertarán varias filas con una sola sentencia. Utilizaremos el siguiente formato:

```
INSERT INTO NombreTabla [( NombreColumna [,NombreColumna...] ) ]
  SELECT FormatoSelect
```

Notación: la lista de columnas en las que insertamos va es opcional, por lo que va entre corchetes.

En el formato anterior podemos destacar:

- La **lista de columnas es opcional** pero deberá especificarse cuando las columnas que devuelve la consulta no coinciden en número o en orden con las columnas de la tabla destino.
- La **consulta puede ser cualquier comando de selección** válido siempre que exista una correspondencia entre las columnas devueltas y las columnas de la tabla destino, o la lista de columnas.

En este caso existe correspondencia en número y en orden entre las columnas de la tabla destino y las columnas de la selección; por tanto, no hace falta especificar la lista de columnas y el comando requerido será:

```
mysql> INSERT INTO departamentos2
    -> SELECT * FROM departamentos
    -> WHERE LENGTH(dnombre)> 8;
Query OK, 3 rows affected (0.05 sec)
Records: 3      Duplicates: 0
                Warnings: 0

mysql> SELECT *
    -> FROM departamentos2;
+--------+---------------+-----------+
| DEP_NO | DNOMBRE       | LOCALIDAD |
+--------+---------------+-----------+
|     10 | CONTABILIDAD  | BARCELONA |
|     20 | INVESTIGACION | VALENCIA  |
|     40 | PRODUCCION    | SEVILLA   |
+--------+---------------+-----------+
3 rows in set (0.00 sec)
```

Si la tabla destino tuviese una estructura diferente deberemos forzar la correspondencia, bien al especificar la lista de selección, bien especificando la lista de columnas, o bien utilizando ambos recursos.

Por ejemplo, supongamos que la tabla destino es *n2dep*

```
mysql> CREATE TABLE n2dept

    -> ( DEP_NO                                  INT(2),
    ->     NOMBRE                               VARCHAR(14),
    ->     CONSTRAINT PK_DEPARTAMENTOS_DEP_NO PRIMARY KEY(DEP_NO)
    -> );
Query OK, 0 rows affected (0.52
```

sec) En este caso

procederemos:

```
mysql> INSERT INTO n2dept
    -> SELECT dep_no, dnombre FROM departamentos
    -> WHERE LENGTH(dnombre)> 8;
Query OK, 3 rows affected (0.40 sec)
Records: 3          Duplicates: 0
                    Warnings: 0


mysql> SELECT *
    -> FROM n2dept;

+--------+---------------+
| DEP_NO | NOMBRE                  |
+--------+---------------+
|          10 | CONTABILIDAD          |
|          20 | INVESTIGACION |
|          40 | PRODUCCION            |
+--------+---------------+
3 rows in set (0.00 sec)
```

2.2 – Modificaciones con subconsultas

En ocasiones la condición que deben cumplir las filas que deseamos modificar implica realizar una subconsulta a otras tablas. En estos casos se incluirá la subconsulta en la condición y los operadores necesarios tal como estudiamos en el apartado correspondiente del tema SUBCONSULTAS. La subconsulta estará en la cláusula WHERE.
Esta subconsulta tiene las siguientes limitaciones:

- La tabla destino no puede aparecer en la consulta.
- No se puede incluir una cláusula ORDER BY en la consulta.

Por ejemplo, supongamos que se desea elevar en 500 Euros. el salario de todos los empleados cuyo departamento no esté en MADRID

```
mysql> UPDATE empleados
    ->     SET salario = salario + 500
    ->     WHERE dep_no NOT IN (SELECT dep_no
    ->                                     FROM departamentos
    ->                                     WHERE localidad <> 'MADRID');

Query OK, 4 rows affected (0.05 sec) Rows
matched: 4          Changed: 4
                    Warnings: 0
```

2.3 – Eliminaciones con subconsultas

En ocasiones la condición que deben cumplir las filas que deseamos eliminar implica realizar una subconsulta a otras tablas. En estos casos se incluirá la subconsulta en la condición y los operadores necesarios tal como estudiamos en el apartado correspondiente del tema subconsultas. Ésta aparecerá en

la cláusula WHERE con las mismas restricciones que en las modificaciones:

- La tabla destino no puede aparecer en la consulta.
- No se puede incluir una cláusula ORDER BY en la consulta.

Supongamos que queremos eliminar de la tabla departamentos aquellos que no tienen ningún empleado.

```
mysql> DELETE FROM departamentos
    -> WHERE dep_no NOT IN
                    (SELECT DISTINCT dep_no FROMempleados);
Query OK, 1 row affected (0.02 sec)

mysql> SELECT *
    -> FROM departamentos;
+--------+---------------+-----------+
| DEP_NO | DNOMBRE       | LOCALIDAD |
+--------+---------------+-----------+
|     10 | CONTABILIDAD  | BARCELONA |
|     20 | INVESTIGACION | VALENCIA  |
|     30 | VENTAS        | MADRID    |
+--------+---------------+-----------+
3 rows in set (0.00 sec)
```

El siguiente ejemplo eliminará los departamentos que tienen menos de tres empleados.

Nota: esta orden la ejecutamos con las tablas departamentos2 y empleados2 que tiene borrado en cascada (con empleados y departamentos no sería posible por la restricción de integridad)

```
mysql> DELETE FROM departamentos2
    -> WHERE dep_no IN (SELECT dep_no
  ->                    FROMempleados2
  ->                    GROUP BYdep_no
    ->                      HAVING count(*) <
3); Query OK, 1 row affected (0.39 sec)

mysql> SELECT *
    -> FROM departamentos2;
+--------+--------------+-----------+
| DEP_NO | DNOMBRE      | LOCALIDAD |
+--------+--------------+-----------+
|     10 | CONTABILIDAD | BARCELONA |
|     30 | VENTAS       | MADRID    |
|     40 | PRODUCCION   | SEVILLA   |
+--------+--------------+-----------+
3 rows in set (0.00 sec)
```

El ejemplo anterior borrará los departamentos que tengan menos de tres empleados pero, para que entre en la lista de selección, el departamento deberá tener al menos un empleado. Por tanto, los empleados que no tengan ningún empleado no se borrarán. Para evitar esto podemos cambiar la condición indicando que se borren aquellos departamentos que no están en la lista de departamentos con tres o más empleados.

```
mysql> DELETE FROM departamentos2
    -> WHERE dep_no NOT IN (SELECT dep_no
    -> FROM empleados2

    -> GROUP BY dep_no
    -> HAVING count(*) >= 3);
Query OK, 2 rows affected (0.12sec)

mysql> SELECT *
    -> FROM departamentos2;
+--------+--------------+----------+
```

```
| DEP_NO| DNOMBRE              | LOCALIDAD|
+-------+-------------+----------+
|      10 | CONTABILIDAD | BARCELONA |
|      30 | VENTAS              | MADRID          |
+-------+-------------+----------+
2 rows inset (0.00 sec)
```

Esta última orden borrará todos los departamentos que tienen: ninguno, uno o dos empleados.

