1. MYSQL

1.1. Introducción

1.1.1. Objetivo

Estos apuntes evitan mostrar la sintaxis completa, con todas sus variantes, de las sentencias SQL. En caso de querer acceder a ellas hacer uso de la documentación MySql oficial.

1.1.2. Notación

En los modelos de sintaxis y ejemplos indicados en estos apuntes:

- El promt del sistema operativo se indica así:

```
shell>
```

- El promt de MySql se indica así:

```
mysql>
```

- Las palabras escritas en mayúsculas son palabras reservadas de MySql.
- Las palabras en minúsculas son nombres genéricos que deberán sustituirse por los valores que interesen en cada caso.
- Los corchetes "[…]" indican elementos que pueden o no aparecer
- La barra vertical "|" indica elementos alternativos. Junto a las llaves "{A|B}" uno de ellos es obligatorio A o B, junto a corchetes "[A|B]" es opcional que aparezca uno u otro pero no ambos es decir puede aparecer A, B o ninguno de ellos.
- La sintaxis completa de cada elemento puede consultarse en el manual de referencia de MySQL. Aquí sólo encontraremos parte de las sintaxis posibles, en todo caso, sólo aquellas partes que nos interesan y que están dentro de nuestro nivel.

1.2. Instalación de MySql

TODO: pendiente

1.3. Cliente de línea de comandos MySql

1.3.1. Ayuda

```
shell> mysql --help
```

1.3.2. Iniciar sesión

```
shell> mysql -h host -u user -p
Enter password: ********
```

Ejemplo (con el que debemos empezar todos):

```
shell> mysql -h localhost -u root –p
```

Si queremos que no suenen pitidos de error debemos añadir otro parámetro al final "-b".

1.3.3. Cerrar sesión

```
mysql> QUIT
mysql> EXIT
```

1.3.4. Informaciones básicas

Conocer las bases de datos existentes:

```
SHOW DATABASES
```

Notar que debemos acabar las sentencias SQL con ";", si no lo hacemos al acabar una línea podemos incluir varias líneas en una sentencia.

En MySql existen funciones (*funcion()*) que llevan un paréntesis tras ellas. Dentro del paréntesis pueden llevar o no argumentos.

```
mysql> SELECT version();
```

También existen variables, que van sin paréntesis.

```
mysql> SELECT CURRENT_DATE;
```

Las palabras reservadas no diferencian entre mayúsculas y minúsculas.

```
mysql> SELECT current_date;
```

Podemos saber quién es el usuario actual:

```
mysql> SELECT user();
Podemos incluir comentarios:
mysql> SELECT user()/*un comentario */;
```

1.4. DDL: Lenguaje de definición de datos

DDL (Data Definition Language) es la parte del lenguaje usada para crear las bases de datos, las tablas y demás estructuras necesarias para guardar los datos.

1.4.1. Creación y borrado de bases de datos

Creación:

```
CREATE DATABASE [IF NOT EXISTS] db_name
```

Utilización o conexión a una base de datos:

```
USE database
```

Comprobación de qué base de datos estamos utilizando:

```
mysql> SELECT DATABASE();
```

Borrado, si incluimos "IF EXISTS" evitamos un posible error si no encuentra la base de datos, es muy útil cuando realicemos scripts:

```
DROP DATABASE [IF EXISTS] db_name
```

Εφεμπλο 1. – Crear la base de datos "dbejemplo" y comprobar su existencia.

```
mysql> CREATE DATABASE IF NOT EXISTS dbejemplo; /*Crea la base*/
mysql> SHOW DATABASES;/*Comprobación*/
```

Εφεμπλο 2. – Utilizar la base de datos creada

```
mysql> USE dbejemplo;/*Apertura de base*/
```

Εφεμπλο 3. – Borrar la base de datos "dbejemplo", si existe, y comprueba su borrado.

```
mysql> DROP DATABASE IF EXISTS dbejemplo;
mysql> SHOW DATABASES;
```

1.4.2. Tipos de datos en MySql

Ver Apéndice A, apartado 1, type

Vea también Apendice B, comparativa con los tipos de datos Access.

Para más información ver el manual de referencia MySQL (Capítulo 11. Tipos de columna)

1.4.3. Creación de una tabla

Ver manual de referencia MySQL, apartados 3.3.2 y 13.1.5

Definición de columnas

Creación de una tabla donde sólo especificamos las columnas:

```
CREATE TABLE nombretabla (columna1 tipo1 restriccionescolumna1, columna2 tipo2
restriccionescolumna2, …, restriccionestabla1, restriccionestabla2, …);
```

Una vez creada podemos consultar las tablas existentes y su estructura con las sentencias:

```
SHOW TABLES;
DESCRIBE nombretabla;
SHOW CREATE TABLE nombretabla;
```

Εφεμπλο 4. – Vuelve a crear la base de datos dbejemplo. Crea una tabla llamada "persona" con dos columnas: "nombre" que puede contener cadenas de hasta 40 caracteres y "fecha" de tipo fecha. Una vez creada comprueba el trabajo realizado.

```
mysql> USE dbejemplo
mysql> CREATE TABLE persona (nombre VARCHAR(40), fecha DATE);
mysql> SHOW TABLES;
mysql> DESCRIBE persona;/*mustra las características de la tabla*/
mysql> SHOW CREATE TABLE persona;/*muestra el CREATE TABLE que se uso*/
```

Modificadores y restricciones de columna

Al final de la definición de cada columna podemos añadir varias partículas que modifican la definición de cada una de ellas:

PRIMAY KEY	Para definir la clave primaria de un solo campo
NULL	Para permitir valores nulos (opción por defecto)
NOT NULL	Para no permitir valores nulos
AUTO_INCREMENT	Hace autoincremental un campo numérico
DEFAULT x	Define un valor por defecto (x puede ser NULL)
COMMENT '…'	Para añadir comentarios a las columnas

El elemento “COMMENT” es un comentario que se utiliza en herramientas que generan documentación automática de la base de datos.

