
MySQL

INTRODUCCIÓN

MySql:

SGBD relacional, multihilo y multiusuario. Su origen está en mSQL

My (puede que el nombre de la hija del creador o librerías con este prefijo).

Es un entorno con Licencia GNU/GPL. Hay versiones para Windows, Linux y Mac
Actualmente en la web oficial de MySql encontramos:

MySQL Enterprise Edition.

MySQLCluster CGE.

MySQLCommunity Server.

MySQLCluster.

MySQLWorkbench.

Antes, MySql utilizaba como mecanismo de almacenamiento de datos MyIsam. Actualmente utiliza InnoDB (código abierto), el cuál permite integridad referencial en el almacenamiento de datos y soporta transacciones (conjunto de operaciones de base de datos (es decir, sentencias SQL), que son procesadas como un todo, de forma que las operaciones que están incluidas dentro de esas transacciones base de datos se validan (commit) o se cancelan (rollback) como una única operación.)

VENTAJAS DE MYSQL

1. Open Source (De momento)
2. Velocidad al realizar las operaciones SGBD con mejor rendimiento.
3. Bajo costo en requerimientos. Puede ser ejecutado en una máquina con escasos recursos.
4. Facilidad de configuración e instalación. Soporta gran variedad de Sistemas Operativos
5. Baja probabilidad de corromper datos, incluso si los errores no se producen en el propio gestor, sino en el sistema en el que está.
6. Su conectividad, velocidad, y seguridad hacen de MySQL Server altamente apropiado para acceder bases de datos en Internet
7. El software MySQL usa la licencia GPL

DESVENTAJAS MYSQL

1. Un gran porcentaje de las utilidades de MySQL no están documentadas.
2. No es intuitivo, como otros programas (ACCESS), resulta muy interesante manejarse en código SQL (MySQL es una versión limitada del estándar global), o mediante un programa que facilite un interface.

EJECUCION DE MYSQL

Para poder trabajar con MySQL y posteriormente extraer los datos y “pintarlos” en una página web, necesitamos:

-Un servidor web (Apache)
-Un interprete o lenguaje “catalizador” entre la base de datos y la página web (PHP)
-Un lenguaje que permita crear Bases de datos (MySQL)
-A poder ser, un entorno gráfico de gestión de las bases de datos (PhpMyAdmin)

PAQUETES DE INSTALACIÓN

¿Qué es LAMP?: https://www.youtube.com/watch?v=rIRusnyxlW4

MAMP: https://www.mamp.info/en/

WAMP: http://www.wampserver.com/en/

XAMPP: https://www.apachefriends.org/es/index.html

XAMPP

Servidor de plataforma libre.

Integra en una sola app:

-Servido web Apache

-Intérprete de lenguaje de scripts (PHP)

-Servidor de BBDD MySql/MariaDB

-Servidor FTP Filezilla

-Software de administración Visual (PhpMyAdmin)

Con Xampp podemos trabajar en:

-Local.
-Configuración funcional y sencilla.
-La seguridad no es su punto fuerte.

XAMPP = X (para cualquier sistema operativo), A (Apache), M (MySQL), P (PHP) y P (Perl).

La licencia de esta aplicación es GNU ((General Public License), está orientada principalmente a proteger la libre distribución, modificación y uso de software.

La filosofía de XAMPP:
- Distribución fácil de instalar, para que los desarrolladores web cuenten con todo lo necesario ya configurado.

XAMPP solamente requiere descargar y ejecutar un archivo .zip, .tar, o .exe, con unas pequeñas configuraciones en alguno de sus componentes que el servidor Web necesitará.

Una de las características sobresalientes de este sistema es que es multiplataforma.

XAMPP (ventajas y desventajas)

Facilidad de instalación y configuración básica.

Incluye otros servicios como servidor de correos y servidor FTP.

Xampp trae PhpMyAdmin para administrar las bases de datos de MySQL, sin embargo para tareas más específicas es mejor utilizar la consola (línea de comandos) y Xampp no la soporta.

Xampp trae las últimas versiones de las aplicaciones que instala, sin embargo cuando pasa el tiempo y salen nuevas versiones de las mismas, no queda otra salida que reinstalar todo Xampp.

INTERFAZ PHPMYADMIN

Con la interfaz nos permite:

- Visualizar y borrar bases de datos, tablas, vistas, campos e índices.
- Mostrar múltiples resultados a través de procedimientos almacenados o consultas.
- Crear, copiar, borrar, renombrar y alterar bases de datos, tablas, campos e índices.
- Realizar labores de mantenimiento de servidor, bases de datos y tablas, dando consejos acerca de la configuración del servidor.
- Cargar tablas con el contenido de ficheros de texto.
- Crear y leer volcados de tablas.
- Exportar datos a varios formatos: CSV, XML, PDF
- Importar datos y estructuras MySQL de plantillas OpenDocument así como también archivos XML, CSV y SQL.
- Administrar múltiples servidores.
- Gestionar privilegios y usuarios de MySQL.
- Mediante Query-by-example (QBE), crear consultas complejas conectando automáticamente las tablas necesarias.
- Crear gráficos PDF del diseño de su base de datos.
-Visualizar cambios en bases de datos, tablas y vistas.
- Capacidad de trabajar con tablas InnoDB y claves foráneas.
- Capacidad de utilizar mysqli, la extensión MySQL mejorada.
- Crear, editar, ejecutar y eliminar funciones y procedimientos almacenados («storedprocedures»).
- Crear, editar, exportar y eliminar eventos y disparadores.
- Comunicarse en 62 idiomas distintos

Para arrancar PHPMYADMIN:

Poner directamente en el navegador http://localhost/phpmyadmin.

Desde el Control Panel de XAMPP, pulsar el botón Admin

LENGUAJE SQL (Structure Query Languaje)

SQL:DEFINICIÓN DE DATOS (DDL)

Parte del Lenguaje SQL proporcionado por el SGBD que permite a los usuarios definir las estructuras que almacenarán los datos así como de los procedimientos o funciones que permitan consultarlos.

El DDL (Data Definition Language) es la parte del SQL que más varía de un sistema a otro ya que esa área tiene que ver con cómo se organizan internamente los datos y eso, cada sistema lo hace de una manera u otra.

CREACIÓN DE BASES DE DATOS

Para crear el contendor de nuestras tablas, vistas y en general "objetos", definimos la creación de una nueva base de datos.

Para crear la Base de datos emplearemos o bien el interface gráfico del que dispongamos (Php MyAdmin) o bien la consola MySql, o escribimos un fichero con extensión *sql para posteriormente importarlo a través del entorno.

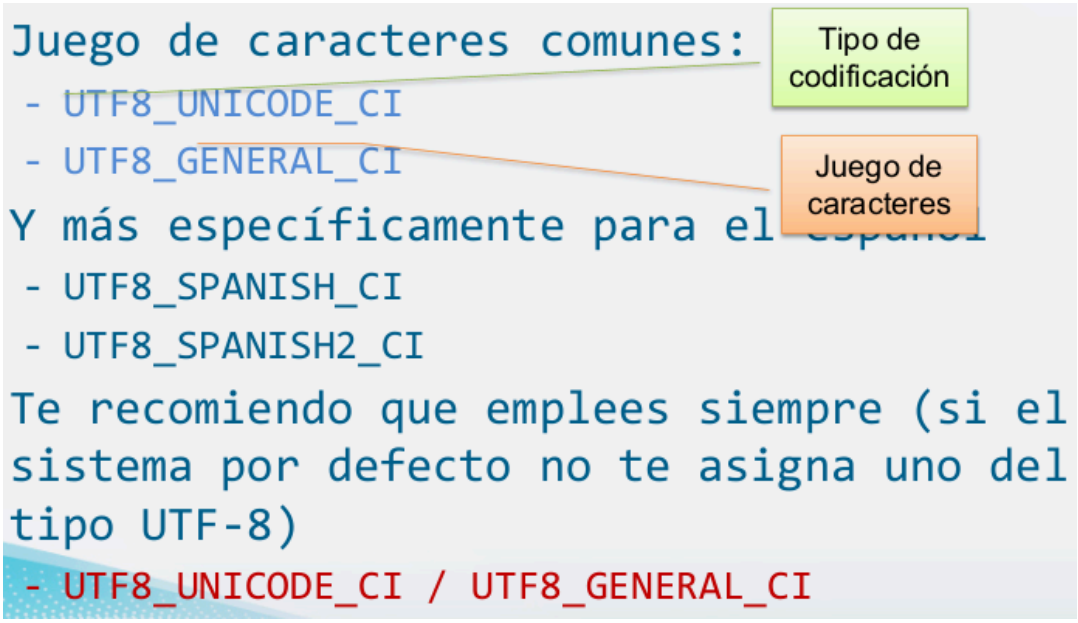
Si escribimos los comandos terminaremos siempre con ;

Comandos habituales

Desde la consola/fichero podemos:

Create database nombre_de_la_base_datos;

Lo normal es que el juego de caracteres sea el adecuado, sirviéndote varios tipos, el más global es UTF8_UNICODE, pero serviría cualquier conjunto de caracteres que incluya las particularidades del idioma sobre el que va a estar basado el proyecto, para el español servirían los siguientes juegos de caracteres



La creación de la base de datos definiendo el juego de caracteres sería, en su forma más sencilla.