Se pueden utilizar subconsultas anidadas a varios niveles, pero respetando la siguiente **restricción: la tabla destino no puede aparecer en la cláusula FROM de ninguna de las subconsultas que intervienen en la selección. Si se permiten referencias externas,** como en el siguiente ejemplo:

```
mysql> DELETE FROM departamentos
    -> WHERE NOT EXISTS (SELECT *
    ->                          FROM empleados
    ->                          WHERE empleados.dep_no=departamentos.dep_no);
Query OK, 0 rows affected (0.00 sec)
```

En estos casos la subconsulta con la referencia externa realiza la selección sobre la tabla destino antes de que se elimine ninguna fila.

Nota: debe tenerse en cuenta que algunos productos comerciales permiten saltar esta restricción pero otros no.

3 - Vistas

Podemos definir una vista como una consulta almacenada en la base de datos que se utiliza como una tabla virtual. Se define asociadas a una o varias tablas y no almacena los datos sino que trabaja sobre los datos de las tablas sobre las que está definida, estando así en todo momento actualizada.

3.1 - ¿Qué son las vistas y para qué sirven?.

Se trata de una perspectiva de la base de datos o ventana que permite a uno o varios usuarios ver solamente las filas y columnas necesarias para su trabajo.

Entre las ventajas que ofrece la utilización de vistas cabe destacar:

- Seguridad y confidencialidad: ya que la vista ocultará los datos confidenciales o aquellos para los que el usuario no tenga permiso.
- Comodidad: ya que solamente muestra los datos relevantes, permitiendo, incluso trabajar con agrupaciones de filas como si se tratase de una única fila o con composiciones de varias tablas como si se tratasede una única tabla.
- Independencia respecto a posibles cambios en los nombres de las columnas, de las tablas, etcétera.

Por ejemplo, la siguiente consulta permite al departamento de VENTAS realizar la gestión de sus empleados ocultando la información relativa a los empleados de otros departamentos.

```
SELECT *
FROM EMPLEADOS
WHERE dep_no=30;
```

La siguiente consulta permite a cualquier empleado de la empresa obtener información no confidencial de cualquier otro empleado ocultando las columnas SALARIO y COMISION:

```
SELECT emp_no, apellido, oficio, director, fecha_alta, dep_no FROM empleados;
```

Para ello crearemos vistas y permitiremos a los usuarios tener acceso a las vistas sin tenerlo de la tabla completa.

3.2 - Creación y utilización devistas

Como hemos dicho son tablas virtuales resultado de una consulta realizadas sobre tablas ya existentes. Las vistas no ocupan espacio en la base de datos ya que lo único que se almacena es la definición de la vista. El gestor de la base de datos se encargará de comprobar los comandos SQLque hagan referencia a la vista, transformándolos en los comandos correspondientes referidos a las tablas originales, todo ello de forma transparente para el usuario.

3.2.1 – Formato de la creación de vistas.

Para crear una vista se utiliza el comando CREATE VIEW según el siguiente formato genérico:

```
CREATE VIEW NombreVista
                [(DefiniciónColumna [,DefiniciónColumna….] )]
AS Consulta;
```

Notación: la lista de columnas que define las columnas de la vista es opcional.

donde **NombreVista……..** es el nombre que tendrá la vista que se va a crear.
DefinicionColumnas…….. permite especificar un nombre y su correspondiente tipo para cada columna de la vista.
Si no se especifica, cada columna quedará con el nombre o el alias correspondiente y el tipo asignado por la consulta.
Consulta…….. en una consulta cuy o resultado formará la vista.

El siguiente ejemplo crea la vista emple_dep30 para la gestión de los empleados del departamento 30 mencionada en el apartado anterior.

```
mysql> CREATE VIEW emple_dep30 AS
    ->     SELECT * FROM EMPLEADOS
    ->     WHERE DEP_NO = 30;
Query OK, 0 rows affected (0.55 sec)

CREATE VIEW
emple_dep30 AS SELECT *
FROM EMPLEADOS WHERE
DEP_NO = 30;
```

A continuación se muestra la sentencia que crea la vista datos_emple que contiene información de todos los empleados ocultando la información confidencial.

```
mysql> CREATE VIEW datos_emple AS
    -> SELECT emp_no, apellido, oficio,director,
                    fecha_alta, dep_no
    -> FROM empleados;

Query OK, 0 rows affected (0.00 sec)
```

Las vistas pueden a su vez definirse sobre otras vistas. Si ya tenemos creada las vista datos_emple, podríamos crear otra vista sobre ella:

```
mysql> CREATE VIEW datos_emple_10 AS
    ->     SELECT *
    ->     FROMdatos_emple
    ->     WHEREdep_no=10;
Query OK, 0 rows affected (0.00 sec)
```

3.2.2 – Utilización de vistas

Una vez definida puede ser utilizada para consultas de igual forma que una tabla.
Con algunas restricciones, todos los formatos de selección vistos para las tablas son aplicables para la selección de filas en las vistas.

Por ejemplo:

```
mysql> SELECT * FROM datos_emple;

+--------+----------+-----------+----------+------------+--------+
| 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 (0.01 sec)
```

También podemos seleccionar solo algunas columnas y poner una condición

```
mysql> SELECT apellido, director
    ->     FROM datos_emple
    ->     WHERE oficio = 'VENDEDOR';

+----------+----------+
| apellido | director |
+----------+----------+
| ALONSO   |     7698 |
| MARTIN   |     7698 |
| CALVO    |     7698 |
+----------+----------+
3 rows in set (0.00 sec)
```

Pero debe tenerse en cuenta que si al definir la vista hemos indicado nuevos nombres para columnas y expresiones si podremos hacer referencia a ellos en las sentencias de selección, pero si hemos omitido la definición de las columnas y en la sentencia de creación hemos realizado la selección de todas las columnas (creada con select *) solo puede hacerse una selección de todas las columnas de la vista (con *)

La vista emple_dep30 la creamos sin especificar nuevo nombre para las columnas de la vista y con una sentencia select *. Podemos obtener los datos de la vista si escribimos:

```
mysql> SELECT * FROM emple_dep30;

+-------+----------+---------+---------+------------+---------+--------+--------+
|EMP_NO | APELLIDO | OFICIO  |DIRECTOR | FECHA_ALTA | SALARIO |COMISION| DEP_NO |
+-------+----------+---------+---------+------------+---------+--------+--------+
| 7499| ALONSO          | VENDEDOR|          7698 | 1981-02-23 | 1400.00 | 400.00 |
                          30 |
| 7654| MARTIN          | VENDEDOR|          7698 | 1981-09-28 | 1500.00 |1600.00 |
                          30 |
| 7698 | GARRIDO  | DIRECTOR|                7839 | 1981-05-01 | 3850.12 |                  NULL |
                                                30 |
| 7844| CALVO           | VENDEDOR|          7698 | 1981-09-08 | 1800.00 |                  0.00 |
                          30 |
+-------+----------+---------+---------+------------+---------+--------+--------+
4  rows in set (0.06 sec)
```

Pero no podemos referenciar las columnas en la selección al recuperar datos de la vista:

```
mysql> SELECT apellido FROM emple_dep30;
ERROR 1054 (42S22): Unknown column 'apellido' in 'fieldlist'
```

3.2.3 - Restricciones para la creación y utilización de vistas

No se puede usar la cláusula ORDER BY en la creación de una vista ya que las filas de una tabla no están ordenadas (la vista es una tabla virtual). No obstante, si se puede utilizar dicha cláusula a la hora de recuperar datos de la vista.

