**SQL**

**MySQL:** Es un sistema de manejo de bases de datos que utiliza SQL como lenguaje de distinción de datos (LLD) y como lenguaje de manipulación de datos (LMD)

**Para empezar, vamos a D:\MySQL\mysql-9.0.1-winx64\bin (la carpeta bin del programa) y en la barra para explorar archivos escribimos cmd**

**Para iniciar el servidor desde la consola:**

**mysqld**

o

**mysqld --console**

(este es para q nos diga si aparecen mensajes de error en la misma ventana de comandos)

El mensaje que tendría que aparececr al final es “… ready for connections..”

Para detener el servidor desde la consola:

**mysqladmin -u root shutdown**

**Teoria**: El servidor de MySQL sigue el modelo Cliente-Servidor, esto quiere decir que el servidor una vez que esta corriendo va a escuchar por conecciones en un numero de IP y un numero de puerto y los clientes que pueden ser programas se conectan a través de la red con el servidor y hacer consultar (queries) u otras operaciones.

Nosotros lo haremos todo a través del localhost pero también se podría hacer a través de la red.

Para permitir que los programas clientes se conecten al servidor MySQL permite definir distintos usuarios con diferentes privilegios

Mientras tengamos la ventana abierta estará corriendo el servidor.

**Para entrar como cliente** vamos a la carpeta bin escribimos en la barra de exploración de carpetas:

mysql -u root

Esto ejecuta al cliente y se conecta al servidor utilizando el usuario root

Si funciona todo bien veremos el prompt mysql y la versión del servidor, desde aca podemos ejecutar consultas:

**show databases;**

Este comando **nos** **muestra** las **bases** de **datos** que están **creadas** en el **servidor**, la primera vez solo mostrara las bases de datos propias de SQL que almacenan los datos de los usuarios y configuraciones del servidor.

Para setear la contraseña:

Set password for root@localhost=’bd-dcic’;

**Siempre que ejecutamos un comando desde el cliente se finaliza con “;”**

Ahora como tenemos contra tenemos q ingresar asi:

**mysql -u root -p**

Luego la contra:

bd-dcic

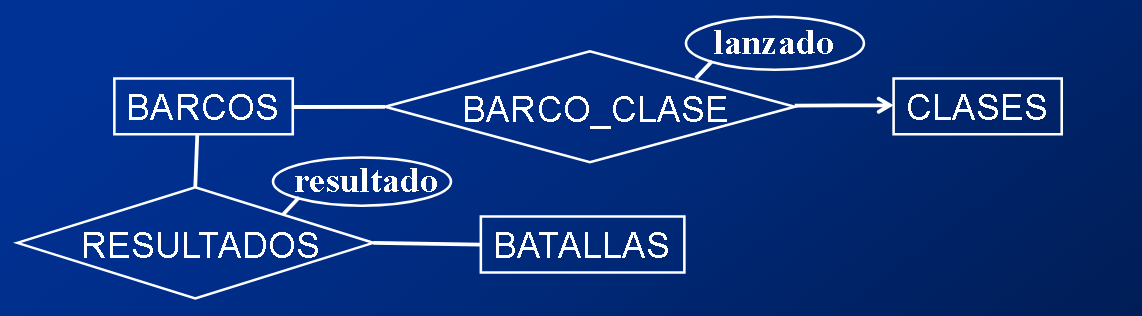
**Crear una base de datos**

Para crear una base de datos utilizando el cliente mysql hay dos opciones:

1. **Versión más practica**: Poner una secuencia de sentencias SQL en un archivo de texto, y ejecutar toda la secuencia de una vez. (Batch Mode - sección 3.5 del manual)
2. Ingresar las sentencias SQL una a una desde la línea de comandos, es decir, crear la base de datos y luego crear tabla por tabla. (ver sección 3.3 del manual).

Ejemplo:

Consideremos una B.D. sobre barcos y batallas:



BARCOS (nombre barco, id, capitan)

CLASES (clase, tipo, pais, nro caniones, calibre, desplazamiento)

BARCO CLASE (nombre barco, clase, lanzado)

BATALLAS (nombre batalla, fecha)

RESULTADOS (nombre barco, nombre batalla, resultado)

Para crear esta base de datos utilizaremos la opción de batch mode.