Create database nombre_de_la_base_datos DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;

Create database ventas DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;

Operaciones Generales

Borrado de una Base de datos: **DROP**

Drop database nombre_de_la_base_datos

Drop database ventas;

Conectarse a una Base de datos para lacreación de objetos (Tablas, vistas…):**USE**

Use nombre_de_la_base_datos;

Use ventas;

TIPOS DE DATOS DE UNA BBDD

Definen el tipo de contenido que va a almacenar las celdas de las tablas.

Representación de la información:
Símbolos numéricos
Alfanuméricos
Formatos de fecha, hora, binarios, etc.

Todas estas clases o divisiones son tipos de datos.

TD Numérico: Cantidad de dinero que disponemos en una cuenta bancaria en Suiza.

TD Fecha: Nuestra fecha de cumpleaños

TIPOS DE DATOS PARA EL ALMACENAMIENTO Y GESTIÓN DE LA INFORMACIÓN & CREACIÓN DE TABLAS

Los tipos de datos que puede haber en un campo, se pueden agrupar en tres grandes grupos:

1.- Tipos numéricos
2.- Tipos de Fecha
3.- Tipos de Cadena

TD NUMÉRICOS

Podemos dividirlos en dos grandes grupos:

-Con coma flotante
-Sin coma flotante

TinyInt: Entero con o sin signo. Con signo el rango de valores válidos va desde -128 a 127. Sin signo, el rango de valores es de 0 a 255
Bit ó Bool: un número entero que puede ser 0 ó 1
SmallInt: número entero con o sin signo.
MediumInt: número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607.
Integer, Int: número entero con o sin signo. Sin signo el rango va desde 0 a 429.4967.295
BigInt: número entero con o sin signo. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.
Float: número pequeño en coma flotante de precisión simple.
xReal, Double: número en coma flotante de precisión doble.
Decimal, Dec, Numeric: Número en coma flotante desempaquetado. El número se almacena como una cadena

TD FECHA

Date
DateTime
TimeStamp
Time
Year

TD CADENA

Char(n)
VarChar(n)
TinyText
TinyBlob
Blob y Text

MediumBlob
MediumText
LongBlob
LongText
Enum
Set

REFERENCIA CLARA A LOS DIFERENTES TIPOS DE DATOS.
http://www.w3schools.com/sql/sql_datatypes.asp

CREACIÓN DE TABLAS: CREATE

CREATE TABLE nbtabla (nbcol tipo restricción_tipo1 ó restricción_tipo2, …);

Restricción de columna: Aparece dentro de la definiciónde la columna después del tipo de dato y afecta a una columna, la que se está definiendo (por ejemplo que su contenido no sea null)

Restricción de tabla: Se define después de definir todas las columnas de la tabla y afecta a una columna o a una combinación de columnas. (por ejemplo la definiciónde una clave principal, para varias columnas o el tipo de motor para la tabla)

Vista del sistema para ver las tablas existentes en la BD actual: SHOW

Show tables;

Creación de tabla usuario con los campos, nombre y clave

Create table usuarios (nombre varchar(30), clave varchar(10));

Ver la definición estructural de una tabla, mediante: DESCRIBE

describe usuarios;

Eliminar una tabla al completo: DROP

drop table usuarios;

Renombrar una tabla: (Sólo nombre de tabla): RENAME

RENAME TABLE tbl_nomb TO nuevo_tbl_nomb;

RENAME TABLE usuarios TO users;

CREACIÓN DE TABLAS My(SQL)

Comprobar la existencia de la tabla (también para bases de datos) antes de proceder a su creación para evitar sobreescrituras.

…if not exists…

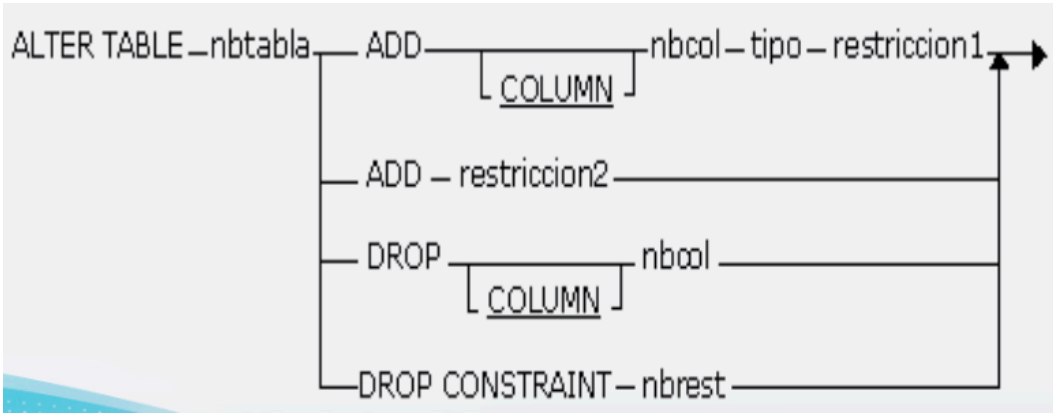
```
create table if not exists productos
        id int unsigned primary key,
        titulo varchar(100) not null default '',
        precio decimal(12, 2) not null default 0,
        fecha_alta timestamp default current_timestamp);
```

MODIFICACIÓN DE TABLAS: ALTER TABLE

¿Para que se utiliza?

-agregar nuevos campos
-eliminar campos existentes
-modificar el tipo de dato deun campo
-agregar o quitar modificadores como "null", “unsigned", “auto_increment"
-cambiar el nombre de un campo
-agregar o eliminar la clave primaria
-agregar y eliminar índices
-renombrar una tabla

Estructura de ALTER TABLE:



Añadir un nuevo campo (Columna): ADD

alter table libros add cantidad smallint unsigned not null;

--

alter table libros add cantidad tinyint unsigned after autor;

Eliminar un campo (Columna): DROP

alter table libros drop editorial;

Hay que tener cuidado al eliminar un campo, éste puede ser clave primaria. Es posible eliminar un campo que es clave primaria.

Modificar valores de los campos: MODIFY

alter table libros modify cantidad smallint unsigned;

--

alter table libros modify titulo varchar(10) not null;

OJO al alterar los tipos de los campos de una tabla que ya tiene registros cargados.

Campo de texto de longitud 50 a 30 de longitud, los registros cargados en ese campo que superen los 30 caracteres, se cortarán.

Campo que fue definido permitiendo valores nulos, se cargaron registros con valores nulos y luego se lo define "not null", los registros con ese campo NULO cambiarán al valor por defecto según el tipo (cadena vacía para tipo texto y 0 para numéricos), ya que "null" se convierte en un valor inválido.

INDICES: TIPO Y GESTIÓN

Parecido al índice de un libro donde tenemos los capítulos del libro y la página donde empieza cada capítulo.

Permite recuperar las filas de una tabla de forma más rápida además de proporcionar una ordenación distinta a la natural de la tabla.

y las f**Un índice se define sobre una columna o sobre un grupo de columnas,** ilas se ordenarán según los valores contenidos en esas columnas. Por ejemplo, si definimos un índice sobre la columna *población* de la tabla de *clientes*, el índice permitirá recuperar los clientes ordenados por orden alfabético de población.

En índices **sobre varias columnas**, los registros se ordenarán **por la primera columna definida, dentro de un mismo valor de la primera columna se ordenarán por la segunda columna,** y así sucesivamente.

Ventajas:

Si una tabla tiene definido un índice sobre una columna se puede localizar mucho más rápidamente una fila que tenga un determinado valor en esa columna.

Recuperar las filas de una tabla de forma ordenada por la columna en cuestión también será mucho más rápido.

Desventajas:

Al ser el índice una estructura de datos adicional a la tabla, ocupa un poco más de espacio en disco.

Cuando se añaden, modifican o se borran filas de la tabla, el sistema debe actualizar los índices afectados por esos cambios lo que supone un tiempo de proceso mayor.

No es aconsejable definir índices de forma indiscriminada.

Las desventajas comentadas no son nada comparados con las ventajas si la columna sobre la cual se define el índice es una columna que se va a utilizar a menudo para buscar u ordenar las filas de la tabla.

Tipos de indices:

"primary key": es el que definimos como clave primaria. Los valores indexados deben ser únicos y además no pueden ser nulos. SQL le da elnombre "PRIMARY".

Una tabla solamente puede tener una clave primaria.

Una clave primaria puede estar formada por varios campos

"index": crea un índice común, *los valores no necesariamente son únicos y*aceptan valores "null".

Podemos darle un nombre (etiquetarlo, alias), si no se lo damos, se coloca uno por defecto. "key" es sinónimo de "index", puedo emplearlos indistintamente. Puede haber varios por tabla.