Es obligatorio especificar la lista de nombres de columnas de la vista o un alias cuando la consulta devuelve funciones de agrupamiento como SUM, COUNT, etcétera y posteriormente quiere hacerse referencia a ellas.

Pueden utilizarse funciones de agrupación sobre columnas de vistas que se basan a su vez en funciones de agrupación lo que permite resolver los casos en los que un doble agrupamiento que no está permitido por el estándar. Así creamos una vista con una primera función de agrupación y sobre ella aplicamos la segunda función de agrupación, obteniendo el resultado deseado.

3.2.4 - Ejemplos creación y utilización de vistas

Como hemos dicho una vez creada la vista se puede utilizar como si se tratase de una tabla (observando las restricciones anteriores). Veamos lo que podemos hacer con las vistas con los ejemplos.

1- El siguiente ejemplo crea la vista *datos_vendedores* que muestra solamente las columnas *emp_no, apellido, director, fecha_alta, dep_no,* de aquellos empleados cuyo oficio es VENDEDOR.

```
mysql> CREATE VIEW datos_vendedores
    ->    (num_vendedor, apellido, director, fecha_alta, dep_no)AS
    ->    SELECT emp_no, apellido, director, fecha_alta,dep_no
    ->    FROM empleados
    ->    WHERE oficio = 'VENDEDOR';
Query OK, 0 rows affected (0.00 sec)
```

Los datos accesibles mediante la vista creada serán:

```
mysql> SELECT       *
    ->    FROM datos_vendedores;
+--------------+----------+----------+------------+--------+
| num_vendedor | apellido | director | fecha_alta | dep_no|
+--------------+----------+----------+------------+--------+
|          7499 | ALONSO       |       7698 | 1981-02-23 |          30 |
```

```
|          7654 | MARTIN          |          7698 | 1981-09-28 |                30 |
|          7844 | CALVO           |          7698 | 1981-09-08 |                30 |
+--------------+----------+----------+------------+--------+
3 rows in set (0.02 sec)
```

2- También se pueden crear vistas a partir de consultas que incluyen agrupaciones, como en el siguiente ejemplo:

```
mysql> CREATE VIEW resumen_dep1
    ->     (dep_no, num_empleados, suma_salario, suma_comision)AS
    ->     SELECT dep_no, COUNT(emp_no),SUM(salario),
                    SUM(IFNULL(comision,0))
    ->     FROM empleados
    ->     GROUP BY dep_no;
Query OK, 0 rows affected (0.01 sec)
```

En estos casos, cada fila de la vista corresponderá a varias filas en la tabla original tal como se puede comprobar en la siguiente consulta:

```
mysql> SELECT *
    ->     FROM resumen_dep1;
+--------+---------------+--------------+---------------+
| dep_no | num_empleados | suma_salario | suma_comision |
+--------+---------------+--------------+---------------+
|          10 |                          3 |             9800.50 |                    0.00 |
|          20 |                          2 |             4750.00 |                    0.00 |
|          30 |                          4 |             8550.12 |                 2000.00 |
+--------+---------------+--------------+---------------+
3 rows in set (0.02 sec)
```

Normalmente la mayoría de las columnas de este tipo de vistas corresponden a funciones de columna tales como SUM, AVERAGE, MAX, MIN, etcétera. Por ello el estándar SQL establece en estos casos la obligatoriedad de especificar la lista de columnas o de alias si posteriormente quiere hacerse referencia a ellas. Aunque algunos gestores de bases de datos permiten saltar esta restricción. No es aconsejable ya que las columnas correspondientes de la vista quedarán con nombres como COUNT(EMP_NO), *SUM(SALARIO), SUM(COMISION)* lo cual no resulta operativo para su posterior utilización.

```
mysql> SELECT dep_no, num_empleados, suma_salario
    -> FROM resumen_dep1;
+--------+---------------+--------------+
| dep_no | num_empleados | suma_salario|
+--------+---------------+--------------+
|          10 |                          3 |             9800.50 |
|          20 |                          2 |             4750.00 |
|          30 |                          4 |             8550.12 |
+--------+---------------+--------------+
3 rows in set (0.14 sec)
```

3- Sobre esta vista, con una función de agrupación, podemos hacer otra función de agrupación, por ejemplo obtener el departamento con mayor suma de salarios para sus empleados:

Creamos la vista resumen_dep2con dos totales resultado e aplicar funciones de grupo

```
mysql> CREATE VIEW resumen_dep2
    ->ASSELECTdep_no,COUNT(emp_no)"num_empleados",
```

```
                SUM(salario) "suma_salario",
    -> FROM empleados
    -> GROUP BY dep_no;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT dep_no, num_empleados, suma_salario
    -> FROM resumen_dep2;
+--------+---------------+--------------+
| dep_no | num_empleados | suma_salario |
+--------+---------------+--------------+
|     10 |             3 |      9800.50 |
|     20 |             2 |      4750.00 |
|     30 |             4 |      8550.12 |
+--------+---------------+--------------+
3 rows in set (0.00 sec)
```

Y sobre ella volvemos a aplicar una función de grupo para hallar el máximo de las sumas de los salarios

```
mysql> SELECT MAX(suma_salario)
    -> FROM resumen_dep2;
+------------------+
| MAX(suma_salario) |
+------------------+
|          9800.50 |
+------------------+
1 row in set (0.00 sec)
```

Y también podemos obtener el departamento el que pertenece este salario máximo

```
mysql> SELECT dep_no, dnombre
    -> FROM departamentos
    -> WHERE dep_no = (SELECT dep_no
    ->                 FROM resumen_dep2
    ->                 WHERE suma_salario=(SELECT MAX(suma_salario)
    ->                                     FROM resumen_dep2));

+--------+--------------+
| dep_no | dnombre      |
+--------+--------------+
|     10 | CONTABILIDAD |
+--------+--------------+
1 row in set (0.00 sec)
```