• Creamos un archivo de texto (batallas.sql) con la secuencia de sentencias necesarias para crear la base de datos.

• Luego desde el cliente ejecutamos toda la ruta del archivo (la parte de “mysql>” es el promt que aparece en el cmd):

mysql> **\. batallas.sql**

El profe para hacerlo más fácil puso el archivo en la carpeta bin de MyQSL entonces ya está en la carpeta donde se abrió el cmd.

Este último comando lo que hace es que tome como dato de entrada el archivo y ejecute en secuencia todas las sentencias SQL que hay dentro del archivo

Nota**: si una de las sentencias produce un error**, es posible que alguna de las otras sentencias de la secuencia se ejecute con éxito. Antes de ejecutar el archivo nuevamente**, borrar la base de datos desde el cliente** con:

**mysql> drop database batallas;**

• El comando drop database no elimina los usuarios. Si la secuencia de sentencias crea algún usuario, es necesario eliminarlos mediante:

**mysql> drop user nombre\_usuario@host;**

**Archivos SQL**

#Archivo batch (batalllas.sql) para la creación de la Base de datos del práctico de SQL

#Lo que esta después del "#" es un comentario

# Creo de la Base de Datos

**CREATE** **DATABASE** batallas;

# selecciono la base de datos sobre la cual voy a hacer modificaciones

**USE** batallas;

#-------------------------------------------------------------------------

# Creación Tablas para las entidades

**CREATE** **TABLE** barcos ( #creamos la tabla barcos

nombre\_barco VARCHAR(45) NOT NULL**1**,

capitan VARCHAR**2**(45) NOT NULL,

id INT UNSIGNED NOT NULL AUTO\_INCREMENT**3**,

**CONSTRAINT4** pk\_barcos

**PRIMARY** **KEY5** (nombre\_barco),

**KEY** (id)

) ENGINE=InnoDB**6**;

El tipo VARCHAR es una cadena de caracteres de longitud que varía y que tiene como máximo 45 caracteres.

**1.**La restricción **NOT NULL** indica que no puede tener valores nulos.

**2.INT** **UNSIGNED** quiere decir que será entero y será no signado, o sea solo puede tomar valores positivos

**3.AUTO\_INCREMENT** es para cuando en la tabla barcos insertamos una tupla si no especificamos el valor del id le asigna automáticamente un numero siguiente al número que fue ingresado. De esta forma manejamos automáticamente lo que son los identificadores a los que les asignamos valores de forma creciente.

**4 y 5.** Para la llave primaria **PRIMARY KEY** opcionalmente se puede poner la palabra **CONSTRAIT** y un nombre, esto nos sirve para cuando por ejemplo insertamos un barco y ese nombre de barco ya existía, entonces como no puede haber 2 nombres de barcos iguales, nos va a dar un error en la PRIMARY KEY y nos retorna lo que escribimos luego del CONSTRAIT, en este caso seria “pk\_barcos”. Esto nos sirve de referencia para saber cuál fue la PRYMARY KEY que fallo.

**6**. **ENGINE=InnoDB** acá definimos de que tipo va a ser la tabla de MySQL, se puede no poner nada, pero este tipo de tablas son las más completas y son las que utilizaremos.

Para la definición de esquemas:

CREATE TABLE nombre\_tabla (

atrib1 <tipo> [<restricción>],

atrib2 <tipo> [<restricción>],

....

atribN <tipo> [<restricción>],

<restriccion\_integridad\_1>,

....

<restriccion\_integridad\_k>);

Las restricciones de integridad son las llaves primarias y las llaves foráneas.

Podemos:

Definir el esquema de la tabla (atributos y sus tipos).

Definir, si existen, restricciones sobre los atributos.

Un valor no puede ser nulo (not null)

Restricciónes de valores (Ej. Unsigned, Unique), subconjunto válido, etc.