"unique": Los valores *deben ser únicos y diferentes,* aparece un mensaje de error si intentamos agregar un registro con un valor ya existente.

Permite valores nulos y pueden definirse varios por tabla. Podemos darle un nombre, si no se lo damos, se coloca uno por defecto.

Todos los índices pueden ser multicolumna, es decir, pueden estar formados por más de 1 campo.

En MySQL otros dos tipos de índices son:

FULLTEXT: Permiten realizar búsquedas de palabras. Sólo pueden usarse sobre columnas CHAR, VARCHAR o TEXT

SPATIAL: Este tipo de índices solo puede usarse sobre columnas de datos geométricos (spatial) y en el motor MyISAM.

CREACIÓN DE ÍNDICE PRIMARIO

create table libros(codigo int unsigned auto_increment, titulo varchar(40) not null, autor varchar(30), editorial varchar(15), primary key(codigo));

Para ver el índice

Show index from libros;

CREACIÓN DE ÍNDICE COMÚN (Normal)

create table libros(código int unsigned auto_increment, titulo varchar(40) not null, autor varchar(30), editorial varchar(15),

--Definimos índices al final de la tabla

primary key(codigo), index i_editorial (editorial));

CREACIÓN DE ÍNDICES ÚNICOS

create table libros(codigo int unsigned auto_increment, titulo varchar(40) not null, autor varchar(30), editorial varchar(15),

--Definimos índices al final de la tabla

index i_titulo(titulo), unique i_tituloeditorial (titulo, editorial));

RENOMBRAR INDICES

En ocasiones puede resultar útil nombrar los índices con un “mote” o “alias” para hacer más sencilla su utilización en determinadas operaciones de codificación.

Se crean empleando la palabra reservada **CONSTRAINT,** seguida del nombre que queramos (generalmente **pk_NombreAlias** para las principales y **fk_NombreAlias** para las ajenas) cuando se crean los índices.

```
create table producto_imagen (id_prodimg int (10) unsigned, id_prod int (10) unsigned
not null, ruta_imagen varchar (255) not null, constraint pk_id_prodimg primary key
(id_prodimg), constraint fk_id_prod foreign key (id_prod) references producto (id_prod) );
```

Tipo	Nombre	Palabra clave	Valores únicos	Acepta null	Cantidad por tabla
clave primaria	PRIMARY	no	Si	No	1
común	darlo o por defecto	"index" o "key"	No	Si	varios
único	darlo o por defecto	"unique"	Si	Si	varios

CREACIÓN DE ÍNDICES EN TABLAS EXISTENTES

Para agregar un índice común a una tabla existente usamos **"create index"**, indicamos el nombre, sobre qué tabla y el o los campos por los cuales se indexará, entre paréntesis.

```
Create index i_editorial on libros (editorial);
```

Para agregar un índice único a una tabla existente usamos **"create unique index"**, indicamos el nombre, sobre qué tabla y entre paréntesis, el o los campos por los cuales se indexará.

```
Create unique index i_tituloeditorial on libros (titulo,editorial);
```

Un índice **PRIMARY** no puede agregarse, se crea automáticamente al definir una clave primaria.

BORRADO DE ÍNDICES

```
drop index i_editorial on libros;
drop index i_tituloeditorial on libros;
```

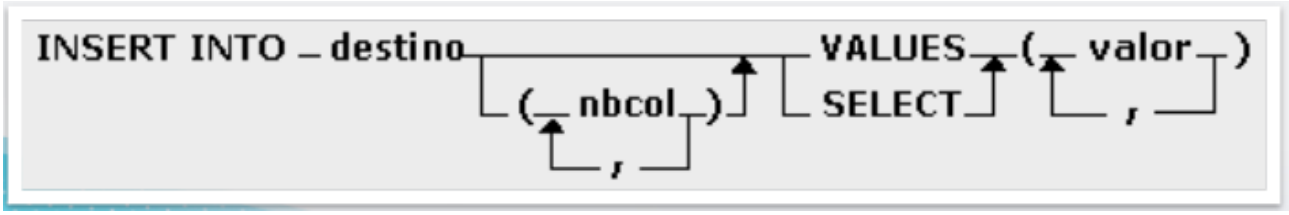
Se elimina el índice con **"drop index”** seguido de su nombre y **"on"** seguido del nombre de la tabla a la cual pertenece.

Podemos eliminar los índices creados con **"index"** y con **"unique"** pero no el que se crea al definir una clave primaria. **Un índice PRIMARY se elimina automáticamente al eliminar la clave primaria**

INSERCIÓN DE REGISTROS

Se trata de añadir filas enteras a una tabla. Se pueden insertar una o varias filas de golpe.

El esquema sintáctico sería el siguiente:



Cuando **no se indica ninguna lista de columnas después de la tabla destino,** se asume por defecto todas las columnas de la tabla, en este caso, los valores se tienen que especificar en el mismo orden en que aparecen las columnas en la ventana de diseño de dicha tabla, y se tiene que utilizar el valor NULL para rellenar las columnas de las cuales no tenemos valores.

```
INSERT INTO empleados VALUES (200, 'Juan López', 30, NULL, 'rep ventas', #06/23/01#, NULL,
        350000, 0);
```

OPCIONES INSERCIÓN

Escribimos una vez la sentencia INSERT INTO y separamos las distintas inserciones (tuplas) por una coma.

```
insert into colegios (cod_colegio, Nombre,Direccion,cp) values (4,'Celestino',
        'cSalSipuedes','28001'), (5,'Celestino','c/SalSipuedes','28001');
```

Si el primer campo a completar es autoIncrement y no queremos estar pendiente del número correspondiente omitimos el campo en el primer paréntesis

```
insert into colegios (Nombre,Direccion,cp) values ('AAA','BBB','Varios'),
        ('AAA','BBB','Varios');
```

Repetimos la sentencia INSERT por cada registro finalizando cada línea con punto y coma;

```
insert into colegios (cod_colegio, Nombre,Direccion,cp) values (4,'Celestino',
        'cSalSipuedes','28001'); insert into colegios (cod_colegio, Nombre,Direccion,cp)
        values (5,'Celestino','c/SalSipuedes','28001');
```

POSIBLES ERRORES DE INSERCIÓN

Si la tabla de destino tiene PK y en ese campo intentamos no asignar valor, asignar el valor nulo o un valor que ya existe en la tabla, el motor de base de datos no añade la fila y da un mensaje de error de 'infracciones de clave'.

Si tenemos definido un índice único (sin duplicados) e intentamos asignar un valor que ya existe en la tabla también devuelve el mismo error.

Si la tabla está relacionada con otra, se seguirán las reglas de integridad referencial.

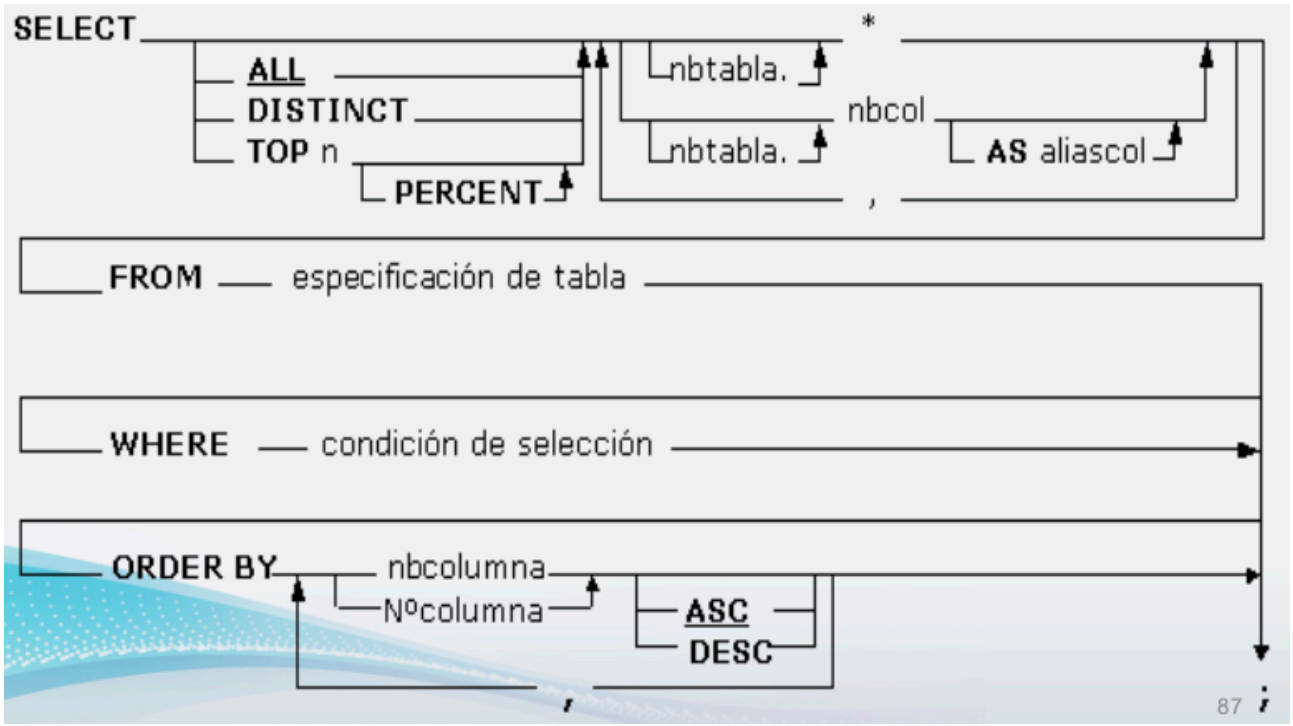
CONSULTAS DE REGISTROS - Select

Permite recuperar datos de una o varias tablas. Es la sentencia más compleja y potente de las sentencias SQL. Empezaremos viendo las consultas más simples, basadas en una sola tabla.