4 –Así mismo, se pueden crear vistas que incluyan todas o varias de las posibilidades estudiadas. Por ejemplo la siguiente vista permite trabajar con datos de dos tablas, agrupados y seleccionando las filas que interesan (en este caso todos los departamentos que tengan más de dos empleados):

```
mysql> CREATE VIEW resumen_emp_dep
    ->     (departamento,      num_empleados, suma_salario) AS
    ->       SELECT dnombre, COUNT(emp_no), SUM(salario)
    ->       FROM empleados, departamentos
    ->       WHERE empleados.dep_no = departamentos.dep_no
    ->       GROUP BY empleados.dep_no,dnombre
    ->       HAVING COUNT(*) > 2;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT        departamento, num_empleados
    -> FROM resumen_emp_dep;
+--------------+---------------+
| departamento | num_empleados|
+--------------+---------------+
| CONTABILIDAD |                3 |
| VENTAS       |                4 |
+--------------+---------------+
2 rows in set (0.02 sec)
```

3.3 - Eliminación de vistas

3.3.1 - Formato de borrado de vistas

La sentencia DROP VIEW permite eliminar la definición de una vista.

```
DROP VIEW [IF EXISTS] NombreVista
 [RESTRICT | CASCADE ]
```

La cláusula IF EXISTS previene los errores que puedan producirse si no existe la tabla que queremos borrar
La cláusula CASCADE Y RESTRICTestán permitidas pero no implementada en la versión 5.

3.3.2 - Ejemplos de borrado de vistas

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

3.3.3 - Borrado de las tablas o vistas asociadas a una vista

Si se borran las tablas a las vistas sobre las que están definidas las vistas la vista se queda invalida (no se borra su definición pero no está utilizable).

```
mysql> SELECT *
    -> FROM datos_emple_10;

+--------+----------+-----------+----------+------------+--------+
| emp_no | apellido | oficio                | director | fecha_alta | dep_no |
+--------+----------+-----------+----------+------------+--------+
|   7521 | LOPEZ    | EMPLEADO  |     7782 | 1981-05-08 |     10 |
|   7782 | MARTINEZ | DIRECTOR  |     7839 | 1981-06-09 |     10 |
|   7839 | REY      | PRESIDENTE |    NULL | 1981-11-17 |     10 |
+--------+----------+-----------+----------+------------+--------+
3 rows in set (0.05 sec)
```

Si borramos datos_emple sobre la que está definida datos_emple_10 esta pasará a estar inválida.

```
mysql> DROP VIEW IF EXISTSdatos_emple;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT *
    -> FROM datos_emple_10;
ERROR 1356 (HY000): View 'test.datos_emple_10' references invalid table(s) or column(s)
```

PARTE III

Tema 10. Creación de procedimientos

Un procedimiento, al igual que una función, es una rutina formada por un conjunto de sentencias SQL. Un procedimiento tiene un determinado nombre y puede usar una serie de parámetros que pueden ser de tres tipos IN (el procedimiento recibe el parámetro y no le modifica, le usa para consultar y utilizar su valor), OUT (el procedimiento únicamente puede escribir en el parámetro, no puede consultarle), INOUT (el procedimiento recibe el parámetro y puede consultarle y modificarle). Los parámetros OUT o INOUT se usan cuando se desea que un procedimiento nos devuelva valores en determinadas variables.

Los procedimientos, una vez creados, quedan almacenados en el servidor MySQL y pueden ser llamados por cualquier usuario con autorización para ello y, por supuesto, por el usuario que los creó. Los procedimientos se ejecutan con la instrucción:

CALL nomProc (parámetros);

Dado que un procedimiento consta de varias instrucciones, desde el cliente MySQL no hay que enviar todas esas instrucciones (una a una) cuando se quieren ejecutar sino simplemente se envía la llamada al procedimientos. Por tanto, con los procedimientos se agiliza el envío de instrucciones desde los clientes al servidor (no recibiendo éste tantas peticiones de tareas). Como contrapartida, el servidor tiene una mayor carga de trabajo al tener que buscar y decodificar los procedimientos almacenados cuando son invocados desde los clientes.
Los procedimientos y las funciones aumentan la seguridad del sistema ya que se pueden establecer autorizaciones variadas de ejecución para los usuarios sobre los procedimientos y funciones que se deseen. Además los usuarios utilizarán los procedimiento y funciones onstruidos para realizar las tareas cotidianas y no tendrán que construir esos procesos sobre la base de las sentencias que los forman, con el posible riesgo de alteraciones no deseadas de los datos por operaciones indebidamente realizadas.

Los procedimiento y las funciones pueden ser reutilizados para construir otras rutinas. Se pueden crear librerías de procedimientos y funciones que pueden ser invocados desde otros lenguajes de programación.

Todo procedimiento queda asociado a la base de datos abierta cuando se creó el procedimiento. Al ejecutar un procedimiento, el servidor MySQL ejecutará automáticamente una sentencia **USE basedatos**, donde basedatos es la asociada al procedimiento. De esta forma, podemos ejecutar un procedimiento asociado a una base de datos distinta a la que tenemos abierta especificando, en la llamada al procedimiento, un cualificador de la base de datos, de la forma ***CALL nomBASEDATOS.nomProc (parámetros)***. Por ejemplo, si se quiere llamar a un procedimiento llamado **Proce1** que no usa parámetros asociado a tablas de la base de datos **alquileres** y tenemos actualmente abierta la base de datos **test**, para llamar al procedimiento deberemos escribir:

CALL alquileres.Proce1();

Cuando se crea un procedimiento, el servidor MySQL nos devolverá indicaciones sobre los errores que pueda tener o no el procedimiento. Si la sintaxis del procedimiento es correcta, el servidor almacenará dicho procedimiento pero **no le ejecutará en ese momento.** Para jecutarle, una vez que esté almacenado en el servidor, usaremos la sentencia CALL anteriormente indicada.

Si se intenta crear un procedimiento con un nombre que ya existe, el servidor MySQL no lo permite. Si realmente queremos crear un procedimiento con un nombre que ya existe, deberemos eliminar el primero para después crear el nuevo procedimiento.