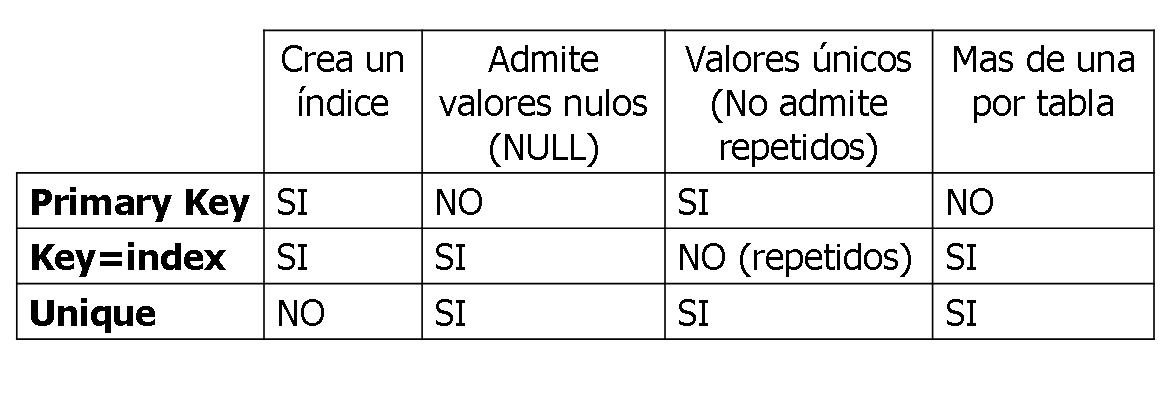
Definir restricciones a nivel de tabla

Clave primaria: primary key ( A1, ..., An )

Clave foránea: foreign key ( A1, ..., An ) references R( B1, ..., Bn )

Índices: key( A1, ..., An ), index ( A1, ..., An)

**Similitudes y diferencias entre primary key, key y unique**



Primary Key sobre un conjunto de atributos/campos crea un índice para esos atributos, esto permite que las consultas sobre estos atributos se hagan mas rápido. A su vez Primary key no admite valores nulos, tampoco admite valores repetidos y no puede haber mas de una por tabla.

Cuando escribimos:

CREATE TABLE transferencia (

…

CONSTRAINT fk\_transferencia

FOREIGN KEY (nro\_cliente) REFERENCES cliente\_ca (nro\_cliente),

**ON DELETE RESTRICT ON UPDATE CASCADE**

**Esto quiere decir que en la eliminación lo restringa y en la actualización lo haga en cascada, esto quiere decir lo siguiente, como estamos referenciando a otra tabla, en este caso a la tabla cliente\_ca, si en algún momento eliminamos una tupla de la tabla cliente\_ca que está siendo referenciada por la clase en la que estamos (transferencia en este caso) nos quedaría una referencia colgada. Pero esto lo impedimos con la última línea de código “ON DELETE RESTRIC...”, al intentar borrar un cliente nos va a dar un error y nos va a decir que no se puede borrar porque está violando una restricción de llave foránea. En el caso de la actualización ocurre cuando le cambiamos el nro\_cliente al poner que se hace en cascada “ON UPDATE CASCADE” actualiza todas las referencias que había con nro\_cliente en todas las tablas en las que aparecía con el nuevo número.**

También está la restricción:

**ON DETELE RESTRICT ON UPDATE RESTRICT**

**Esto lo que hace es no permitirnos cambiar el número del cliente.**

También esta y es la más peligrosa:

CREATE TABLE resultados (

…

FOREING KEY (nombre\_batalla) REFERENCES batallas (nombre\_batalla)

**ON DELETE CASCADE ON UPDATE CASCADE**

**)**

**Al borrar le pedimos que lo haga en cascada, esto es, si borramos una batalla de la tabla batallas borrara todas las tuplas de resultados que hagan referencia a esa batalla.**

**No es muy común el siguiente tipo de tablas:**

Si queremos **modificar** la **estructura** de una **tabla** utilizaremos la sentencia:

**ALTER TABLE <nombre\_tabla>**

**DROP columna** #elimina una columna que no queramos mas

**MODIFY columna <modificación>** #modificar una columna, ejemplo: permitir que ahora

Se puedan agregar valores nulos o enteros,etc.