La sintaxis completa de la definición de columna es:

```
col_name type [NOT NULL | NULL] [DEFAULT default_value]
[AUTO_INCREMENT] [UNIQUE [KEY] | [PRIMARY] KEY]
[COMMENT 'string']
```

Εφεμπλο 5. – Prueba las siguientes definiciones de la tabla pais en la que se prueban los anteriores modificadores.

```
mysql> CREATE TABLE pais1 (nombre VARCHAR(20) NOT NULL, poblacion INT NULL);
```

Permitir o no valores nulos.

```
mysql> CREATE TABLE pais2 (nombre VARCHAR(20) NOT NULL,
    -> poblacion INT NULL DEFAULT 1000000);
```

Indicar valores por defecto.

```
mysql> CREATE TABLE pais3 (nombre VARCHAR(20) NOT NULL PRIMARY KEY,
    -> poblacion INT NULL DEFAULT 1000000);
```

Indicar la clave principal

```
mysql> CREATE TABLE pais4 (clave INT AUTO_INCREMENT PRIMARY KEY,
    -> nombre VARCHAR(20) NOT NULL,
    -> poblacion INT NULL DEFAULT 5000);
```

Hacer un campo autoincremental

```
mysql> CREATE TABLE pais5
    -> (clave INT AUTO_INCREMENT PRIMARY KEY COMMENT 'Clave principal',
    -> nombre VARCHAR(50) NOT NULL,
    -> poblacion INT NULL DEFAULT 5000);
```

Restricciones de tabla

Hablamos de restricciones de tabla porque las definimos después de definir las columnas. Tienen la particularidad de poder afectar a más de una columna. En este tipo de restricciones podemos definir:

- Claves primarias
- Índices (únicos o no)
- Claves ajenas.

Claves primarias

Podemos indicar en este lugar la creación de la clave principal. Si la clave principal es compuesta (más de un campo) su definición debe realizarse de este modo. Si es simple (un solo campo) puede hacerse aquí o como restricción de columna. Sintaxis:

```
[CONSTRAINT [symbol]] PRIMARY KEY (index_col_name,...)
```

Εφεμπλο 6. – Creación de una tabla con clave principal compuesta.

```
mysql> CREATE TABLE mitabla(
    -> id INT,
    -> nombre VARCHAR(19),
    -> apellido VARCHAR(20),
    -> CONSTRAINT  pk PRIMARY KEY (nombre, apellido));
```

Índices y claves alternativas

Podemos incluir índices de dos tipos, que permitan valores duplicados para lo que usaremos “INDEX” (o “KEY” indistintamente) y que no los permitan para lo que usaremos “UNIQUE” o “UNIQUE INDEX”, lo que equivale a una “clave candidata”. En ambos casos se permiten los valores nulos. Las dos sintaxis de uno y otro tipo son:

```
KEY [index_name] (index_col_name,...) /*índice no único con KEY idem siguiente*/
INDEX [index_name] (index_col_name,...) /*índice no único usando INDEX*/
[CONSTRAINT [symbol]] UNIQUE [INDEX] [index_name] (index_col_name,...)
Donde index_colname:
index_col_name:
col_name [(length)] [ASC | DESC]
```

Decir de lo anterior que los índices pueden tener un nombre (index_name). Los índices únicos son restricciones o CONSTRAINT además de índices y pueden tener otro identificador o símbolo (symbol).

Como podemos ver podemos crear índices sobre parte de un campo (ver ejemplo 8), además el índice puede definirse con orden ascendente (predeterminado) o descendente.

Εφεμπλο 7. – Índice con dos campos. Vemos dos modos de crearlo, uno en el que no damos nombre al índice y otro en el que le llamamos “mitabla2index1”

```
mysql> CREATE TABLE mitabla2 (
    -> id INT,
    -> nombre VARCHAR(19),
    -> apellido VARCHAR(20),
    -> INDEX mitabla2index1(nombre, apellido));

mysql> CREATE TABLE mitabla3 (
    -> id INT,
    -> nombre VARCHAR(19),
    -> apellido VARCHAR(20),
    -> INDEX (nombre, apellido));
```

Como vemos en los ejemplos estos elementos los añadimos después de haber definido todas las columnas.

Εφεμπλο 8. – Podemos crear índice sobre parte de un campo de texto, en el ejemplo siguiente lo hacemos sobre 4 caracteres del campo nombre.

```
mysql> CREATE TABLE mitabla4 (
    -> id INT,
    -> nombre VARCHAR(19),
    -> INDEX (nombre(4)));
```

Εφεμπλο 9. – El siguiente es un ejemplo de índice único. Observar como hemos dado un nombre a la restricción de tipo único (mitabla6iu1) y otro al índice (index1).:

```
mysql> CREATE TABLE mitabla6 (
    -> id INT,
    -> nombre VARCHAR(19) NOT NULL,
    -> CONSTRAINT  mitabla6iu1 UNIQUE INDEX  index1(nombre));
```

Los índices únicos son la manera de plasmar en el esquema relacional las claves alternativas del diagrama E/R (claves candidatas no elegidas como clave principal).

Claves ajenas

MySql puede utilizar varios **motores o "engines"** para almacenar las bases de datos. Esto hace referencia a la estructura de los ficheros donde se guardan las tablas. Por defecto en Windows el montor es "InnoDB". Este es el único motor que tolera el uso de claves ajenas y es por lo tanto el que usaremos siempre. Podemos verificar cual es el motor por defecto en nuestro sistema. Debemos abrir el fichero my.ini en el directorio de instalación y buscar la entrada "default-storage-engine". Es normal que coincida con esto:

```
default-storage-engine=INNODB
```

Si no lo especificamos al crear una tabla se crean según lo indicado en my.ini aunque en alguno de los siguientes ejemplos veremos que se puede añadir a la definición de la tabla el tipo de motor utilizado.