Sintaxis para crear un procedimiento:
CREATE PROCEDURE *NomProc* ([*parametro*[,...]])
[*caracteristica* ...]
BEGIN
Cuerpo_procedimiento
END

Parámetro tiene la sintaxis:
[IN | OUT | INOUT] *NomParam tipo*

tipo:
Cualquier tipo de dato MySQL

característica:

LANGUAGE SQL | [NOT] DETERMINISTIC | SQL SECURITY {DEFINER |
INVOKER}
| COMMENT '*string*'
cuerpo_procedimiento:

Casi todas las sentencias SQL válidas.
La lista de parámetros debe encerrarse entre paréntesis, aunque el procedimiento no use parámetros. Por defecto, si no se especifica el tipo de paso de parámetro, todo parámetro es IN.
Para especificar otro tipo de paso de parámetro, se escribirá OUT o INOUT delante del nombre del parámetro.
En característica se pueden especificar varios valores. LANGUAGE SQL significa que el cuerpo del procedimiento está escrito en SQL. Por defecto se tiene esa característica para prever la posible construcción de procedimientos almacenados con otros lenguajes como Java. Un procedimiento o función DETERMINISTIC si obtiene siempre lo mismo cuando recibe los mismos valores. La característica SQL SEQURITY sirve especificar si el procedimiento es llamado usando los permisos del usuario que lo creó (DEFINER, que es el valor por defecto), o usando los permisos del usuario que está haciendo la llamada (INVOKER). La característica COMMENT sirve para poner un comentario que aparecerá cuando se ejecute una sentencia para ver el contenido de un procedimiento o de una función:

SHOW CREATE PROCEDURE o
SHOW CREATE FUNCTION.

El cuerpo del procedimiento comienza con una sentencia BEGIN, termina con una sentencia END y constará de varias instrucciones, cada una terminada con el carácter delimitador ';'. Para que durante la creación del procedimiento no se interpreten los caracteres

punto y coma como delimitadores de cada instrucción y que, por tanto, hagan que se ejecute automáticamente cada instrucción, habrá que cambiar temporalmente, antes de empezar a crear el procedimiento, el carácter delimitador o finalizador de instrucciones SQL en MySQL. Para cambiar el carácter delimitador se usa la sentencia DELIMITER. Por ejemplo, para hacer que el delimitador de sentencias sea '//', habrá que ejecutar:

DELIMITER //

Una vez que hayamos creado el procedimiento, podremos volver a establecer el punto y coma como delimitador ya que si no lo hacemos, para ejecutar cualquier sentencia tendremos que finalizarla con los caracteres establecidos como delimitadores, en el ejemplo propuesto '//'. Para que vuelva a ser el carácter ';' delimitador, habrá que ejecutar:

DELIMITER ;

A continuación se muestra un ejemplo básico de procedimiento que permite obtener un listado de todos los clientes de la base de datos **alquileres** y otro listado de todos los automóviles de la misma base de datos. Debemos tener abierta la base de datos alquileres. Suponiendo que hemos establecido que el delimitador de instrucciones mientras se crea el procedimiento sea '//',

deberemos escribir lo siguiente para crear el procedimiento:

```
CREATE PROCEDURE listados()
BEGIN
SELECT * FROM clientes;
SELECT * FROM automoviles;
END//
```

Tras introducir el carácter delimitador y pulsar ENTER, el servidor MySQL recibirá el procedimiento e indicará si éste se ha creado o si tiene algún tipo de error. Si ha sido creado, podremos ejecutarlo en la línea de comandos MySQL o dentro de otro procedimiento con la sentencia:

```
CALL listados();
```

A continuación se muestra un ejemplo en el que se recibe un parámetro de entrada que indica la matrícula de un coche y, a continuación, muestra las características del coche y devuelve en un parámetro de salida el número de contratos realizados sobre ese coche:

```
CREATE PROCEDURE numcontratos(m CHAR(7), OUT c INT)
BEGIN
SELECT * FROM automoviles WHERE matricula=m;
SELECT count(*) INTO c FROM contratos WHERE matricula=m;
END//
```

Como puede verse para asignar en variables o parámetros los valores devueltos por una consulta se usa la sintaxis:

```
SELECT expr1, expr2, .... INTO var1,var2,…
```

Esto deja los valores devueltos en las variables indicadas y no los muestra en pantalla.

Para que se pueda hacer esto, es absolutamente necesario que la consulta devuelva una sola fila. Si devuelve más de una fila no se puede realizar ya que no se puede asignar a una variable una lista de contenidos.

Dado que el procedimiento anterior usa un parámetro de salida, para llamar al procedimiento es necesario pasar una variable donde se cargue el dato devuelto por el procedimiento a través del parámetro correspondiente. Para definir una variable en la línea de comandos (variable temporal de sistema), la variable debe tener el nombre precedido del carácter '@' y se debe usar la sentencia:

SET @NomVar=valor;

Por ejemplo, podemos variables que se comporten como tipo numérico, fecha o cadena de caracteres de la siguiente forma:

SET @Num=9;
SET @Fecha='2004-11-25';
SET @Cadena='mi nombre es luis';

También se puede crear una variable de sistema especificándola como parámetro en la llamada a un procedimiento. Se tenga ya creada la anterior variable @Num o se cree en la llamada al procedimiento, podemos llamar al procedimiento para que devuelva en @Num el número de contratos efectuados sobre el coche de matrícula '3273BGH' mediante:

CALL numcontratos('3273BGH', @Num);

Tras hacer esto, podemos usar el valor cargado en @Num en cualquier sentencia para, por ejemplo, conocer sobre que coches se han realizado más contratos que sobre el anterior o, simplemente para conocer el valor de la variable como se muestra a continuación:

SELECT @Num;
SELECT automóviles.matricula, marca, modelo, count(*) FROM automoviles INNER JOIN contratos ON contratos.matricula=automoviles.matricula GROUP BY automoviles.matricula HAVING count(*)>@Num;

Definición de variables locales al procedimiento:

Dentro de un procedimiento se pueden definir variables locales, es decir, que sólo tienen existencia mientras se ejecuta el procedimiento, después quedan destruidas. Al igual que las demás sentencias del procedimiento, la declaración de variables debe estar dentro del bloque BEGIN .... END. Para definir cualquier variable se usa la sentencia:

DECLARE nombre tipo[DEFAULT valor];

Donde tipo es cualquiera de los tipos admitidos por MySQL. Por cada variable declarada hay que usar una sentencia DECLARE. Para modificar el valor de una variable o de un parámetro mediante una asignación debe usarse la sentencia:

SET variable=expresión;

El siguiente ejemplo, muestra un procedimiento que lista los vehículos con menos de 2500 kilómetros y, después, los vehículos con menos kilómetros que los anteriores más 5000 kilómetros:

```
CREATE PROCEDURE usovariable()
BEGIN
DECLARE a INT;
SET a=2500;
SELECT CONCAT('AUTOMOVILES CON MENOS DE ',a,' KILOMETROS') AS
'...';
SELECT * FROM automoviles WHERE kilometros<a;
SET a=a+5000;
SELECT CONCAT('AUTOMOVILES CON MENOS DE ',a,' KILOMETROS') AS
'...';
SELECT * FROM automoviles WHERE kilometros<a;
END//
```

Creación de funciones

Las funciones son rutinas compuestas por varias sentencias SQL que devuelven un resultado. La forma de crear una función es similar a la de crear un procedimiento. Las funciones, una vez creadas, quedan almacenadas en el servidor y pueden ser invocadas en cualquier momento por cualquier cliente MySQL. Respecto de los procedimientos, las funciones resentan las siguientes diferencias:

- Las funciones devuelven siempre un dato a través de una sentencia RETURN. El dato se
- corresponde con un tipo declarado para la función.