**ADD columna <tipo> [<restriccion1>,..]** #agregar columna con el tipo que queramos

**ADD/DROP CONSTRAINT restriccion1…** #eliminar y/o agregar CONSTRAINT

**Borrado de tablas desde el cmd como cliente:**

La instrucción **DROP TABLE <tabla>** permite borrar el contenido y el esquema de una tabla.

Para poder borrar una tabla la misma **no debe estar referenciada por otra**.

De la misma forma, para poder borrar una columna con ALTER TABLE, esta no debe estar referenciada por otra tabla.

**Lenguaje de Manipulación**

**de Datos (DML)**

**Almacenar (insertar) datos.**

**Consultar datos almacenados.**

**Modificar el contenido de los datos almacenados.**

**Actualizando el contenido de la Base de Datos**

**Agregando datos a una tabla con INSERT**

Existen varias formas de agregar datos a una tabla. Sea R representa una relación con atributos A1, ..., An y Vi representa un valor

1. Sin especificar los atributos:

**INSERT INTO R VALUES (V1, ...,Vn)**

1. Especificando todos los atributos

**INSERT INTO R(A1, ...,An) VALUES (V1, ...,Vn)**

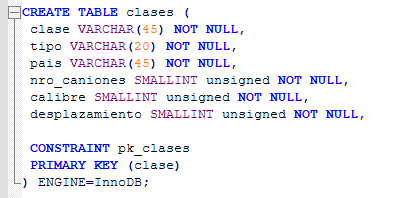
Al atributo A1 se le asignara el valor V1 y asi para todos los atributos y valores.

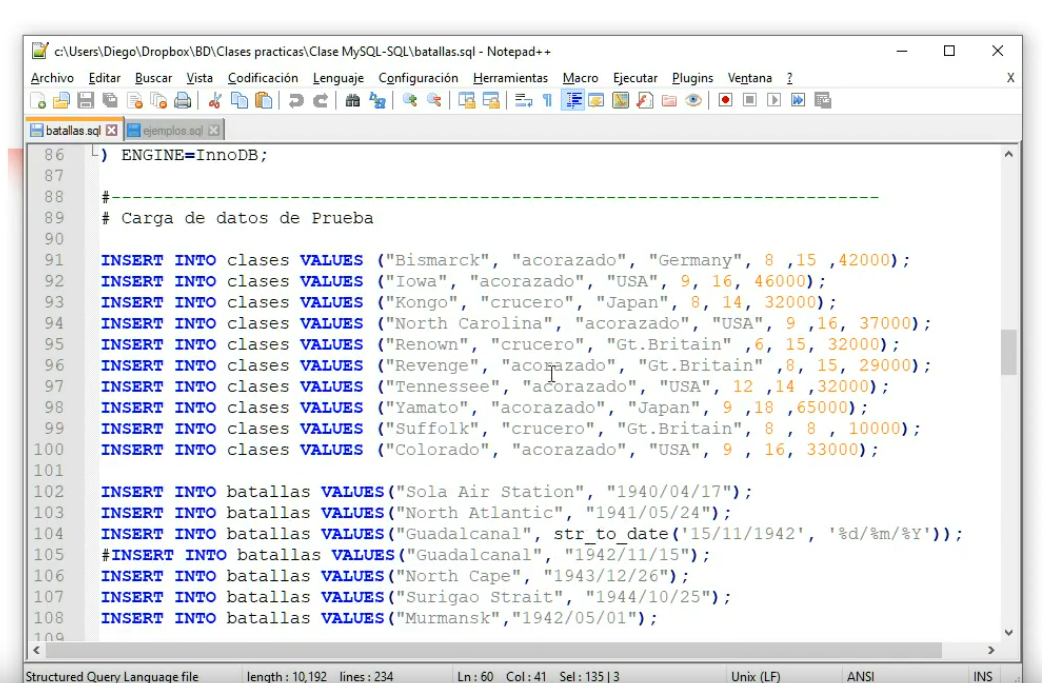
1. Especificando algunos atributos en cualquier orden. Los demás atributos toman el valor por defecto de su tipo:

**INSERT INTO R(A1, ...,Ak) VALUES (V1, ...,Vk)**

**Ejemplo del 1:**

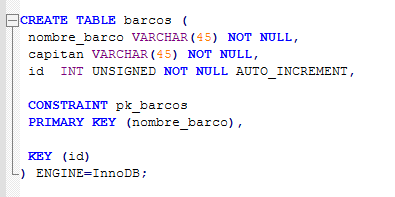
**Sea la clase clases la siguiente:**

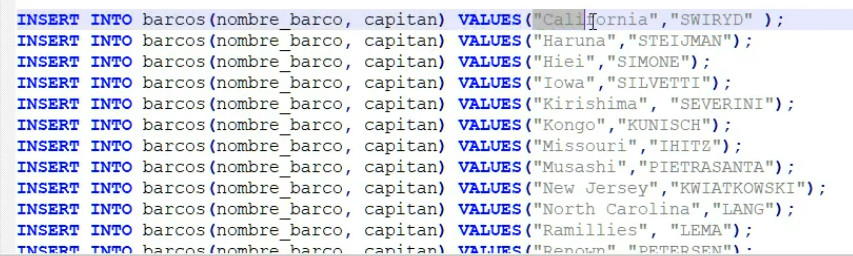
****

****

**Ejemplo del 2:**

**Sea la clase barcos:**

****

****

**Borrar filas de una tabla**

Borrar todas las tuplas de una tabla/relación R:

**DELETE FROM R;**

**Ejemplo**: DELETE FROM barcos; esto borraría todas las tuplas de la tabla barcos.

Para borrar algunas tuplas que satisfacen cierta condición:

**DELETE FROM R WHERE <condición>;**

**Ejemplo**: DELETE FROM batallas WHERE nombre\_batalla="North Cape";

Borra de las batallas las que tengan como nombre “North cape”

Observaciones sobre DELETE

La operación DELETE sin cláusula de condición borra todas las filas de una tabla, no borra su esquema. O sea, la tabla va a seguir existiendo, pero no tendrá nada en su contenido.

La operación de DELETE será exitosa siempre que no se violen las restricciones preexistentes (por ej. de llaves foráneas).

**Actualizar filas de una tabla**

Actualizar todas las tuplas de una tabla:

**UPDATE R SET <nuevas asignaciones>;**

**Ejemplo:** UPDATE barcos SET capitan = “SIMONE”; Esto actualiza todas las tuplas de barcos y setear al campo de capitán con SIMONE

Para actualizar tuplas que cumplan con una condición:

**UPDATE R SET <nuevas asignaciones>; WHERE <condición>;**

**Ejemplo:** UPDATE resultados SET resultado= “hundido”

WHERE nombre\_batalla=“Guadalcanal”;

De la tabla resultados aquellas tuplas donde el nombre de batalla es Guadalcanal el resultado lo va a cambiar a hundido.

Al igual que en el caso de INSERT y DELETE esta operación solo será exitosa si como resultado de la modificación se siguen respetando todas las restricciones preexistentes.

**Recuperando el contenido de la Base de Datos (consultas)**

La estructura básica consiste de tres cláusulas: SELECT, FROM y WHERE.

**SELECT**: Muestra los atributos deseados. Proyección del álgebra relacional.

**FROM**: Especificamos una o más tablas de las cuales queremos recuperar datos. Producto cartesiano del álgebra relacional.

**WHERE**: Especificamos la condición que tienen que cumplir las tuplas para aparecer en el resultado. Selección del álgebra relacional.