Según el estándar SQL las claves ajenas se pueden definir añadiendo la restricción al final de la definición de campo, como la clave principal, lo que llamamos restricción de columna. En este caso tendría la siguiente sintaxis:

```
nombreCampo TipoDato REFERENCES tabla(campo)
```

MySQL no produce ningún error usando este método pero realmente no crea la clave ajena. Lo incluimos aquí porque este método si es adecuado en otros gestores de bases de datos pero no debemos usarlo nunca con MySQL.

También se pueden definir como restricción de tabla con la siguiente sintaxis resumida:

```
CONSTRAINT nombre FOREIGN KEY(listacampos) REFERENCES TablaReferida(listacampos)
```

La sintaxis completa es:

```
[CONSTRAINT [symbol]] FOREIGN KEY [index_name] (index_col_name,...) [reference_definition]
reference_definition:
```

```
REFERENCES tbl_name [(index_col_name,...)]
[ON DELETE reference_option]
[ON UPDATE reference_option]
reference_option:
RESTRICT | CASCADE | SET NULL | NO ACTION
```

En el primer bloque expresamos el nombre de la restricción, el tipo (FOREIGN KEY) y los campos a los que afecta. En el segundo bloque indicamos el nombre de la tabla referenciada, y opcionalmente los campos en la misma, por defecto será la clave principal. El último bloque indica que hacer en esta tabla en caso de borrado o modificación de registros relacionados en la tabla referenciada o principal.

RESTRICT: no permitir la eliminación/borrado.

CASCADE: borrar o modificar en cascada

SET NULL: deja el campo a nulo, eliminando así un falso enlace

NO ACTION: equivalente a RESTRICT.

Como podemos ver al crear una clave ajena podemos darle nombre a la restricción creada. A su vez creamos implícitamente un índice al que podemos darle nombre también.

Εφεμπλο 10. – Creación de dos tablas con una relación de clave ajena añadida en la definición de columnas.

NOTA IMPORTANTE: este modo existe en el estándar SQL pero en MySql no funciona, es decir, no se produce ningún error pero no se asegura la integridad referencial. Por lo que vemos como sería pero usaremos el siguiente método siempre.

Εφεμπλο 11. – Creación de dos tablas con una relación de clave ajena añadida como restricción de columna: ¡¡NO VALIDO EN MySQL!!.

```
mysql> CREATE TABLE personas (
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> nombre VARCHAR(40),
    -> fecha DATE);
mysql> CREATE TABLE telefonos (
    -> numero VARCHAR(12),
    -> id INT NOT NULL REFERENCES personas (id)
    -> ON DELETE CASCADE ON UPDATE CASCADE);
```

Εφεμπλο 12. – Creación de dos tablas con una relación de clave ajena añadida como restricción de tabla.

```
mysql> CREATE TABLE personas2 (
    -> id INT AUTO_INCREMENT PRIMARY KEY,
    -> nombre VARCHAR(40),
    -> fecha DATE);
mysql> CREATE TABLE telefonos2 (
   -> numero VARCHAR(12),
   -> id INT NOT NULL,
   -> CONSTRAINT `telefonos2fk1` FOREIGN KEY (id) REFERENCES personas2 (id)
   -> ON DELETE CASCADE ON UPDATE CASCADE);
```

1.4.4. Modificación de una tabla

Para mayor detalle ver manual de referencia MySQL, apartado 13.1.2

Para añadir una columna:

```
ALTER TABLE tbl_name ADD [COLUMN] column_definition [FIRST | AFTER col_name ], ...
```

Como vemos podemos añadir una (o más columnas) con la sintáxis anterior. La definición de la misma sigue la misma pauta que en la creación de tablas. Podemos indicar también su posición dentro de la tabla, por defecto se coloca al final.

Para borrar una columna:

```
ALTER TABLE tbl_name DROP [COLUMN] col_name
```

Εφεμπλο 13. – Vamos a añadir el campo apellido a la tabla persona del ejemplo 4 después de la columna nombre. A continuación lo borramos.

```
ALTER TABLE persnoa ADD COLUMN apellido VARCHAR(50) AFTER nombre;
ALTER TABLE persona DROP COLUMN apellido;
```

Igualmente podemos añadir un índice o una restricción y de manera similar borrarlo, para esta operación debemos hacer uso del nombre del índice o de la restricción. Las sintáxis son las siguientes:

```
ALTER TABLE tbl_name ADD INDEX [index_name] (index_col_name,...);
ALTER TABLE tbl_name ADD [CONSTRAINT [symbol]]...;/*completar según tipo constraint:
UNIQUE INDEX, PRIMARY KEY O FOREIGN KEY*/
ALTER TABLE tbl_name DROP PRIMARY KEY;
ALTER TABLE tbl_name DROP INDEX index_name;
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

Algunas otras posibilidades interesantes son:

```
ALTER TABLE tbl_name ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}/*
para añadir un valor por defecto o borrar el existente*/
ALTER TABLE tbl_name CHANGE [COLUMN] old_col_name column_definition [FIRST|AFTER
col_name]/*modificar la definición de una columna, nombre incluido y posición
incluida*/
ALTER TABLE tbl_name MODIFY [COLUMN] column_definition [FIRST | AFTER col_name]
/*modificar la definición y posición de una columna pero no el nombre*/
ALTER TABLE tbl_name RENAME [TO] new_tbl_name
```

En MySQL podemos anidar varias modificaciones en una sola sentencia ALTER. La palabre literal hace se refiere a un valor concreto, numérico, de texto (entre comillas simples o dobles), de fecha (también entre comillas) o NULL es decir valor nulo

Εφεμπλο 14. – Añadir a la tabla persona del ejemplo 4 una columna apellido (VARCHAR(50)) y otra "nacidoen" (VARCHAR(40)). Una vez hecho esto borrarlas en una sóla sentencia:

```
ALTER TABLE persona ADD COLUMN apellido VARCHAR(50), ADD COLUMN nacidoen
VARCHAR(40);
ALTER TABLE persona DROP COLUMNA apellido, DROP COLUMN nacidoen;
```

1.4.5. Creación de vistas

Pendiente de explicar en el apartado de DML

1.4.6. Creación y uso de scripts

Podemos almacenar varias sentencias SQL en un fichero de script. Consiste en escribir las sentencias en un fichero de texto tal como las escribiríamos en el cliente, acabando lógicamente cada sentencia con ";".

El fichero puede guardarse con extensión "txt" pero es habitual hacerlo con extensión "sql".

Para usar uno de estros script debemos hacerlo del siguiente modo:

```
mysql > SOURCE NombreFichero.sql;
```

Por ejemplo el fichero "prueba.sql" ubicado en "c:\scripts" se ejecutaría:

```
mysql > SOURCE c:/scripts/prueba.sql
```

EJERCICIOS (3 HORAS)

Partiendo del diccionario de datos del apéndice B realiza las siguientes acciones mediante SQL:

.1 Utiliza “tee” para direccionar el texto de la consola a un fichero de texto con tu nombre. Al acabar la sesión entrega dicho fichero al profesor.

.2 Crea la base de datos supermercado.

.3 Crea las tabla Artículo, considera si puedes crearla con las claves ajenas o no. Haz lo mismo con las tablas Pedido y con ArticuloPedido.

.4 Borra el campo Cantidad de la tabla ArticuloPedido.

.5 Añade el campo borrado en el ejercicio anterior.

.6 Crea la tabla Proveedor

.7 Añade un índice a la tabla proveedor de tipo único sobre el campo “Nombre”.

.8 Crea todas las tablas restantes considerando cada una de las restricciones, observaciones y claves ajenas y principales.

.9 Utiliza “Alter Table” para crear las claves ajenas que falten.

.10 Utiliza “Describe Table” y “Show Create Table” para verificar que toda la base de datos está completa.

NOTA: los campos AUTO_INCREMENT añadir el modificador UNSIGNED y ZEROFILL al tipo de datos: INT(5) UNSIGNED ZEROFILL

SCRIPTS (1 HORA)

.11 Crea un script llamado “supermercado.sql” que cree la base de datos supermercado. Inicialmente debe borrar la base de datos si existe y antes de crear una tabla debemos borrarla si existe previamente.

1.5. Copias de seguridad, exportación e importación de datos.

Recordamos que DDL era el lenguaje de definición de datos con el que creábamos las estructuras de datos. DML quiere decir Lenguaje de manipulación de datos y con él vamos a ser capaces de **introducir** datos en una base de datos, **modificarlos**, **borrarlos** y **consultarlos**. Dado que el trabajo de consulta es el más versátil y necesario vamos a empezar por ahí.

Para trabajar y hacer prácticas es necesario disponer de una base de datos que nos permita manejar tablas que no estén vacías. Para lograrlo podemos utilizar la base de datos Neptuno que se distribuye con MS Access.

Para enlazar dos gestores de bases de datos existe un estándar desarrollado por Microsoft llamado ODBC (Open Database Conectivity). MySQL tiene soporte para ODBC pero debemos instalar un paquete adicional. Podemos encontrar el instalador de este paquete en la carpeta descargas del CD del libro.

Para exportar toda la base de datos Neptuno vamos a tomar imágenes de la tarea realizada y después haremos una guía de exportación. Podemos hacer el trabajo en equipo y así nos resultará más fácil. Pasos a seguir:

- Instalar el soporte ODBC de MySQL.
- Preparación de la base de datos MySQL. Vamos a crear una base de datos llamada Neptuno en MySQL.
- Preparación de ODBC. Vamos al panel de control, herramientas administrativas y allí en orígenes de datos ODBC vamos a añadir un origen de datos DSN de sistema (eso permite que sea usado por todos los usuarios de

nuestro equipo). Debemos buscar el origen de datos (MySQL) y configurarlo para que se conecte a la base de datos que acabamos de crear.

- Reparto y preparación de tablas de Neptuno. Vamos a repartirnos una o dos tablas por alumno. Para las tablas que se nos asignen debemos eliminar los espacios, vocales acentuadas y las "ñ" de los nombres las tablas y dentro de ellas de los nombres de los campos.
- Exportación mediante ODBC a MySQL.
- Creación de copias de seguridad de las tablas en MySQL

```
shell>mysqldump --host=localhost --user=root --password=obdoc --opt neptuno categorias > c:/categorias.sql
```

- Intercambio de ficheros de copia de seguridad.
- Carga de las copias de seguridad creadas por nuestros compañeros.

```
mysql>SOURCE c:/categories.sql
```

1.6. DML

Vamos a ir realizando ejemplos para ver las posibiliddes de cada una de las sentencias SQL en lugar de intentar desentrañar todas las posibilidades de la sintaxis de las mismas.

1.6.1. SELECT: Consultas sencillas

La cláusula SELECT se utiliza para seleccionar registros de una o varias tablas y su sintáxis más sencilla es:

```
SELECT listacampos FROM listatablas
```

Selección de todos los registros y todos los campos.

Para seleccionar todos los campos de una tabla se usa "*", se puede usar sólo si hay una sóla tabla, en caso contrario debe hacerse del modo "NombreTabla.*".

Εφεμπλο 15. – Selecciona todos los registros de la tabla categorias

```
mysql> SELECT * FROM categorias;
mysql> SELECT categorias.* FROM categorias;
```

Εφεμπλο 16. – Selecciona todos los registros de la tabla "productos".

Selección de algunos campos.

Εφεμπλο 17. – Selecciona los campos idCategoria y NombreCategoria de la tabla categorias.

```
mysql> SELECT idCategoria, NombreCategoria  FROM categorias;
```

Εφεμπλο 18. – Selecciona el nombre y el país de la tabla clientes

Renombrar campos.

Los campos pueden renombrarse con la cláusula AS, "NombreOriginal AS NombreNuevo".

Εφεμπλο 19. – Selecciona el idCategoria y el NombreCategoria de la tabla categorias de manera que los campos devueltos se llamen clave y nombre

```
mysql> SELECT idCategoria AS clave, NombreCategoria AS nombre FROM categorias;
```

Εφεμπλο 20. – Selecciona el idPedido y el idCliente de la tabla pedidos y renombra dichos campos como pedido y cliente.

Seleccionar registros específicos

Para seleccionar registros añadimos la cláusula "WHERE" después de "FOR". Para relacionar un campo con otro o con un literal usaremos los operadores: =, >, <, <=, >=, <>. Casos más especiales son BETWEEN … AND … para filtrar entre dos valores, y LIKE que veremos un poco más adelante cuando usamos patrones.