- Las funciones no pueden trabajar con parámetros OUT o INOUT.

- Las funciones son llamadas a ejecución, al igual que las funciones internas de MySQL,
- escribiendo su nombre y la lista de parámetros pasados a la función encerrados entre paréntesis. Por tanto no usa una sentencia de llamada como la sentencia CALL de llamada a los procedimientos.

- Las funciones podrán ser llamadas desde cualquier sentencia SQL como SELECT, UPDATE, INSERT, DELETE. Los procedimientos nunca pueden ser llamados a ejecución dentro de otra sentencia.

La sintaxis para crear una función es la siguiente:

```
CREATE FUNCTION nomFuncion([parametro[,...]])
RETURNS tipo
[caracteristica ...]
BEGIN
CuerpoRutina
END
```

Con respecto a los procedimientos aparece la cláusula **RETURNS tipo** que sirve para indicar el tipo de dato resultado que devuelve la función. Para devolver un resultado, la función debe

incluir dentro del cuerpo de la rutina, la sentencia **RETURN expresion**, debiendo ser expresión del mismo tipo que la función. Generalmente la sentencia RETURN para devolver un resultado es la última del cuerop de la función.

Como un ejemplo básico vamos a presentar como se realizaría una función que recibe una cadena de caracteres y devuelve el mensaje HOLA seguido de esa cadena de caracteres. Se supone que se ha establecido // como delimitador:

```
CREATE FUNCTION Saludo (s CHAR(20))
RETURNS CHAR(50)
BEGIN
DECLARE a CHAR(50) DEFAULT 'HOLA, ';
SET a=CONCAT(a, s, '!');
RETURN a;
END //
```

Una vez creada la función, podemos llamar a esa función dentro de una SELECT cualquiera como se presenta a continuación obteniendo el resultado mostrado:

```
mysql> SELECT saludo ('caracola');
+----------------------+
|saludo ('caracola') |
+----------------------+
| HOLA, caracola! |
+----------------------+
```

Otro ejemplo de función es una función que recibe una fecha y devuelve el año correspondiente a esa fecha (suponiendo que no existe ya una función para realizarlo). En la función convertiremos primero la fecha en cadena de caracteres, después extraeremos los cuatro caracteres de la izquierda (corresponden al año) y convertiremos esa cadena en número entero sin signo. Ese número será lo que devuelve la función:

```
CREATE FUNCTION anno(f DATE)
RETURNS INT
BEGIN
DECLARE an CHAR(10);
DECLARE a INT;
SET an=CONVERT(f,CHAR);
SET an=LEFT(an,4);
SET a=CONVERT(an,UNSIGNED);
RETURN a;
END//
```

Podemos probar la función pasándole la fecha actual para conocer cual será el año próximo en forma numérica:

```
mysql> SELECT anno(curdate())+1 AS 'año próximo';
+---------------+
| año próximo |
+---------------+
| 2011 |
+---------------+
```

1 row in set (0.00 sec)

Mostrar el contenido de un procedimiento o función:

SHOW CREATE {PROCEDURE | FUNCTION} *sp_name*

Eliminar un procedimiento o función:

DROP {PROCEDURE | FUNCTION} [IF EXISTS] *sp_name*

La cláusula IF EXIST sirve para que la instrucción no devuelva un código de error cuando el procedimiento o función no existe.

Modificar la característica de un procedimiento o función

ALTER {PROCEDURE | FUNCTION} *nombre* [*característica* ...]

Sentencias de control de flujo

En SQL de MySQL se dispone de varias sentencias de control de flujo o estructuras de control de flujo. Estas sentencias sirven para codificar estructuras de decisión y repetitivas en una función o en un procedimiento. Las sentencias de control de flujo disponibles son: **IF, CASE, LOOP, WHILE, ITERATE, y LEAVE.** No se admite la sentencia **FOR** que si es admitida en SQL para otros gestores de bases de datos.

Dentro de cada una de estas estructuras puede haber tantas instrucciones SQL como queramos e incluso otras estructuras anidadas.

IF
IF *condicion* THEN *sentencias* [ELSEIF *otra_condicion* THEN *sentencias*]
...
[ELSE *sentencias*]
END IF;

Si la condición del IF es verdadera se ejecutan las correspondientes sentencias. Si no es verdadera se evalúa la condición del primer ELSEIF y si es verdadera, se ejecutan las entencias correspondientes y así sucesivamente con todos los ELSEIF. Si ninguna de las condiciones es verdadera, caso de haber ELSE, se ejecutan las sentencias asociadas al ELSE.

CASE

La sentencia CASE sirve para implementar una estructura de decisión múltiple con dos posibles sintaxis. En la primera sintaxis se ejecutan las sentencias correspondientes al primer valor que sea igual a la expresión. Si ninguno de los valores es igual a la expresión, se ejecutan las sentencias que hay dentro de ELSE, caso de que hubiera ELSE. En la segunda sintaxis se ejecutan las sentencias correspondientes a la primera condición que se cumpla y si no se cumpliera ninguna de las condiciones, se ejecutarían las sentencias que hay dentro del ELSE, caso de que haya ELSE.

Primera Sintaxis:

CASE ***expresion***
WHEN ***valor1*** **THEN** ***sentencias1***
[WHEN ***valor2*** **THEN** ***sentencias2*****]**
……..
[WHEN ***valorN*** **THEN** ***sentenciasN*****]**
[ELSE ***sentencias_else*****]**
END CASE;

Segunda Sintaxis:

CASE
WHEN ***condicion1*** **THEN** ***sentencias1***
[WHEN ***condicion2*** **THEN** ***sentencias2*****]**
……..
[WHEN ***condicionN*** **THEN** ***sentenciasN*****]**
[ELSE ***sentencias_else*****]**
END CASE;

LOOP

Permite realizar un bucle repetitivo que no tiene ninguna condición de salida. Para salir de un bucle LOOP es necesario incluir la sentencia de salida forzada de bucle LEAVE que veremos continuación. La sintaxis para la sentencia LOOP es:

[etiqueta:]LOOP
statement_list
END LOOP [etiqueta];

La etiqueta es una marca que sirve para que se pueda saltar al comienzo o al final del bucle con las sentencias LEAVE o ITERATE. La que hay la principio y al final del bucle debe ser la misma.