Una consulta típica SQL es de la forma:

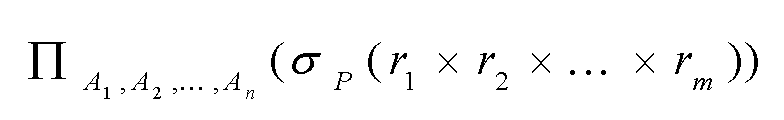
Select A1, A2, ..., An

From r1, r2, ..., rm

[ where P]

* Ai representa un atributo
* ri representa una relación
* P es un predicado sobre los atributos de A1, A2, ..., Am

Esta consulta se expresa y resuelve como en Algebra Relacional:

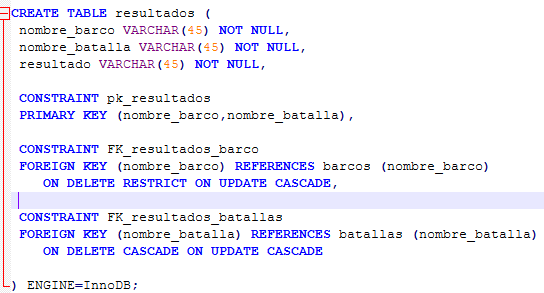


Proyectar **todos** los atributos del resultado:

**SELECT \* FROM …;**

**Esto recupera todos los atributos de las tablas que mencionemos**

**Ejemplo:** Sea la clase resultados la siguiente:



La consulta:

SELECT \* FROM resultados; Nos mostrara el nombre del barco, el resultado y la fecha

**Proyectar = Mostrar**

Proyectar algunos atributos del resultado: listar los atributos separados por coma

**SELECT A1, A2, …, An FROM …;**

**Ejemplo:** SELECT nombre\_barco, nombre\_batalla FROM resultados;

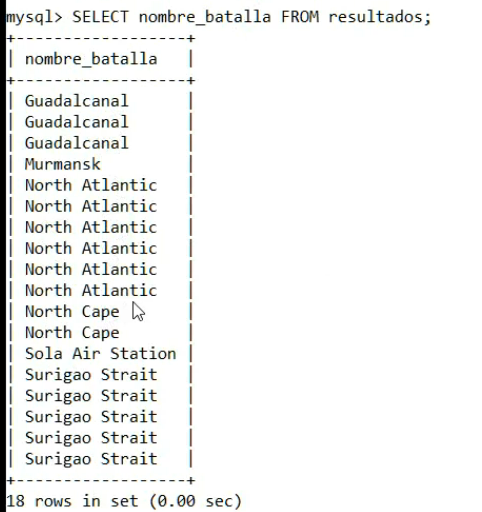
****

Las cláusulas **SELECT, FROM, WHERE** utilizan semántica Bag, es decir, que **no** se **eliminan** las **tuplas** **duplicadas**.

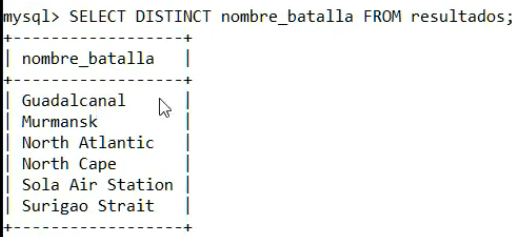
Para utilizar una semántica de conjunto (Set) es necesario utilizar la cláusula **DISTINCT** **luego** **del** **SELECT**. Que **elimina** las **tuplas** **repetidas**.

**SELECT DISTINCT A1, A2, ..., An FROM …**

**Ejemplo** SELECT sin DISTINCT:



**Ejemplo** SELECT con DISTINCT:



Para especificar que tuplas queremos devolver en el resultado utilizaremos **WHERE**.

Las expresiones se arman con los operadores de comparación >, <, =, <>, >=, <= y los operadores lógicos AND, OR y NOT.

Los valores que se comparan pueden incluir atributos de las relaciones mencionadas en el FROM y/o constantes.

Pueden usarse operadores aritméticos (+, -, etc) siempre que se respeten los tipos de los datos a comparar.

**Ejemplo**:

**SELECT** nombre\_barco, nombre\_batalla

**FROM** resultados

**WHERE** resultado="hundido";

Solo devolverá los nombres de barcos y de batallas cuyos resultados tengan “hundido”

-------------------------------

**SELECT** nombre\_barco

**FROM** resultados

**WHERE** resultado="hundido" **AND** nombre\_batalla="North Atlantic";

Solo devolverá los nombres de barcos cuyos resultados tengan “hundido” y el nombre de batalla sea “North Atlantic”.