El resultado de la consulta es una tabla lógica, porque no se guarda en el disco sino que está en memoria y cada vez que ejecutamos la consulta se
vuelve a calcular. (MySql)

En otros sistemas de BD SI se guarda en disco el resultado de la consulta.(Access)

Cuando ejecutamos la consulta se visualiza el resultado en forma de tabla con columnas y filas, pues en la SELECT tenemos que indicar qué columnas queremos que tenga el resultado y qué filas queremos seleccionar de la tabla origen.



Con la cláusula FROM indicamos en qué tabla tiene que buscar la información. En este capítulo de consultas simples el resultado se obtiene de una única tabla, pero evidentemente podemos extraer información de un número n de tablas estén o no relacionadas.

SELECT Nombre, codPostal FROM municipio;

ALIAS EN UNA SELECT

Alias de tabla es un segundo nombre que asignamos a la tabla. Si en una consulta definimos un alias para la tabla, esta se deberá nombrar utilizando ese nombre y no su nombre real, además ese nombre sólo es válido en la consulta donde se define.

Ejemplo: SELECT .....FROM oficinas ofi ;
equivalente a SELECT .....FROM oficinas AS ofi

Los datos se buscan en la tabla *oficinas* que queda renombrada en esta consulta con *ofi.*

SELECCIÓN DE COLUMNAS

La lista de columnas que queremos que aparezcan en el resultado es lo que llamamos lista de selección y se especifica delante de la cláusula FROM.

Ej. *SELECT Nombre, cod_colegio FROM colegios*

Se utiliza el asterisco * en la lista de selección para indicar **'todas las columnas de la tabla'**.

ALIAS DE COLUMNA

Al visualizar el resultado de la consulta, las columnas toman el nombre que tiene la columna en la tabla. Para cambiar ese nombre definimos un alias de columna mediante la cláusula **AS** será el nombre que aparecerá como título de la columna.

Ejemplo:

SELECT Nombre AS N, cod_colegio AS code FROM colegios

COLUMNAS CALCULADAS

Una consulta SQL puede incluir **columnas calculadas** cuyos valores se calculan a partir de los valores de los datos almacenados.

Para crear una columna calculada, se especifica en la lista de selección una **expresión** en vez de un nombre de columna.

SELECT ciudad, región, (ventas+objetivo) AS superavit FROM oficinas

SELECT idfab, idproducto, descripcion, (existencias * precio) AS valoracion FROM productos

Podemos emplear en las Select funciones del lenguaje

SELECT titulo, MONTH(fec_public), YEAR(fec_public) FROM libros

Lista el nombre, mes y año del contrato de cada vendedor.

La función MONTH() devuelve el mes de una fecha. La función YEAR() devuelve el año de una fecha.

SELECT DISTINCT

Con la cláusula "distinct" se especifica que los registros con ciertos datos duplicados sean obviadas en el resultado. Por ejemplo, queremos conocer todos los autores de los cuales tenemos libros.

Evidentemente tenemos más de un libro escrito por el mismo autor pero sólo queremos conocer los diferentes autores

select autor from libros;

Aparecen repetidos. Para obtenerla lista de autores sinrepetición usamos:

select distinct autor from libros;

Si hacemos DISTINCT en un select, nos mostrará registros con valores no duplicados **AL COMPLETO** para los campos en cuestión.

—> *Select distinct autor, editorial from libros;*

—> *SELECT distinct provincia, nombre FROM municipios;*

ORDENACION DE LAS FILAS: ORDER BY

SELECT código, titulo, autor FROM libros ORDER BY código

es equivalente a:

SELECT código, titulo, autor FROM libros ORDER BY 1

ASC (por defecto) o DESC

SELECT codigo_mun, provincia, nombre FROM municipios ORDER BY 3 DESC

CLÁUSULA WHERE

La cláusula WHERE selecciona únicamente las filas que cumplan la condición de selección especificada.

En la consulta sólo aparecerán las filas para las cuales la condición es verdadera (TRUE).

Para construir las condiciones tenemos los llamados criterios y la combinación de estos en expresiones

```
SELECT nombre FROM empleados WHERE oficina = 12
```

Lista el nombre de los empleados de la oficina 12.

```
SELECT nombre FROM empleados WHERE oficina = 12 AND edad > 30
```

Lista el nombre de los empleados de la oficina 12 que tengan más de 30 años.

CLÁUSULA IN

La sentencia IN se usa con WHERE para indicar valores a incluir en los resultados de la consulta. Por ejemplo, queremos ver los DISTRICT (Provincias, Estados -no países-) a los que pertenecen las ciudades de Madrid, New York y Roma:

```
SELECT Name, District
        FROM City
        WHERE Name IN ("Madrid","New York","Roma");
```

CLÁUSULA LIMIT

Mediante el uso de clausula LIMIT sacamos sólo una parte del resultado de la ejecución de una consulta. Muy útil cuando la salida de registros es bastante elevada. Puede darse con uno o dos argumentos:

- LIMIT num_filas
- LIMIT num_saltar, num_filas

Veamos un ejemplo de cada uno:

```
SELECT Name FROM Country          (Devuelve 239 Registros)

SELECT Name FROM Country
        LIMIT 5;                  (Devuelve 5 Registros)
```

Ahora otro ejemplo que ya vimos:

```
SELECT DISTINCT Continent FROM Country          (Salen 7 Continentes.)
```

Pues bien, queremos omitir los 2 primeros y presentar los siguientes 5.

```
SELECT DISTINCT Continent FROM Country
        LIMIT 2,5;
```

Veamos otros dos ejemplos incluyendo el operador de comparación >

```
SELECT Name, Population FROM Country
        WHERE Population > 35000000
        AND (Continent = "Europe" OR Continent = "North America")
        ORDER BY Population DESC;
```

CLÁUSULA LIKE (Comparar patrones)

La comparación entre patrones se realiza con la palabra reservada LIKE. Sintaxis:

expresión LIKE patrón

Normalmente LIKE lo usaremos con los caracteres especiales _ (guión bajo) y % (porcentake):

1. % equivale te a cualquier número de caracteres. Ejemplos:
    a) 'a%' → Equivale a una cadena que empiece por a
    b) '%b' → Equivale a una cadena que termine en b
    c) '%c% → Cualquier cadena que contenga c
2. _ equivale a cualquier carácter simple. Por ejemplo:
    a) 'as_' puede equivaler a... así, asa, as@, as?, etc

Un ejemplo:

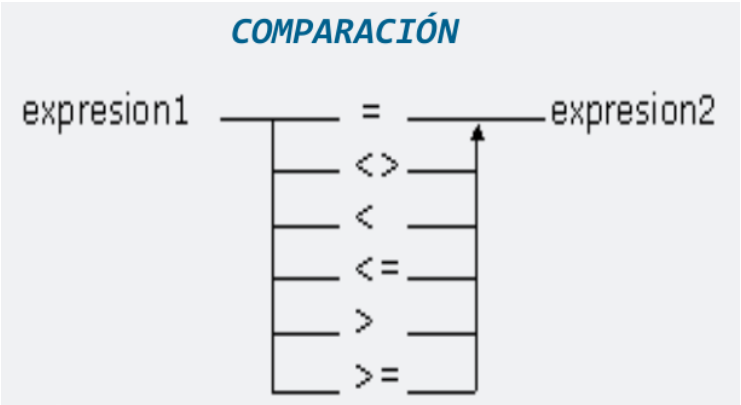
```
USE world;
SELECT name FROM Country
        WHERE name LIKE ‘United%';          (Salen 4 registros.)
```

Para invertir los patrones de comparación, debemos usar NOT LIKE.