LEAVE
Permite salir de un bucle cualquiera indicando un salto a la etiqueta que marca el final del bucle.una etiqueta. La etiqueta debe indicarse antes de comenzar el bucle en la forma ***NombreEtiqueta:***. La sintaxis de la sentencia es: **LEAVE** ***label***

ITERATE
ITERATE sólo puede usarse dentro de LOOP, REPEAT, y WHILE. La sentencia ITERATE provoca que se produzca un salto para reiniciarse de nuevo el bucle desde su primera sentencia. Para ello, es necesario que el bucle correspondiente este marcado con una tiqueta. La sintaxis de esta sentencia es: ***ITERATE etiqueta***

El siguiente ejemplo muestra como se pueden usar las sentencias LEAVE y ITERATE dentro de un bucle LOOP para salir del bucle o volver al inicio del bucle:

CREATE PROCEDURE bucleloop(p1 INT, OUT p2 INT)
BEGIN
SET p2=0;
etiq1: LOOP
SET p1 = p1 + 1;
SET p2=p2+1;

IF p1 < 10 THEN
ITERATE etiq1;
END IF;
LEAVE etiq1;
END LOOP etiq1;
SET p2 = p1;
END
REPEAT

Permite implementar una estructura repetitiva del tipo ***repetir…hasta***. En esta estructura repetitiva se empieza ejecutando las sentencias que están dentro de REPEAT y, al final, se finaliza si se cumple la condición indicada en UNTIL. Si la condición es verdadera, se sale del bucle, y si es falsa se vuelve al comienzo del bucle.

La sintaxis es:
[*etiqueta*:] REPEAT
sentemcias
UNTIL *condicion*
END REPEAT [*end_label*];

Ejemplo:

CREATE PROCEDURE buclerepeat(p1 INT, OUT p2 INT)
BEGIN
SET p2 = 0;
REPEAT
SET p2 = p2 + 1;
SET p1=p1-1;
UNTIL p2 > p1 END REPEAT;
END

WHILE
Permite implementar un bucle del tipo mientras. En este bucle se evalúa inicialmente la condición y, si esta se cumple, se ejecutan las sentencias que hay dentro del bucle y cuando se llega a la última se vuelve a evaluar la condición del WHILE, repitiéndose el proceso anterior si la condición se cumple. Cuando la condición sea falsa, se produce la salida del bucle.
La sintaxis es:
[*begin_label*:] WHILE *condicion* DO
sentencias
END WHILE [*end_label*];

Ejemplo:
CREATE PROCEDURE bucle()
BEGIN
DECLARE v1 INT DEFAULT 5;
WHILE v1 > 0 DO
SET v1 = v1 - 1;
END WHILE;
END

MANIPULADORES DE ERROR

Cuando se ejecuta una sentencia SQL, el servidor devuelve un código de error (numérico) relativo a esa sentencia. Por ejemplo, se devuelve código de cuando se ejecuta una sentencia que trata de insertar una fila con un valor que ya existe en la columna que sea clave primaria. Si la sentencia no ha generado ningún error, entonces devuelve 0 como código de error. Si una sentencia forma parte de un procedimiento o función y genera un error, entonces se termina automáticamente la ejecución del procedimiento o función. Esto, generalmente, es un grave problema ya que es normal que, aunque una sentencia de una rutina produzca error, se desee procesar las siguientes sentencias de la rutina. Para solucionar este problema, MySQL permite usar **manipuladores de errores** que sirven para indicar como debe responder el servidor MyQL, en procedimientos y funciones, a situaciones de error.
Para trabajar con manipuladores de error hay que declarar estos manipuladores e indicar sobre que sentencia o sentencias actuarán. Al declarar un manipulador puede haberse declarado antes la condición ante la que responderá el manipulador, dándole un nombre a esa condición. Para declarar condiciones de manipulación de excepciones hay que usar la sintaxis:

DECLARE *nombre_condicion* CONDITION FOR {SQLSTATE *valor_estado* | *código_error_MySQL};

Por ejemplo, si insertamos una fila en una tabla y esa fila no se puede insertar porqué el valor de la clave primaria ya existe, MySQL devuelve una indicación de error cuyo valor de estado es '23000' y cuyo código de error es 1062 (los valores de estado devueltos para las sentencias ejecutadas son cadenas de caracteres mientras que los códigos de error son enteros. Para crear una condición de manipulación de excepciones sobre esa situación de error, declararemos el manipulador de la forma:

DECLARE *claverepe* CONDITION FOR SQLSTATE '23000';

Para declarar un manipulador de excepción hay que usar la sintaxis:

DECLARE *tipo_manipulador* HANDLER FOR *valor_condicion*[,...] *sentencia;*
Tipo_manipiulador:
CONTINUE
| EXIT
valor_condicion:
SQLSTATE *valor_estado*
| *nombre_condicion*
| SQLWARNING
| NOT FOUND
| SQLEXCEPTION
| *código_error_MySQL*

Si se produce cualquiera de las condiciones de error declaradas en el manipulador, se ejecutará la sentencia especificada para el manipulador. Un manipulador de tipo CONTINUE hace que prosiga la ejecución de la siguiente sentencia a aquella donde se hay producido un error controlado por el manipulador. Un manipulador de tipo EXIT hace que se termine el bloque en el que se encuentra la sentencia que ha producido el error controlado por el manipulador y, por tanto, termine la rutina en la que se encuentre.

- SQLWARNING se usa para referenciar a todos los valores de estado que comienzan por 01.
- NOT FOUND se usa para referenciar a todos valores de estado que comienzan por 02.
- SQLEXCEPTION se usa para referenciar a todos los valores de estado que no son

controlados por SQLWARNING y por NOT FOUND.

Ejemplos:
Dado el siguiente procedimiento:
create procedure manipuladores()
begin
begin
insert into automoviles (matricula,marca, modelo) values('1234BMY','audi','A6');
*select * from clientes;*
select count(*) from clientes;
end;
*select * from contratos;*
end//

Suponiendo que hacemos la llamada al procedimiento y la matrícula a insertar ya existe en la tabla, no se ejecutará ninguna de las sentencias que siguen a la sentencia INSERT que da error.

mysql> call manipuladores()//
ERROR 1062 (23000): Duplicate entry '1234BMY' for key 1

Sabiendo que un error de inserción por clave duplicada da el valor de estado 23000 y el código de error 1062, modificaríamos el procedimiento anterior para que controlase ese error de una forma similar a la siguiente:

create procedure manipuladores()
begin
declare CONTINUE HANDLER FOR SQLSTATE '23000' select 'hubo error';
begin
insert into automoviles (matricula,marca, modelo) values('1234BMY','audi','A6');
*select * from clientes;*
select count(*) from clientes;
end;
*select * from contratos;*
end//

Ahora, al ejecutar el procedimiento, se ejecuta la sentencia asociada al manipulador cuando se produce el error, es decir, se escribe el mensaje ***hubo error*** y se continua (tipo CONTINUE) con la sentencia que hay a continuación de la sentencia INSERT que produjo error. Si el manipulador fuese tipo EXIT se ejecutaría la sentencia asociada al manipulador pero no las siguientes y se iría al final del procedimiento.