**Comparación de Strings**

Pueden realizarse comparaciones con operadores relacionales <, >, >=, <=, =, <> según el orden lexicográfico.

También comparaciones de patrones:

**<Atributo> LIKE <patrón>**

**<Atributo> NOT LIKE <patrón>**

El patrón es un string que puede contener dos caracteres especiales **%** y **\_**

**Comparación con LIKE**

s LIKE p [ESCAPE ‘<caracter>’]

Los caracteres distintos dé % y \_ deben corresponderse con ellos mismos.

El caracter % (en s) puede corresponderse con cualquier cadena de 0 o más caracteres (en p).

El caracter \_ (en s) se corresponde con un único caracter cualquiera (en p)

**Ejemplos**:

SELECT \*

FROM batallas

WHERE fecha > "1942-01-01";

--------------------

**SELECT nombre\_barco**

**FROM barcos**

**WHERE nombre\_barco like 'Re%';**

**Listar todos los barcos donde el campo nombre\_barco empieza con los caracteres Re**

Captura de como funciona:

****

--------------------

SELECT nombre\_barco

FROM barcos

WHERE nombre\_barco > 'R%';

**Listar todos los barcos donde el campo nombre\_barco empieza con los caracteres R y siga con cualquier cosa.**

**Multiples relaciones**

Si después de la palabra reservada **FROM** se enumera más de una relación, SQL hace el producto entre ellas.

Un producto cartesiano sin clausula **WHERE** generalmente da resultados sin significado semántico.

**Ejemplo**:

**SELECT \* FROM barcos, resultados;**

Esto hace el producto de ambas tablas, la tabla resultados tiene 18 filas y la de resultados tiene 33, al hacer el producto cartesiano combina cada fila de barcos con cada fila de resultados y eso nos da una tabla enorme de de 594 tuplas (18x33 tuplas).

**Ejemplo** más sensato:

**SELECT \* FROM barcos, resultados**

**WHERE barcos.nombre\_barco=resultados.nombre\_barco;**

Acá queremos vincular de las tablas barcos y resultados haciendo que coincidan los nombres de los barcos de la tabla barcos y los nombres de los barcos de la tabla resultados.

Esto se vería así:



**Ejemplo** utilizando and:

**SELECT barcos.nombre\_barco, capitan, nombre\_batalla, resultado FROM barcos, resultados**

**WHERE barcos.nombre\_barco=resultados.nombre\_barco AND resultado="hundido";**

**Importante: como el campo nombre\_barco se encuentra en 2 tablas hay que especificar de cual estamos hablando por ello se pone primero el nombre de la tabla y luego el campo. En el caso de “resultado “no se especifica nada porque ese campo solo existe en la tabla resultados.**

**Usuarios y Autorización**

La definición de usuarios y la asignación de privilegios nos permiten restringir el acceso sobre las bases de datos almacenadas en el servidor.

Simplifica el uso del sistema y mejora la seguridad ya que permite a los usuarios acceder solo a los datos que necesitan.

Por ejemplo, podemos definir un usuario con acceso sólo a ciertas tablas dentro de una o más bases de datos. Para cada tabla podemos definir el tipo de acceso (leer, insertar, modificar, etc).

Los **usuarios** se **identifican** con un **nombre** y un **dominio**:

**nombre@dominio**

El dominio identifica desde donde puede conectarse el usuario al servidor de MySQL.

**Ejemplos**:

* **root@localhost**, el **usuario** **root** **solo** puede **conectarse** desde la **PC** **donde** está **corriendo** el **servidor** **MySQL**.
* uns@**’%.uns.edu.ar’**, el usuario uns **solo** puede **conectarse** desde una **PC** del **dominio** **uns.edu.ar.**
* **admin**@’%’ (o admin, sin especificar dominio), el usuario admin **puede** **conectarse** **desde** **cualquier** **dominio**. Atencion!: se debe eliminar el usuario vacío (drop user ’’@localhost) para poder conectarse usando este usuario desde localhost.