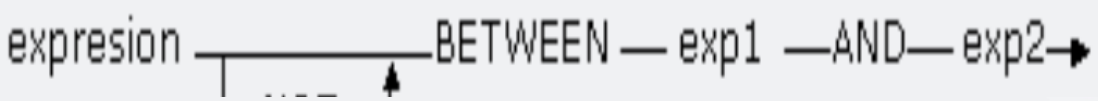
Ejemplo:

```
SELECT name FROM Country
        WHERE name NOT LIKE ‘United%';   (Salen 235 registros.)
```

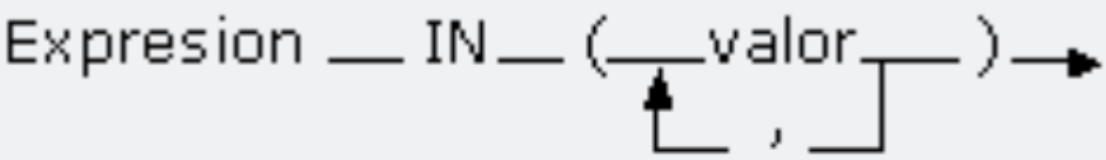
CONDICIONES DE SELECCIÓN



RANGO



PERTENENCIA A UN CONJUNTO

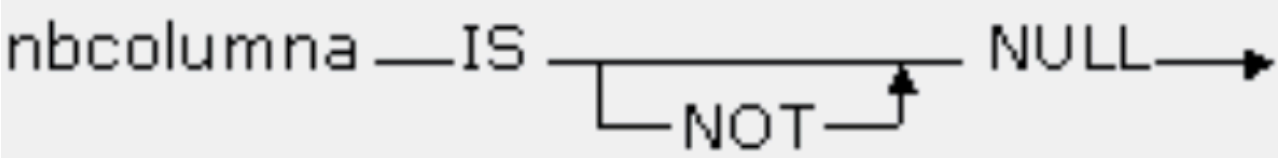


SELECT numemp, nombre, oficina FROM empleados WHERE oficina IN (12,14,16)

Otra opción, más tediosa

SELECT numemp, nombre FROM empleados WHERE (oficina = 12) OR (oficina = 14)
OR (oficina = 16)

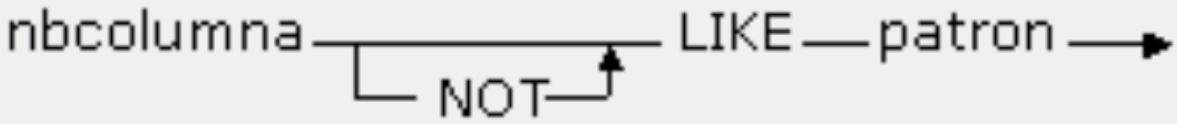
VALORES NULOS




```
# SELECT distinct(autor) FROM libros
   WHERE autor IS NOT NULL
```

PATRÓN

¿¿Que es un patrón??




```
SELECT * FROM hijos
      WHERE hijoNombre LIKE '%ar%'
```

operadores de comparación útiles para cláusulas WHERE

Operador	*Nombre (Si aplicable)*	*Ejemplo*	*Descripción*
=	**Igual**	**Clienteid=3**	Comprueba si dos valores son iguales
>	**Mayor que**	**Cantidad** > 40.000	Comprueba si un valor es mayor que otro
<	**Menor que**	**Cantidad** < 40.000	Comprueba si un valor es menor que otro
>=	**Mayor o igual que**	**Cantidad** >= 40.000	Comprueba si un valor es mayor o igual que otro
<=	**Menor o igual que**	**Cantidad** <= 40.000	Comprueba si un valor es menor o igual que otro
!= or <>	**No igual**	**Cantidad** != 0	Comprueba si dos valores no son iguales
IS NOT NULL	**direccion is NOT NULL**		Comprueba si un campo realmente contiene un valor
IS NULL	**direccion is NULL**		Comprueba si un campo no contiene un valor
BETWEEN	**Cantidad between 0 y 60.000**		Comprueba si un valor es mayor o igual a un valor mínimo y menor o igual que un valor máximo
IN	**Ciudad in ("Málaga","Barna")**		Comprueba si un valor está en un grupo particular
NOT IN	**Ciudad not in ("Málaga","Barna")**		Comprueba si un valor no está en el grupo
LIKE	**Patrón de concordancia**	**Nombre LIKE** ("Fred %")	Comprueba si un valor concuerda con un patrón usando un patrón de concordancia simple SQL
NOT LIKE	**Patrón de concordancia**	**Nombre NOT LIKE** ("Fred %")	Comprueba si un valor no concuerda con un patrón de concordancia
REGEXP	**Expresión Regular**	**Nombre REGEXP**	Comprueba si un valor concuerda con una expresión regular

CONSULTAS MULTITABLA (UNIÓN)

La Unión de dos (o más) conjuntos es una operación que resulta en otro conjunto, cuyos elementos son los elementos de los conjuntos iniciales.

P={2,4,6,…}
I={1,3,5,…}
N={1,2,3,4…}

Tenemos dos tablas con las mismas columnas y queremos obtener una nueva tabla con las filas de la primera y las filas de la segunda.

Duplica la tabla personas, y nómbrala persona_ac.

Sobre la tabla Duplicada

Edita los registros cuyo PA es <8000 y asígnales ese valor.(son 5)

La tabla resultante tiene las mismas columnas (TODAS O PARTE) que la primera tabla (que son las mismas que las de la segunda tabla).

ALL	Se supone si no incluye uno de los predicados. El motor de base de datos selecciona todos los registros que cumplen las condiciones de la instrucción SQL. Los dos ejemplos siguientes son equivalentes y devuelven todos los registros de la tabla Empleados:
	
	`SELECT ALL *`
	`FROM Employees`
	`ORDER BY EmployeeID;`
	
	`SELECT *`
	`FROM Employees`
	`ORDER BY EmployeeID;`

DISTINCT	Omite los registros que contienen datos duplicados en los campos seleccionados. Para poder ser incluido en los resultados de la consulta, los valores de cada campo enumerado en la instrucción SELECT deben ser únicos. Por ejemplo, puede que varios empleados enumerados en una tabla Empleados tengan el mismo apellido. Si existen dos registros que contienen Pérez en el campo Apellido, la siguiente instrucción SQL devuelve solo uno de ellos:
	
	`SELECT DISTINCT LastName`
	`FROM Employees;`
	
	Si se omite DISTINCT, esta consulta devuelve ambos registros.
	
	Si la cláusula SELECT contiene más de un campo, la combinación de valores de todos los campos debe ser única para que un registro determinado pueda incluirse en los resultados.
	
	El resultado de una consulta en la que se usa DISTINCT no se puede actualizar y no refleja los cambios posteriores efectuados por otros usuarios.

Si **no se usa** la palabra clave **ALL** para **UNION**, todos los registros retornados son únicos, como si hubiera hecho un **DISTINCT** para el conjunto de resultados total.

Cada parte de la UNION tiene que tener al menos el mismo número de columnas y estas tienen que ser del mismo tipo.

CONSULTAS MULTITABLA (PRODUCTO CARTESIANO)

En matemáticas, el producto cartesiano de dos conjuntos es una operación, que resulta en otro conjunto, cuyos elementos son todos los pares ordenados, que pueden formarse tomando el primer elemento del par, del primer conjunto, y el segundo elemento, del segundo conjunto.

Por ejemplo, dados los conjuntos A ={1, 2, 3, 4} y B = {a, b}, su producto cartesiano es:

$$AxB=\{(1,a), (1,b), (2,a), (2,b), (3,a), (3,b), (4,a), (4,b)\}$$

Aplicado a dos tablas se obtiene una tabla con las columnas de la primera tabla unidas a las columnas de la segunda tabla, y las filas de la tabla resultante son todas las posibles combinaciones de filas de la primera tabla con filas de la segunda tabla.

```
select * from persona, persona_ac;
```

¡¡OJO!! El producto cartesiano obtiene todas las posibles combinaciones de filas

El producto cartesiano de dos tablas de 100 registros cada una, serán 100x100 filas. si el producto lo hacemos de estas dos tablas con una tercera de 20 filas, el resultado tendrá 200.000

filas. Exige más a los sgbdr sobretodo si operamos con más de dos tablas o con tablas voluminosas.

```
SELECT municipio.*, persona.Nombre, persona.dni
        FROM persona, municipio
        WHERE municipio.Id_Municipio=persona.Id_Municipio
        ORDER BY municipio.Provincia ASC
```

Esta operación no es de las más utilizadas, porque es menos eficiente que empleando otras técnicas (JOIN)

CONSULTAS MULTITABLA (INNER JOIN)

Permite emparejar filas de distintas tablas de forma más eficiente que con el producto cartesiano cuando una de las columnas de emparejamiento está indexada.

Las consultas con JOIN ON y sus variaciones son más eficientes porque en vez de hacer el producto cartesiano completo y luego seleccionar la filas que cumplen la condición de emparejamiento, para cada fila de una de las tablas busca directamente en la otra tabla las filas que cumplen la condición, con lo cual se emparejan sólo las filas que luego aparecen en el resultado.

```
SELECT municipio.*, persona.Nombre, persona.dni
        FROM municipio INNER JOIN persona
        ON municipio.Id_Municipio=persona.Id_Municipio
```

WHERE me queda libre para otros criterios. En resumen, INNER JOIN, es una forma maseficiente de hacer un producto cartesiano entre tablas.