```
mysql> SELECT listacampos FROM listatablas WHERE condicion
```

Los literales de texto van entre comillas simples o dobles.

Εφεμπλο 21. – Seleccionar el registro de clientes donde idCliente es "HANAR".

```
mysql> SELECT * FROM clientes WHERE idCliente='HANAR';
mysql> SELECT * FROM clientes WHERE idCliente="HANAR";
```

Εφεμπλο 22. – Seleccionar los nombres de las empresas cuya persona de contacto tenga de cargo "propietario".

```
mysql> SELECT NombreCompanhia FROM clientes WHERE CargoContacto='propietario';
mysql> SELECT NombreCompanhia FROM clientes WHERE CargoContacto="propietario";
```

Εφεμπλο 23. – Selecciona todos los campos del empleado cuyo idEmpleado es 4.

```
mysql> SELECT * FROM empleados WHERE idEmpleado = 4;
```

Εφεμπλο 24. – Selecciona el nombre y apellidos de los empleados cuyo idEmpleado esté entre 5 y 8.

```
mysql> SELECT nombre, apellidos FROM empleados WHERE idEmpleado BETWEEN 5 AND 8;
```

Seleccionar registros específicos usando patrones o comodines.

Para sustituir varios caracteres usaremos "%" y para un solo carácter "_". Además si usamos patrones debemos sustituir el operador "=" por "LIKE".

Εφεμπλο 25. – Seleccionar los registros de empleados cuyo nombre empiece por "A".

```
mysql> SELECT * FROM empleados WHERE nombre LIKE 'A%';
```

Εφεμπλο 26. – Seleccionar los registros de empleados cuyo apellido tenga una 'e' como segundo carácter.

```
mysql> SELECT * FROM empleados WHERE apellido LIKE '_e%';
```

Seleccionar registros específicos usando fechas.

Las fechas se formatean como "aaaa-mm-dd" y los literales deben ir entre comillas como los textos.

Εφεμπλο 27. – Seleccionar los idPedido de la tabla pedidos con FechaPedido anteriores al 1 de diciembre de 1996.

```
SELECT IdPedido FROM pedidos WHERE FechaPedido < '31-12-1'
```

Εφεμπλο 28. – Selecciona el nombre y apellidos del empleado nacido el 30 de agosto de 1963.

```
mysql> SELECT nombre, apellidos FROM empleados WHERE FechaNacimiento = '1963-08-
30';
```

Εφεμπλο 29. – Selecciona el nombre y apellidos de los empleados nacidos en julio.

```
mysql> SELECT nombre, apellidos FROM empleados WHERE FechaNacimiento = '%-07-%';
```

Εφεμπλο 30. – Selecciona el nombre y apellidos de los empleados nacidos entre el 1 de julio de 1992 y 31 de diciembre del mismo año.

```
mysql> SELECT nombre, apellidos, FechaContratacion
    -> FROM empleados WHERE
    -> FechaContratacion BETWEEN '1992-07-01' AND '1992-12-31';
```

Ordenar registros

Para ordenar registros debemos añadir una cláusula más al final del SELECT cuya sintáxis se muestra a continuación. Si no indicamos si la ordenación es ascendente (ASC) o descendente (DESC) el valor por defecto es el primero.

```
SELECT ListaCampos
FROM ListaTablas
WHERE Condicion
ORDER BY campo1 [ASC|DESC], campo2 [ASC|DESC], …
```

Εφεμπλο 31. – Selecciona todos los campos de empleados y muestralos ordenados por nombre y apellidos. Dos soluciones válidas:

```
mysql> SELECT * FROM Empleados ORDER BY nombre, apellidos;
mysql> SELECT * FROM Empleados ORDER BY nombre ASC, apellidos ASC;
```

Εφεμπλο 32. – Seleccionar el nombre y apellido de los empleados mostrando en primer lugar los más antiguos en la empresa.

```
mysql> SELECT nombre, apellidos FROM Empleados ORDER BY fechaContratacion DESC;
```

Trabajar con valores NULL

Un campo sin rellenar contiene un valor cuyo tratamiento es siempre especial, el valor nulo. En SQL el valor nulo se expresa como NULL. En la cláusula WHERE se utiliza con las expresiones “IS NULL” o “NOT IS NULL” en lugar de con el “=” como muestran los ejemplos siguientes.

Εφεμπλο 33. – Muestra el nombre y apellidos del empleado o empleados que no tienen jefe.

```
mysql> SELECT nombre, apellidos FROM Empleados WHERE jefe IS NULL;
```

Εφεμπλο 34. – Muestra el nombre y apellidos del empleado o empleados que si tienen jefe.

```
mysql> SELECT nombre, apellidos FROM Empleados WHERE jefe NOT IS NULL;
```

Seleccionar registros combinando más de un requisito

Cuando deseamos combinar más de un requisito debemos combinarlos mediante la operación lógica adecuada. Las operaciones lógicas básicas son:

- NOT: “Negado”, cambia el valor del operador al que se aplica.
- AND: “Y” cierto si ambos operadores son ciertos.
- OR: “O”, cierto si al menos un operador es cierto.
- XOR: “O exclusivo” cierto si un eperador es verdadero y el otro falso.

Hemos de ser muy cuidadosos porque los anteriores operadores se aplican en el orden mostrado arriba, si queremos alterar dicho orden debemos utilizar paréntesis.

Εφεμπλο 35. – Muestra el nombre y apellidos de los trabajadores nacidos en 1960 o después y que son de Londres.