**Ejemplo** creación de usuario y otorgamiento de privilegios esto va en el mismo archivo .SQL donde se escribieron las tablas:

# primero creo un usuario con CREATE USER

**CREATE USER** 'admin\_batallas'@'localhost’ **IDENTIFIED** BY 'pwadmin';

# el usuario admin\_batallas con password 'pwadmin' puede conectarse solo desde la #desde la computadora donde se encuentra el servidor de MySQL (localhost) luego le #**otorgo privilegios utilizando solo la sentencia GRANT**

**GRANT** **ALL** **PRIVILEGES** **ON** batallas. **\* TO** 'admin\_batallas'@'localhost' **WITH GRANT OPTION;**

# El usuario 'admin\_batallas' tiene acceso total a todas las tablas de la B.D. batallas y #puede crear nuevos usuarios y otorgar privilegios, primero creo un usuario con #CREATE USER

**CREATE USER** 'barco'@'localhost' **IDENTIFIED** **BY** 'pwbarco';

# el usuario 'barco' con password 'pwbarco' puede conectarse solo desde localhost, #luego le otorgo privilegios con **GRANT**

**GRANT** **SELECT** **ON** batallas.acorazados **TO** 'barco'@'localhost';

# el usuario 'barco' solo puede acceder a la tabla (vista) acorazados con permiso para #seleccionar

**GRANT** **ALL** **PRIVILEGES** **ON** batallas. **\* lo que hace es otorgar todos los privilegios sobre la bdd batallas**

'localhost' **WITH GRANT OPTION; permite al usuario crear nuevos usuarios y otorgarles privilegios sobre la bdd.**

**GRANT** **SELECT** **ON** batallas.acorazados **TO** 'barco'@'localhost'; permite al usuario solo hacer SELECT sobre la tabla acorazados de la bdd batallas, no va a poder insertar ni eliminar, solo podrá hacer SELECT.

**Usuarios y Autorización**

Estos usuarios, son usuarios del servidor de MySQL en general y definen niveles o perfiles de acceso al servidor.

**Generalmente estos usuarios serán utilizados por las aplicaciones** (no por personas) para acceder al Servidor.

Un caso particular seria por ejemplo un usuario admin, utilizado por el administrador (persona) de la B.D. para tareas de mantenimiento

Si un sistema requiere la definición de usuarios (personas), es conveniente definir estos usuarios dentro de la B.D. del sistema y manejar el acceso desde la/s aplicación/es utilizando usuarios definidos en el servidor.

**Usuarios y Vistas**

MySQL también permite definir vistas de la base de datos. (ver capítulo 19 del manual)

**Una vista es una tabla derivada (i.e. calculada en función de otras tablas o vistas) y persistente en la Base de Datos. Que sea persistente significa que se actualiza en base a los cambios que se realizan.**

Las vistas se crean a partir de una sentencia select que define su contenido.

Podemos restringir a los usuarios a acceder a una o más vistas de una o más bases de datos.

**Ejemplo** de creación de vistas en el mismo archivo .SQL donde se escribieron las tablas:

# Creación de vistas

# acorazados = Datos de todos los barcos que son "acorazados"

**CREATE VIEW** acorazados **AS**

**SELECT** b.nombre\_barco, b.capitan,

c.clase, c.pais, c.nro\_caniones, c.calibre, c. desplazamiento, b\_c.lanzado

**FROM** (barcos as b **JOIN** barco\_clase as b\_c **ON** b.nombre\_barco =

b\_c.nombre\_barco)

**JOIN** clases **as** c **ON** c.clase = b\_c.clase

**WHERE** c.tipo="acorazado";

**Se escribe** CREAT VIEW nombre\_deseado AS

SELECT contenido\_que\_deseemos\_tener\_en\_la\_lista

FROM esto\_se\_vera\_mas\_adelante

WHERE filtramos\_lo\_busquemos

**Esta lista contiene todos los datos de los barcos que son acorazados.**

**En resumen, creamos una tabla más en la bdd.**

Para visualizarla:  