JOIN Vs INNER JOIN

Ambas palabras son equivalentes, INNER JOIN corresponde al intento de hacer el lenguaje "más asequible" al ser humano, lo que se conoce como "syntactic sugar", **es decir basta con JOIN.**

Podemos encontrar INNER JOIN con varias condiciones, aunque resultará más cómodo sacar las condiciones a la línea de WHERE

```
SELECT municipio.*, persona.Nombre, persona.dni
        FROM municipio INNER JOIN persona
        ON (municipio.Id_Municipio=persona.Id_Municipio)
        AND (Provincia LIKE 'M%')
```

```
SELECT municipio.*, persona.Nombre, persona.dni
        FROM  municipio INNER JOIN persona
        ON (municipio.Id_Municipio=persona.Id_Municipio)
        WHERE (Provincia LIKE ‘M%')
```

Al extraer las diferentes condiciones a la línea de WHERE la Consulta (Query), queda más limpia y fácil de gestionar.

```
SELECT municipio.*, persona.Nombre, persona.dni
        FROM municipio INNER JOIN persona
        ON (municipio.Id_Municipio=persona.Id_Municipio)
        WHERE (Provincia LIKE 'M%') AND num_hab
        BETWEEN  10000 AND 25000
```

Un ejemplo de unir 3 tablas en un JOIN seria:

```
SELECT NombreCiudad, NombrePais,Lengua
        FROM CiudadSimple
        JOIN PaísSimple ON CodPais = Codigo
        JOIN LenguajeSimple ON Codigo=CodPaísLengua;
```

CONSULTAS MULTITABLA (LEFT Y RIGHT JOIN)

El producto cartesiano y el INNER JOIN son composiciones internas ya que todos los valores de las filas del resultado son valores que están en alguna de las tablas que se combinan.

Si queremos que aparezcan las filas que no tienen una fila coincidente en la otra tabla, utilizaremos el LEFT o RIGHT JOIN.

LEFT JOIN —> Esta operación consiste en añadir al resultado de INNER JOIN las filas de la tabla de la izquierda que no tienen correspondencia en la otra tabla, y rellenar en esas filas los campos de la tabla de la derecha con valores nulos.

```
SELECT act.autor, act.titulo, aut.nombre_autor, aut.fec_nacimiento
        FROM libros_act  AS act
        LEFT JOIN autores aut
        ON act.autor = aut.nombre_autor
```

```
SELECT * FROM libros_act AS act
        LEFT JOIN autores aut
        ON act.autor = aut.nombre_autor
```

RIGHT JOIN —> Esta operación consiste en añadir al resultado del INNER JOIN las filas de la tabla de la DERECHA que no tienen correspondencia en la otra tabla, y rellenar en esas filas los campos de la tabla de la IZQUIERDA con valores nulos.

```
SELECT act.autor, act.titulo, aut.nombre_autor, aut.fec_nacimiento
        FROM libros_act AS act
        RIGHT JOIN autores AS aut
        ON act.autor = aut.nombre_autor
```

```
SELECT * FROM libros_act AS act
        RIGHT JOIN autores AS aut
        ON act.autor = aut.nombre_autor
```

Ejemplo de INNER JOIN:

Tabla Personas

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	2

Tabla Departamento

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

```
SELECT nombre, apellido1, departamento FROM personas INNER JOIN
departamentos WHERE personas.dep = departamentos.dep
```

nombre	apellido1	departamento
ANTONIO	PEREZ	ADMINISTRACION
ANTONIO	GARCIA	INFORMATICA
PEDRO	RUIZ	INFORMATICA

Ejemplo LEFT JOIN:

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	4

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

```
SELECT nombre, apellido1, departamento
FROM personas
LEFT JOIN departamentos
WHERE personas.dep = departamentos.dep
```

nombre	apellido1	departamento
ANTONIO	PEREZ	ADMINISTRACION
ANTONIO	GARCIA	INFORMATICA
PEDRO	RUIZ	

Aunque el departamento '4' de PEDRO RUIZ no existe en la tabla de departamentos, devolverá la fila con esa columna 'departamento' en blanco.

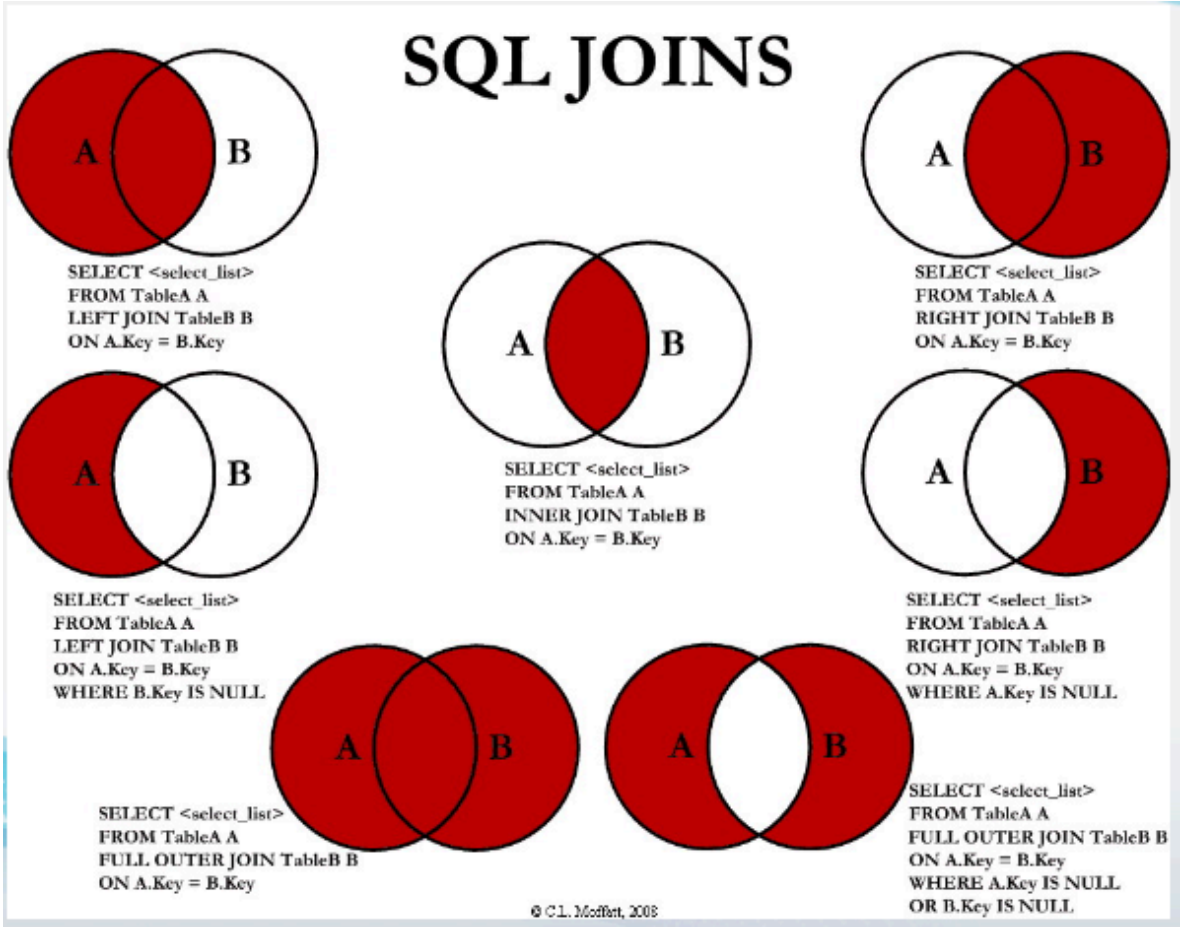
Ejemplo de RIGHT JOIN:

per	nombre	apellido1	apellido2	dep
1	ANTONIO	PEREZ	GOMEZ	1
2	ANTONIO	GARCIA	RODRIGUEZ	2
3	PEDRO	RUIZ	GONZALEZ	4