```
mysql> SELECT nombre, apellidos
```

```
    -> FROM Empleados
    -> WHERE fechaNacimiento > '1960-%' AND ciudad = 'Londres';
```

Εφεμπλο 36. – Muestra el nombre y apellidos de los trabajadores nacidos en 1960 o después y que son de Londres o de Seattle.

```
mysql> SELECT nombre, apellidos
    -> FROM Empleados
    -> WHERE fechaNacimiento > '1960-%' AND (ciudad='Londres' OR ciudad='Seattle');
```

Εφεμπλο 37. – Muestra el nombre y apellidos de los trabajadores nacidos en 1960 o después y que son de Londres o son de Seattle sin importar su fecha de nacimiento.

```
mysql> SELECT nombre, apellidos
    -> FROM Empleados
    -> WHERE fechaNacimiento>('1960-%' AND ciudad='Londres') OR (ciudad='Seattle');
mysql> SELECT nombre, apellidos
    -> FROM Empleados
    -> WHERE fechaNacimiento > '1960-%' AND ciudad='Londres' OR ciudad='Seattle';
```

Εφεμπλο 38. – Muestra el nombre y apellidos de los trabajadores nacidos en 1960 que son de Londres o son de "EE.UU." y no nacieron en 1958.

```
mysql> SELECT nombre, apellidos
    -> FROM Empleados
    -> WHERE fechaNacimiento LIKE '1960-%' AND ciudad='Londres'
    -> OR pais='EE.UU.' AND fechaNacimiento NOT LIKE 1960-%';
mysql> SELECT * FROM Empleados ORDER BY nombre, apellidos;
```

Registros duplicados.

En ocasiones una consulta devuelve colecciones de registros entre los que hay duplicados. SQL dispone de herramientas para mantener dichos registros duplicados o para eliminarlos. Dichas herramientas son las siguientes:

SELECT DISTINCT listacampos FROM … # Elimina los duplicados

SELECT ALL listacampos FROM … # Mantinen todos, sin eliminar duplicados

Cada gestor de bases de datos opta por una política por defecto. MySQL presupone la partícula ALL por lo que no elimina duplicados.

Εφεμπλο 39. – Seleccionar todos los cargos de la tabla empleados ordenados alfabéticamente.

```
mysql> SELECT cargo FROM Empleados ORDER BY cargo;
mysql> SELECT DISTINCT cargo FROM Empleados ORDER BY cargo;
```

Εφεμπλο 40. – Seleccionar todos los cargos de la tabla empleados ordenados alfabéticamente, sin duplicados.

```
mysql> SELECT DISTINCT cargo FROM Empleados ORDER BY cargo;
```

Limitar la salida de una consulta a un número de registros determinado.

Si deseamos seleccionar un número máximo de registros en una consulta debemos añadir la cláusula LIMIT después de la cláusula ORDER. Este tope de registros sólo puede ser expresado en números absolutos (… LIMIT n) pero en porcentaje del total de registros (LIMIT n%) cosa que si aceptan otros gestores. Veámoslo con un par de ejemplos.

Εφεμπλο 41. – Seleccionar el nombre y apellidos de los tres primeros registros de la tabla empleados.

```
mysql> SELECT nombre, apellidos FROM Empleados LIMIT 3;
```

Εφεμπλο 42. – Seleccionar el nombre y apellidos de la mitad de los registros de la tabla empleados, o lo que es igual, del 50%.

```
mysql> SELECT nombre, apellidos FROM Empleados LIMIT 50%;#No aceptado por MySQL
```

Una posibilidad más que nos da la cláusula LIMIT es seleccionar no exactamente los primeros registros sino otro intervalo. Para seleccionar los registros entre el "n-1" y el "m-1" usaremos la siguiente variante, esto es así porque el primer registro tiene el índice 0.

Εφεμπλο 43. – Seleccionar el nombre y apellidos de los registros 3 al 5 de la tabla empleados.

```
mysql> SELECT nombre, apellidos FROM Empleados LIMIT 2,4;
```

Aprovechando el uso de LIMIT y ORDER podemos realizar búsquedas más elaboradas como buscar el primer registro una vez ordenados por determinado criterio. Veámoslo en ejemplos.

Εφεμπλο 44. – Seleccionar el número y cargo del pedido de mayor cargo.

```
mysql> SELECT idPedido, cargo FROM pedidos ORDER BY cargo DESC LIMIT 1; /*Observar
que la ordenación es descendente para que el primer registro sea el de mayor
cargo*/
```

Εφεμπλο 45. – Seleccionar el número y cargo de los diez pedidos de mayor cargo.

```
mysql> SELECT idPedido, cargo FROM pedidos ORDER BY cargo DESC LIMIT 10;
```

Εφεμπλο 46. – Seleccionar el número y cargo del segundo pedido de mayor cargo.

```
mysql> SELECT idPedido, cargo FROM pedidos ORDER BY cargo DESC LIMIT 1,1;
```

EJERCICIOS (2 HORAS)

Utilizando la base de datos Neptuno realiza las siguientes consultas:

.12Selecciona los campos .

1.6.2. SELECT: Consultas con agregados

Hasta ahora todo lo que hemos visto realizaba operaciones de selección sobre todo un registro o una parte de él. En esta sección vamos a ver como realizar consultas sobre agrupaciones de registros. Para este tipo de consultas incluiremos una serie de funciones específicas de cálculos agregados y un par de cláusulas nuevas en la sentencia SELECT. La nueva sintáxis es:

```
SELECT listaCampos
FROM listaTablas
[WHERE condición]
[GROUP BY listaCampos]
[HAVING condición]
[ORDER BY campo1 [ASC|DESC] (, …)]
[LIMIT n[, m]]
```

Las funciones de agregados más habituales son las siguientes:

```
AVG([DISTINCT] expresion)/*promedio*/
COUNT([DISTINCT] expresion) /*cuenta de registros*/
MAX([DISTINCT] expresion) /*máximo, el DISTINCT no cambia el resultado*/
MIN([DISTINCT] expresion) /*mínimo, el DISTINCT no cambia el resultado */
SUM([DISTINCT] expresion) /*suma*/
/*como vemos existe la posibilidad de añadir la partícula DISTINCT lo que puede
matizar la función, no es lo mismo contar cuantos valores toma el campo "x" que
cuantos valores distintso toma dicho campo*/
```

Εφεμπλο 47. – Cuenta cuantos registros hay en la tabla pedidos

```
mysql> SELECT COUNT(*) FROM PEDIDOS;
mysql> SELECT COUNT(*) AS cuenta FROM PEDIDOS;#renombrando el campo calculado
```

Εφεμπλο 48. – Suma el importe de todos los cargos de la tabla pedidos

```
mysql> SELECT SUM(cargo) FROM PEDIDOS;
```

```
mysql> SELECT SUM(cargo) AS SumaCargos FROM PEDIDOS; #renombrando el campo
calculado
```

Εφεμπλο 49. – Calcula el promedio, el máximo y el mínimo del cargo de la tabla pedidos.

```
mysql> SELECT AVG(cargo), MAX(cargo), MIN(cargo) FROM PEDIDOS;
mysql> SELECT AVG(cargo) AS promedio, MAX(cargo) AS Maximo, MIN(cargo) AS Minimo
    -> FROM PEDIDOS; #renombrando los campos calculados
```

Los campos que hay en la listaCampos del SELECT pueden usar funciones agregadas o no. Si no son funciones agregadas deben incluirse en la cláusula GROUP BY. Obsérvalo en el siguiente ejemplo donde pais no es un agregado y debe aparecer dentro de la cláusula GROUP BY.

Εφεμπλο 50. – Contar para cada país cuantos empleados hay.

```
mysql> SELECT pais, COUNT(IdEmpleado)
    -> FROM empleados
    -> GROUP BY pais;
```

Igual que usabamos la cláusula WHERE podemos utilizar la cláusuala HAVING para seleccionar registros usando funciones agregadas.

Εφεμπλο 51. – Seleccionar el IdCliente y calcular la suma de cargos para cada IdCliente de la tabla pedidos. Mostrar sólo aquellos cuya suma de pedidos sea inferior a 100.

```
mysql> SELECT Idcliente, SUM(CARGO) AS suma from pedidos
    -> GROUP BY IdCliente
    -> having suma<100;
```

Εφεμπλο 52. – Seleccionar el IdCliente y el pais de la tabla pedidos. Calcular también el cargo máximo de los pedidos de cada cliente y mostrar aquellos registros para los que dicho máximo sea superior a 100.

```
mysql> SELECT IdCliente, PaisDestinatario, MAX(cargo) AS maximo
    -> FROM pedidos
    -> GROUP BY IdCliente, PaisDestinatario
    -> HAVING maximo<150;
```

HAVING y WHERE funcionan de manera similar, y dentro de HAVING podemos filtrar por campos que no sean agregados, no obstante es importante usar el más adecuado en cada momento. HAVING debe ser usado con agregados mientras que WHERE debe hacerse con no agregados. Esto último es así porque es mucho más eficiente filtrar antes de realizar todo el esfuerzo del cálculo agregado que hacerlo después. Veamos un ejemplo, en ambos casos el resultado es el mismo pero el primero de ellos es el más adecuado por cuestiones de eficiencia.

Εφεμπλο 53. – Seleccionar el IdCliente y el pais de la tabla pedidos. Calcular también el cargo máximo de los pedidos de cada cliente y mostrar aquellos registros para los que dicho máximo sea superior a 100 y para los que PaisDestinatario sea Brasil.

```
mysql> SELECT IdCliente, PaisDestinatario, MAX(cargo) AS maximo
    -> FROM pedidos
    -> WHERE PaisDestinatario='Brasil';
    -> GROUP BY IdCliente, PaisDestinatario
    -> HAVING maximo<150; /*EFICIENTE*/
mysql> SELECT IdCliente, PaisDestinatario, MAX(cargo) AS maximo
    -> FROM pedidos
    -> GROUP BY IdCliente, PaisDestinatario
    -> HAVING maximo<150 AND PaisDestinatario='Brasil';/*NO EFICIENTE*/
```

1.6.3. SELECT: Usar funciones y campos calculados

Podemos emplear campos calculados a partir de otros campos o con la ayuda de funciones que no requieren ningún valor de entrada. El cliente de MySQL no obliga a renombrar estos campos calculados aunque es lo más adecuado y lo que deberíamos hacer si la consulta fuese empleada por otro programa. Ya hemos visto como se renombran los campos con la partícula AS.

Operadores

La manera más sencilla de calcular nuevos campos es con los operadores matemáticos: + suma, - resta, * multiplicación, / división, DIV división entera, MOD resto de división entera y ^ potencia.

Εφεμπλο 54. – La estimación de beneficios de cada pedido se estima en el 5% del cargo. Relizar dos consultas, la primera que calcule el beneficio para cada uno de los pedidos y la segunda que calcule la suma de beneficios de todos los pedidos.

```
mysql> SELECT cargo*0.05 AS Beneficio FROM Pedidos;
mysql> SELECT SUM(cargo*0.05) AS BeneficioTotal FROM Pedidos;
```

1.6.4. SELECT: Consultas con más de una tabla

INNER JOIN

RIGHT Y LEFT JOIN

UNION

1.6.5. SELECT: Consultas con subconsultas

1.6.6. UPDATE

1.6.7. INSERT

1.6.8. DELETE

1.7. DCL: Lenguaje de control de datos. Seguridad

1.7.1. Creación de usuarios

Los usuarios son creados dentro del servidor MySQL. Sus datos se guardan dentro de la base de datos MySQL. Para crear un usuario, la sintaxis es:

```
CREATE USER NombreUsuario [IDENTIFIED BY 'contraseña'];
```

Donde nombre de usuario puede tener la forma “nombre” o “nombre@hostcliente”. “hostcliente” puede hacer referencia a una dirección IP, a un nombre de equipo o a una dirección de red (p.e. 192.168.3.0/255.255.255.0). Veamos un par de ejmplos:

```
CREATE USER rafa IDENTIFIED BY 'rafa'
```

```
CREATE USER martin@pc2 IDENTIFIED BY 'martin'/*solo puede conectarse martin desde
pc2*/
```

Hasta la versión 5 la orden CREATE USER no funcionaba. Los usuarios debían darse de alta de manera implícita mediante la asignación de permisos como se muestra en los siguientes ejemplos, observa que al dar permisos a usuarios inexistentes estos se dan de alta automáticamente:

```
mysql> GRANT USAGE ON *.* TO alumno1 IDENTIFIED BY 'alumno1';
mysql> GRANT ALL PRIVILEGES ON *.* TO alumno1@'192.168.2.0/255.255.255.0'
    ->IDENTIFIED BY alumno1; /*Puede conectarse alumno1 desde cualquier equipo de la
red indicada*/
```

Los nombres de usuario se guardan en la tabla user de la base de datos mysql. Podemos consultarla como una tabla más:

```
mysql> SELECT user, host from mysql.user;
+---------+---------------------------+
| user    | host                      |
+---------+---------------------------+
| alumno1 | %                         |
| rafa    | %                         |
| alumno2 | 192.168.6.0/255.255.255.0 |
| root    | localhost                 |
| martin  | pc2                       |
+---------+---------------------------+
```

Para abrir una sesión cliente debe coincidir el nombre de usuario, la contraseña y el host desde que nos conectamos con alguna entrada de la tabla "mysql.user". En el nombre de host podemos usar caracteres comodines por lo que "%" indica cualquier equipo.

Εφεμπλο 55. – Explica las cinco entradas de la tabla usuarios que acabamos de ver.

Hay cinco usuarios con los nombres de la columna user. Dichos usuarios pueden conectarse. Alumno1 desde cualquier equipo, rafa también, alumno2 desde cualquier equipo de la red 192.168.6.0/255.255.255.0, root sólo desde el localhost y martin desde pc2.

1.7.2. Asignación de permisos

En **MySQL** existen cinco niveles distintos de privilegios que por orden de importancia son:

Globales: se aplican al conjunto de todas las bases de datos en un servidor.

De base de datos: se refieren a bases de datos individuales, y por extensión, a todos los objetos que contiene cada base de datos.

De tabla: se aplican a tablas individuales, y por lo tanto, a todas las columnas de esas tabla.

De columna: se aplican a una columna en una tabla concreta.

De rutina: se aplican a los procedimientos almacenados, tema que queda fuera del alcance de este libro.

Los permisos deben ser asignados con la orden GRANT:

```
GRANT TipoPermiso [(ListaColumnas)]
ON {tabla | * | *.* | BaseDatos.*}
TO Usuario [IDENTIFIED BY 'contraseña']
[WITH GRANT OPTION]
```

Como podemos ver los permisos pueden afectar a una o varias columnas, o a una tabla completa, o a todas las bases de datos (*), o a todas las tablas de todas las bases de datos (*.*) o a todas las tablas de una base de datos concreta ("BaseDatos.*").

Hasta la versión 5, cuando asignábamos un permiso a un usuario inexistente, dicho usuario se creaba automáticamente, ese es el motivo de poder indicar la contraseña después del nombre de usuario.

Si añadimos “WITH GRANT OPTION” permitimos que el usuario conceda permisos a otros usuarios

Página 283 y 284 de los apuntes “curso_mysql.pdf”

Existen multitud de permisos pero los más habituales que podemos asignar son los siguientes:

Privilegio	Significado
ALL [PRIVILEGES]	Activa todos los privilegios excepto *GRANT OPTION*.
ALTER	Permite el uso de ALTER TABLE.
CREATE	Permite el uso de CREATE TABLE.
CREATE ROUTINE	*Crear rutinas almacenadas.*
CREATE VIEW	Permite el uso de CREATE VIEW.
DELETE	Permite el uso de DELETE.
DROP	Permite el uso de DROP TABLE.
EXECUTE	*Permite al usuario ejecutar procedimientos almacenados.*
INDEX	Permite el uso de CREATE INDEX y DROP INDEX.
INSERT	Permite el uso de INSERT.
PROCESS	Permite el uso de SHOW FULL PROCESSLIST.
RELOAD	Permite el uso de FLUSH.
SELECT	Permite el uso de SELECT.
SHOW DATABASES	La sentencia SHOW DATABASES muestra todas las bases de datos.
SHOW VIEW	Permite el uso de SHOW CREATE VIEW.
SHUTDOWN	Permite el uso del apagado de *mysqladmin shutdown*.
UPDATE	Permite el uso de UPDATE.
USAGE	Sinónimo de "sin privilegios".
GRANT OPTION	Permite conceder privilegios.

1.7.3. Retirada de permisos

La orden es REVOKE cuya sintaxis se indica a continuación:

```
REVOKE TipoPermiso  TipoPermiso [(ListaColumnas)]
ON {tabla | * | *.* | BaseDatos.*}
FROM Usuario;
```

1.7.4. Mostrar privilegios

Si queremos saber todos los privilegios que soporta nuestro servidor MySQL:

```
SHOW PRIVILETES\G
```

Si queremos saber los privilegios de la conexión actual.

```
SHOW GRANTS
```

Si queremos saber los privilegios de un usuario concreto, cuya definición es el nombre con o sin nombre de host.

```
SHOW GRANTS FOR Usuario;
SHOW GRANTS FOR 'root'@'localhost'; /*ejemplo*/
```

1.7.5. Renombrar usuarios

RENAME USER UsuarioViejo TO UsuarioNuevo

1.7.6. Cambiar contraseña

```
SET PASSWORD = PASSWORD('contraseña');/*cambiar la del usuario actual*/
SET PASSWORD FOR Usuario = PASSWORD('contraseña');/*cambiar la de Usuario*/
```