La declaración del manipulador podríamos haberla hecho usando el código de error y no el valor de estado de la sentencia. En su lugar podríamos haber puesto:

DECLARE CONTINUE EXIT HANDLER FOR 1062 select 'hubo error';

O incluso podíamos haber definido previamente una condición de error llamada **manija**, para después declarar un manipulador para esa condición de error:

DECLARE manija CONDITION FOR 1062;
DECLARE CONTINUE HANDLER FOR manija select 'hubo error';

Puedes obtener todos los códigos de error MySQL en
http://dev.mysql.com/doc/mysql/en/Errorhandling.
Html

Cursores

Un cursor es una consulta declarada que provoca que el servidor, cuando se realiza la operación abrir cursor, cargue en memoria los resultados de la consulta en una tabla interna. Teniendo abierto el cursor, es posible, mediante una sentencia FETCH, leer una a una las filas correspondientes al cursor y, por tanto, correspondientes a la consulta definida. Los cursores deben declararse después de las variables locales. Se declaran mediante una sentencia:

DECLARE nomCursor CURSOR FOR SELECT ……;

En la SELECT de declaración del cursor puede haber cualquier cláusula utilizada dentro de una SELECT excepto la cláusula INTO. En un procedimiento o en una función podemos definir tantos cursores como necesitemos. El siguiente ejemplo crea un procedimiento que obtiene en una variable S la suma de los kilómetros correspondientes a los 3 primeros automóviles SEAT obtenidos en una consulta:

```
CREATE PROCEDURE curdemo(OUT S INT)
BEGIN
DECLARE k INT;
DECLARE cur1 CURSOR FOR SELECT kilometros FROM automoviles;
OPEN cur1;
SET S=0;
FETCH cur1 INTO k;
SET S=S+k;
FETCH cur1 INTO k;
SET S=S+k;
END
```

Para abrir el cursor o, lo que es lo mismo, hacer que los resultados de la consulta asociada al cursor queden cargados en memoria, se usa la sentencia:

OPEN *nomCursor;*

La primera vez que se lea sobre el cursor se leerá la primera fila de la consulta, la segunda vez se leerá sólo la segunda fila y así sucesivamente. Si quisiéramos volver a leer desde la primera, tendríamos que cerrar el cursor y abrirlo nuevamente. Para leer la fila actualmente disponibles en el cursor, se debe usar la sintaxis:

FETCH *nomCursor* INTO *var1* [*, var2*] ...;

Esta sentencia asigna los valores devueltos de la fila que se está leyendo sobre las variables indicadas tras INTO. Debe haber una variable por cada valor que devuelve el cursor (por cada valor seleccionado en la SELECT). Un cursor se comporta como un puntero que inicialmente apunta a los datos de la primera fila y, cuando se lee, el puntero se incrementa para apuntar a a siguiente fila y así sucesivamente hasta que el puntero llega al final tomando el valor nulo.

Todo cursor abierto debe ser cerrado. No es necesario haber consultado todas las filas controladas por el cursor para cerrar el cursor, se puede cerrar en cualquier momento. La sintaxis para cerrar un cursor es:

CLOSE *nomCursor;*

A continuación se muestra un ejemplo de una función que obtiene usando cursores el número de automóviles existentes (en la tabla automóviles) de la marca que se pase a esa función. Si la marca no existe se controla el error devuelto por una SELECT nula ('02000') asignando un valor 0 a la variable existe. Esto hace que cuando se llegue a no poder leer una fila con la condición dada también se pueda salir del bucle.

```
CREATE FUNCTION sumakil(m char(15))
returns int
BEGIN
DECLARE existe INT DEFAULT 1;
DECLARE tot,k INT;
DECLARE cur1 CURSOR FOR SELECT kilometros FROM automoviles where
marca=m;
DECLARE CONTINUE HANDLER FOR SQLSTATE '02000' SET existe = 0;
SET tot=0;
OPEN cur1;
FETCH cur1 INTO k;
WHILE existe=1 DO
SET tot=tot+k;
FETCH cur1 INTO k;
END WHILE;
CLOSE cur1;
return tot;
END
```

Triggers

Un trigger o disparador es una rutina asociada con una tabla que se activa o ejecuta automáticamente cuando se produce algún evento sobre la tabla (insertar una fila, modificar una columna, etc.). Por ejemplo, el siguiente trigger serviría para ir acumulando en una variable ya declarada, la suma de las edades de cada cliente insertado en la tabla CLIENTES:

```
CREATE TRIGGER suma_edad BEFORE INSERT ON clientes
FOR EACH ROW SET @sum = @sum + NEW.edad;
```

Una vez creado el trigger, cada vez que se inserte una fila en la tabla clientes, se ejecutará automáticamente el trigger suma_edad haciendo que se incremente con la nueva edad nsertada el valor de la variable @sum.

Sintaxis de cración de un trigger

***CREATE TRIGGER* nombreTrigger disparo evento**
***ON* nombreTabla *FOR EACH ROW* sentencia**

El nombre de tabla especifica la tabla que motiva el disparo del trigger cuando se produce algún evento sobre la tabla. No se puede asociar un trigger con una vista o con una tabla de tipo TEMPORARY. Disparo especifica el momento en el que se disparará el trigger, puede ser BEFORE o AFTER indicando antes o después, respectivamente, de ejecutarse la sentencia que produzca el disparo del trigger. El evento indica la acción sobre la tabla que provocará el disparo del trigger y puede ser INSERT, UPDATE, o DELETE. Por ejemplo, un trigger BEFORE para sentencias INSERT puede usarse para comprobar si los valores que van a ser insertados son válidos ante alguna regla.

Nunca puede haber dos triggers o más para una misma tabla que respondan al mismo evento y en el mismo momento de disparo. Sentencia indica la sentencia o sentencias que se ejecutan cuando se se lanza a ejecución el trigger, es decir, la acción realizada por el trigger. Si queremos que se ejecuten varias sentencias, tenemos que agruparlas dentro de un bloque BEGIN ... END. Dado que los triggers son rutinas, podemos usar para ellos las mismas entencias que en los procedimientos y en las funciones.

Para hacer referencia en un trigger a las columnas de las tabla que dispara el trigger, no podemos usar normalmente el nombre de esa columna. Para hacer referencia a esas columnas necesitaremos los especificadores OLD y NEW que indican el valor de la columna antes de la modificación que se trata de hacer y después de la modificación respectivamente. Por ejemplo el valor de la columna nombre que se ha insertado con un INSERT que ha disparado un trigger, se representa como NEW.nombre. Para crear triggers se necesita el SUPER privilegio de usuario.

Eliminación de un trigger

***DROP TRIGGER* tabla.nombreTrigger**
Ejemplo:
CREATE TRIGGER chequeMod BEFORE UPDATE ON saldo FOR EACH ROW
BEGIN
IF NEW.saldo < 0 THEN
SET NEW.saldo = 0;
ELSEIF NEW.saldo > 100 THEN
SET NEW.saldo = 100;
END IF;
END//