dep	departamento
1	ADMINISTRACION
2	INFORMATICA
3	COMERCIAL

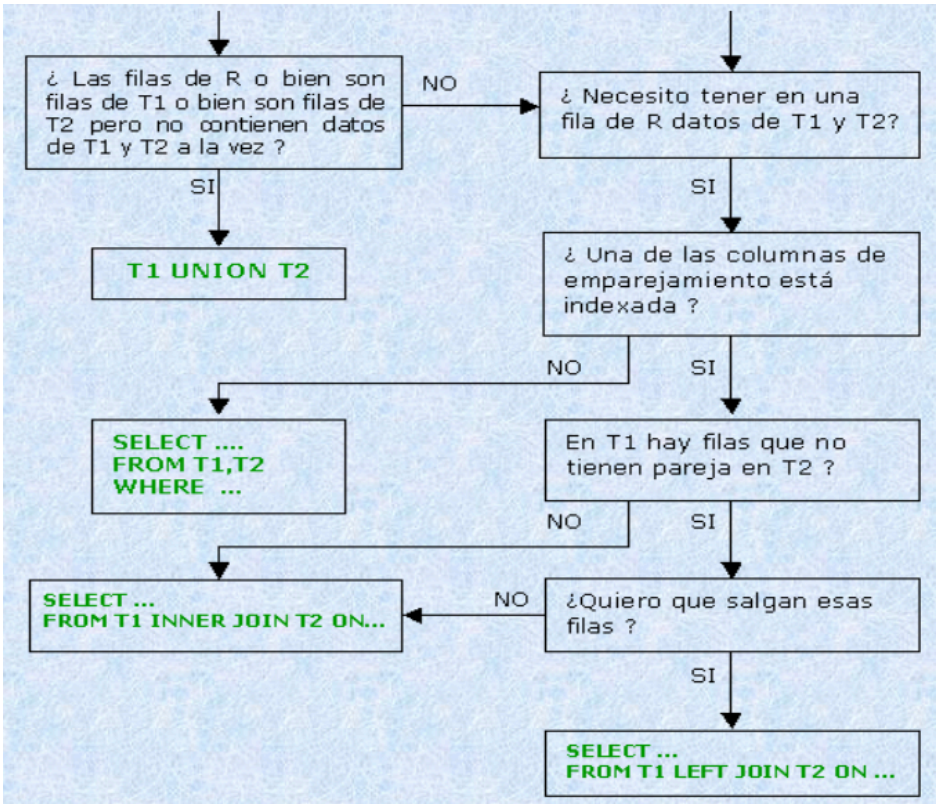
```
SELECT nombre, apellido1, departamento
FROM personas
RIGHT JOIN departamentos
WHERE personas.dep = departamentos.dep
```

nombre	apellido1	departamento
ANTONIO	PEREZ	ADMINISTRACION
ANTONIO	GARCIA	INFORMATICA
		COMERCIAL



¿Qué tipo de operación se debe utilizar en cada caso?

Hemos llamado **T1** y **T2** las tablas de las que queremos sacar los datos y **R** la tabla lógica que representa el resultado de consulta.



Hemos visto cómo usar ON junto a un WHERE en la consulta anterior. Sin embargo ON, dentro de un JOIN puede realizar el mismo papel que un WHERE, añadiendo cláusulas AND adicionales.

Un ejemplo:

```
SELECT City.Name AS Ciudad, Country.Name AS País, City.Population AS Población
    FROM City JOIN Country
    ON CountryCode = Code
    AND City.Name = 'León';
```

Incluso podemos usar expresiones con el anterior AND:

```
SELECT City.Name, Country.Name, City.Population
    FROM City JOIN Country
    ON CountryCode = Code
    AND City.Name = 'León'
    AND City.Population > 1000000;
```

De todos modos, aunque valido, no es recomendable hacer esto. Es preferible usar el ON para
unir campos de varias tablas y emplear el WHERE para filtrar los resultados (añado alias):

```
SELECT City.Name AS Ciudad, Country.Name AS País, City.Population AS Población
    FROM City JOIN Country
    ON CountryCode = Code
    WHERE City.Name = 'León'
    AND City.Population > 1000000;
```

También podemos añadir expresiones LIKE

```
SELECT City.Name AS Ciudad, Country.Name AS País, Region,City.
    Population AS Población
    FROM City JOIN Country
    ON CountryCode = Code
    WHERE City.Name = 'León'
    AND Region LIKE '%America';
```

SELECT COUNT

Imaginemos que nuestra tabla "libros_act" contiene muchos registros. Para averiguar la cantidad sin necesidad de contarlos manualmente usamos la función count().

```
select count(*) from libros_act;
```

Para saber la cantidad de libros de la editorial "Planeta"

```
select count(*) from libros_act
        where editorial='Planeta';
```

Podemos usar la clausula "where" para una consulta más específica.

```
select count(*) from libros_act
        where  autor like '%er%';
```

Para contar los registros que tienen precio (sin tener en cuenta los que tienen valor nulo), usamos la función "count()" y en los paréntesis colocamos el nombre del campo que necesitamos contar.

```
select count(precio)from libros;
```

También es posible «contar los distintos» valores de un campo de una tabla.

¿Cuántas editoriales distintas existen en la tabla libros_act?

```
select count(distinct editorial) from
        libros_act;
```

SELECT MAX-MIN

Para averiguar el valor máximo o mínimo de un campo usamos "max()" y "min()".

Queremos saber cuál es el mayor precio de todos los libros:

```
select max(precio) from libros_act;
```

Queremos saber cuál es el valor mínimo de los libros de "Cervantes", es decir,el libro más barato:

```
select min(precio) from libros_act where
        autor like '%Cervantes%';
```

SELECT SUM

La función "sum()" retorna la suma de los valores que contiene el campo especificado. Por ejemplo, queremos saber la cantidad de libros que tenemos disponibles para la venta.

```
select sum(cantidad) from libros;
```

También podemos combinarla con “where". Por ejemplo, queremos saber cuántos libros tenemos de la editorial Planeta.

```
select sum(cantidad) from libros where editorial ='Planeta';
```

SELECT AVG

La función avg() retorna el valor promedio de los valores del campo especificado. Por ejemplo, queremos saber el promedio del precio de los libros de Cervantes:

```
select avg(precio) from libros_ant where autor like '%Cervantes%';
```

SELECT…GROUP BY

Hemos visto como contar registros, calcular sumas y promedios, obtener valores máximos y mínimos.

Dijimos que operamos sobre conjuntos de registros, no con datos individuales.

Generalmente esta funciones se combinan con la sentencia "group by", que agrupa registros para consultas detalladas.

Si queremos saber cuantos libros de la tabla son de cada autor, tendríamos que:

```
select count(*) from libros where autor='Cervantes';
```

Y repetirlo con cada valor de autor:

```
select count(*) from libros_ant where autor=‘Pedro';

select count(*) from libros_ant where autor=‘Antonio';
```

Pero hay otra manera de hacer lo anterior, utilizando la cláusula “group by":

```
select autor, count(*) as cantidad from libros group by autor;
```

Además de crear un alias para el campo que nos muestra la cuenta (cantidad), nos permite mostrarlo junto con el campo autor.

Para mostrar la suma total de libros por cada autor

```
SELECT autor, sum(cantidad) from libros
        group by autor;
```

SELECT…HAVING

Así como la cláusula "where" permite seleccionar (o rechazar) registros individuales; la cláusula “having" permite seleccionar (o rechazar) un grupo de registros.

La cláusula HAVING, por tanto, necesita ir acompañada de la clausula GROUP BY (que es la que agrupa un cierto número de registros).

Partimos de la tabla Recaudacion que tiene las siguientes tuplas:

id_Recaudacion	Comercial	BeneficioRecaudado
1	Juan	100.000
2	Lucas	125.000
3	Ana	75.000
4	Jose	150.000
5	Juan	80.000
6	Lucas	80.000

Queremos crear una consulta que nos devuelva el nombre de los comerciales que han recaudado un beneficio superior a los 175.000 €.
La consulta en **MySQL** es la siguiente:

```
SELECT Comercial, SUM(BeneficioRecaudado) FROM Recaudacion
GROUPBY Comercial HAVING SUM(BeneficioRecaudado)>175.000;
```

Con esta consulta SQL estamos realizando las siguientes acciones:

- Agrupamos los datos resultanes en funcion del campo Comercial.
- Sumamos los beneficios recaudados de cada Comercial.
- Comprueba que esa suma sea mayor que la condición de la cláusula HAVING, que sea mayor de 175.000.

Esta Consulta nos arroja el siguiente resultado:

Comercial	SUM(BeneficioRecaudado
Juan	180.000
Lucas	205.000

Si queremos saber la cantidad de libros agrupados por editorial usamos la siguiente instrucción que ya vimos:

```
select editorial, count(*) from libros
        group by editorial;
```

Si queremos saber la cantidad de libros agrupados por editorial pero considerando sólo algunos grupos, por ejemplo, los que devuelvan un valor mayor a 2, usamos la siguiente instrucción:

```
select editorial, count(*) from libros
        group by editorial having count(*)>2;
```

Otro ejemplo: Promedio de los precios de los libros agrupados por editorial

```
Select editorial, avg(precio) from libros
        group by editorial;
```

Ahora, sólo queremos aquellos cuyo promedio supere los 25 euros:

```
Select editorial, avg(precio) from libros
        group by editorial having avg(precio)>25;
```

CONSULTAS DE ACCION

Este tipo de consultas modifica el contenido de las tablas. Las más frecuentes que podemos necesitar emplear son:

Agregación de campos de tabla

Eliminación de registros

Actualización de registros

Agregación de campos de otra tabla - INSERT SELECT

```
insert into TABLA_DESTINO (campo1, campo2, …) select campo1, campo2
        from TABLA_ORIGEN where CONDICIÓN;
```

Es condición esencial que el número de campos de la SELECT, coincida en NÚMERO y TIPO con los campos del INSERT.

En la BD censo_para_consultas QUEREMOS PASAR DE LA TABLA PERSONA, A LA TABLA PERSONA_AC, TODOS LOS REGISTROS CUYO PA ES < 5000

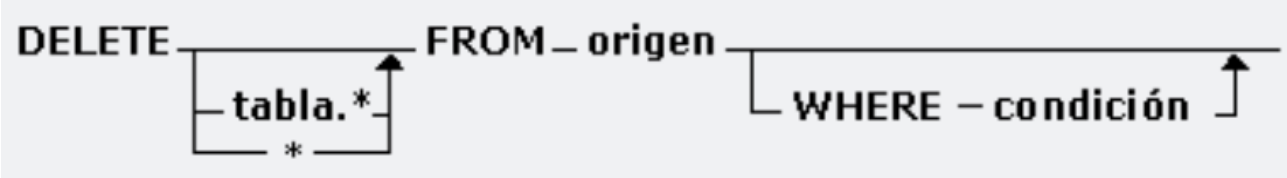
```
INSERT INTO persona_ac (Id_Municipio, dni, Nombre, Apellido, PA, FechaAlta)
        SELECT Id_Municipio, dni, Nombre, Apellido, PA, FechaAlta
        FROM persona WHERE PA < 5000
```

OTRA OPCIÓN, ES COMPLETAR ALGUNOS CAMPOS CON VALORES, Y OTROS CON LO QUE DEVUELVE LA SELECT.

```
INSERT INTO persona_ac (Id_Municipio, Nombre, PA)
    SELECT Id_Municipio, 'Augusto', 100000
    FROM persona WHERE nombre='Augusto'
```

ELIMINACION DE REGISTROS: DELETE

La sentencia DELETE elimina filas de una tabla.



Para no borrar tablas con registros y realizar pruebas, es recomendable realizar una copia a través de la interfaz de phpmyadmin.

Dentro de la base de datos —> MenúOperaciones —> Copiar la tabla a (base de datos.tabla) —> tabla nombre_tabla_copia

Haz una copia de la tabla personas_ac

Borrado de todas las filas de la tabla:

```
DELETE FROM persona_ac_copia;
```

Borrado con condicion /es:

```
DELETE FROM persona_ac_copia WHERE PA< 5000 or dni='';
```

Otros ejemplos

```
DELETE FROM libros_act_copia WHERE autor IN (SELECT nombre_autor
    FROM autores WHERE fec_nacimiento < '2000-01-01');

DELETE a.* FROM libros_act_copia a INNER JOIN libros_ant b ON a.autor = b.autor
    WHERE a.cantidad > 110;
```

BORRADO CON «TRUNCATE»

La sentencia **"truncate table"** vacía la tabla (elimina todos los registros) y vuelve a crear la tabla con la misma estructura.

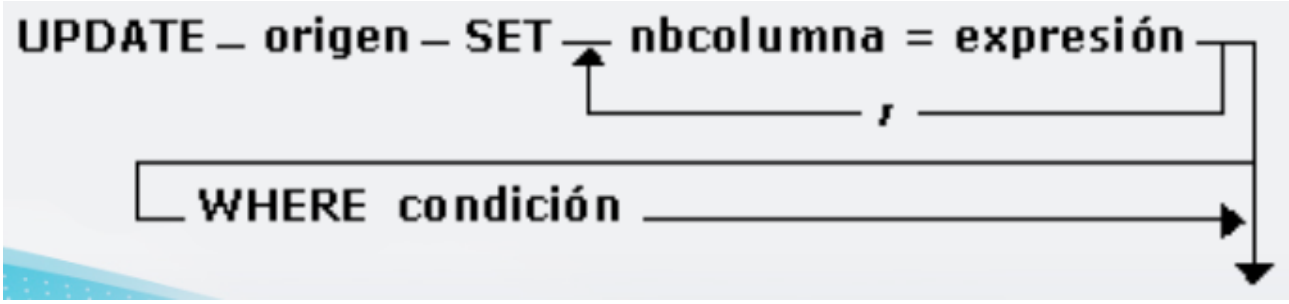
La sentencia "drop table" borra la tabla.

Sintaxis: *TRUNCATE TABLE nombre_tabla;*

ACTUALIZACIÓN DE REGISTROS: UPDATE

La sentencia UPDATE modifica los valores de una o más columnas en las filas seleccionadas de una o varias tablas.

La sintaxis es la siguiente:



Origen puede ser un nombre de tabla, una composición de tablas.

La cláusula SET especifica qué columnas van a modificarse y qué valores asignar a esas columnas.

nbcolumna, es el nombre de la columna a la cual queremos asignar un nuevo valor por lo que debe ser una columna de la tabla origen.

```
UPDATE persona_ac_copia set pa=pa+8

UPDATE persona_ac_copia SET pa=pa*1.21
        WHERE apellido LIKE '%r'
```

Ejemplos

```
UPDATE libros_act INNER JOIN autores
        ON libros_act.autor = autores.nombre_autor
        SET cantidad=cantidad+100

UPDATE libros_act SET cantidad = 0
        WHERE autor IN (SELECT nombre_autor
        FROM autores WHERE pais_nac = 'Australia');
```

SUBCONSULTAS

Una subconsulta es una instrucción SELECT anidada dentro de una instrucción SELECT, SELECT...INTO, INSERT…INTO, DELETE, o UPDATE o dentro de otra subconsulta. Puede utilizar diferentes formas de sintaxis para crear una subconsulta:

1- Una subconsulta es una instrucción SELECT anidada dentro de una instrucción SELECT, SELECT...INTO, INSERT…INTO, DELETE, o UPDATE o dentro de otra subconsulta.

```
SELECT * FROM Productos WHERE IDProducto IN (
    SELECT IDProducto FROM DetallePedido WHERE Descuento = 0.25)
```

2- En una subconsulta, se utiliza una instrucción SELECT para proporcionar un conjunto de uno o más valores especificados para evaluar en la expresión de la cláusula WHERE o HAVING, es decir, recuperar criterios.

```
SELECT Apellido, Nombre, Titulo, Salario FROM Empleados AS T1
    WHERE Salario = (SELECT Avg(Salario) FROM Empleados
    WHERE T1.Titulo = Empleados.Titulo)
    ORDER BY Titulo
```

VISTAS: View

Las vistas pueden considerarse como tablas virtuales en un SGBD relacional.

Una tabla tiene un conjunto de definiciones, y almacena datos físicamente.

Una vista tiene un conjunto de definiciones, que se construye en la parte superior de la(s) tabla(s) u otra(s) vista(s), y no almacena datos físicamente.

Se emplean para poder recuperar los resultados de las consultas sin tener que volver a plantearla.

La sintaxis para la creación de una vista es la siguiente:

```
CREATE VIEW "NOMBRE_VISTA" AS "Instrucción SQL";
```

El nombre de la vista, **no puede ser el mismo que el nombre de una tabla.**

Por ejemplo; Supongamos que tenemos la siguiente tabla:

Tabla ***Customer***

Nombre de Columna	Tipo de Datos
First_Name	char(50)
Last_Name	char(50)
Address	char(50)
City	char(50)
Country	char(25)
Birth_Date	datetime

Queremos crear una vista denominada V_Customer que contiene sólo las columnas First_Name, Last_Name y País de esta tabla, ingresaríamos:

```
CREATE VIEW V_Customer AS
    SELECT First_Name, Last_Name, Country FROM Customer;
```

Ahora tenemos una vista llamada **V_Customer** con la siguiente estructura:

View ***V_Customer***

Nombre de Columna	Tip de Datos
First_Name	char(50)
Last_Name	char(50)
Country	char(25)

Podemos utilizar también una vista para aplicaruniones a dos tablas. En este caso, los usuarios sólo ven una vista en vez de dos tablas, y la instrucción SQL que los usuarios necesitan emitir se vuelve mucho más simple. Digamos que tenemos las siguientes dos tablas:

Tabla ***Geography***

Region_Name	Store_Name
East	Boston
East	New York
West	Los Angeles
West	San Diego

Tabla ***Store_Information***

Store_Name	Sales	Txn_Date
Los Angeles	1500	05-Jan-1999
San Diego	250	07-Jan-1999
Los Angeles	300	08-Jan-1999
Boston	700	08-Jan-1999

Queremos construir una vista que tenga ventas organizadas según la región:

```
CREATE VIEW V_REGION_SALES AS
    SELECT A1.Region_Name AS REGION,
    SUM(A2.Sales) AS SALES
    FROM Geography A1, Store_Information AS A2
    WHERE A1.Store_Name = A2.Store_Name
    GROUP BY A1.Region_Name;
```

Esto nos brinda una vista, ***V_REGION_SALES***, que se ha definido para las ventas de los negocios según los registros de la región. Si deseamos saber el contenido de esta vista, tecleariamos,

```
SELECT * FROM V_REGION_SALES;
```

Resultado:

REGION	SALES
East	700
West	2050

MOSTRAR VISTAS: SHOW

SHOW CREATE VIEW [nombre de la vista]

SELECT table_name from information_schema.tables where table_type='VIEW'

MOSTRAR VISTAS: ALTER VIEW

```
ALTER VIEW miprimeravista AS SELECT *
    FROM autores WHERE fec_nacimiento>'1950/05/05'
```

BORRAR VISTAS: DROP VIEW

drop view [nombre de la vista];