Base de datos relacionales y SQL

¿Qué es una Base de datos?

- Las aplicaciones no solo usan datos para procesar o tratarlos, también los guardan.
- Es importante contar con una organización en los datos que nos permita tratar con ellos.
- Es posible considerar una base de datos como una herramienta que nos ayuda a lograr este sentido.
- Los datos pueden tener muchas estructuras: imágenes, videos, datos en forma de tablas, etc. Para cada caso, existen bases de datos orientadas a guardarlos y tratar con ellos.

Un ejemplo de organización de datos: La Universidad

- Si imaginamos una aplicación que gestione una universidad, la aplicación deberá:
  - Gestionar alumnos , profesores, títulos o asignaturas
  - Introducir notas, modificar expedientes, etc.
- ¿Qué significa gestionar?
  - Operaciones CRUD
    - Create - Crear
    - Read - Leer
    - Update - Actualizar
    - Delete - Eliminar

¿Razón de ser de las bases de datos?

- Organizar la información.
- Tener un registro con toda la manipulación que se hace en el sistema.
- Otorga a las aplicaciones de una durabilidad, es decir, las aplicaciones con el tiempo irán desarrollándose y seguirán siendo usadas.
- Evitar incoherencia con los datos.
- Evitar duplicidad de datos.
- Evitar que la información sea incoherente.
- Control de acceso a la información: podemos saber qué usuario ha realizado qué acción sobre los datos.
- Realizar copias de seguridad.

Bases de datos

- Relacionales - SQL
  - Los datos guardados en la base de datos tienen una estructura y una relación entre sí.
  - **Usan el modelo ACID.**
- No (únicamente) relacionales - NoSQL
  - Tienen características de las SQL pero son aptas para cierto tipos de casos de uso.
  - **Usan el teorema CAP.**
  - Ejemplo: Piensa en imágenes y videos, no siguen una estructura de inicio y fin.
  - Ejemplo: Piensa en las cotizaciones de bolsa, siguen una serie temporal y este tipo de datos se trata de forma diferente.

¿Qué es ACID?

- ACID es un acrónimo en inglés de Atomicity, Consistency, Isolation and Durability:
- Es decir: Atomicidad, Consistencia, Aislamiento y Durabilidad, en español.
- Cuando operamos con las bases de datos relacionales se producen transacciones, es decir operaciones sobre las bases de datos: guardar un registro, editarlo, borrarlo, etc.
- A bajo nivel, una transacción está compuesta por varios procesos que se aplican uno después del otro. **La transacción debe realizarse de una sola vez** y sin que la estructura a medio manipular pueda ser alcanzada por el resto del sistema hasta que se hayan finalizado todos sus procesos.
    - Imagina borrar un registro 2 veces. Algo no va bien si pasa eso.

ACID

- **Atomicity ó Atomicidad:** Cuando una operación o transacción es atómica, tenemos la garantía de que se va a realizar al completo o no se va a realizar, pero no va a quedar incompleta. O “todo o nada”.
- **Consistency ó Consistencia:** Esta propiedad hace referencia a la Integridad de nuestra base de datos, una vez se realice la transacción la base de datos quedará en un estado consistente
- **Isolation ó Aislamiento:** Una operación será aislada de otras, es decir una operación o transacción no va a afectar a otras transacciones. Las transacciones sobre una misma información deben de ser independientes.
- **Durability ó Durabilidad:** Una vez haya terminado la transacción, los cambios perdurarán en el tiempo.

Teorema CAP

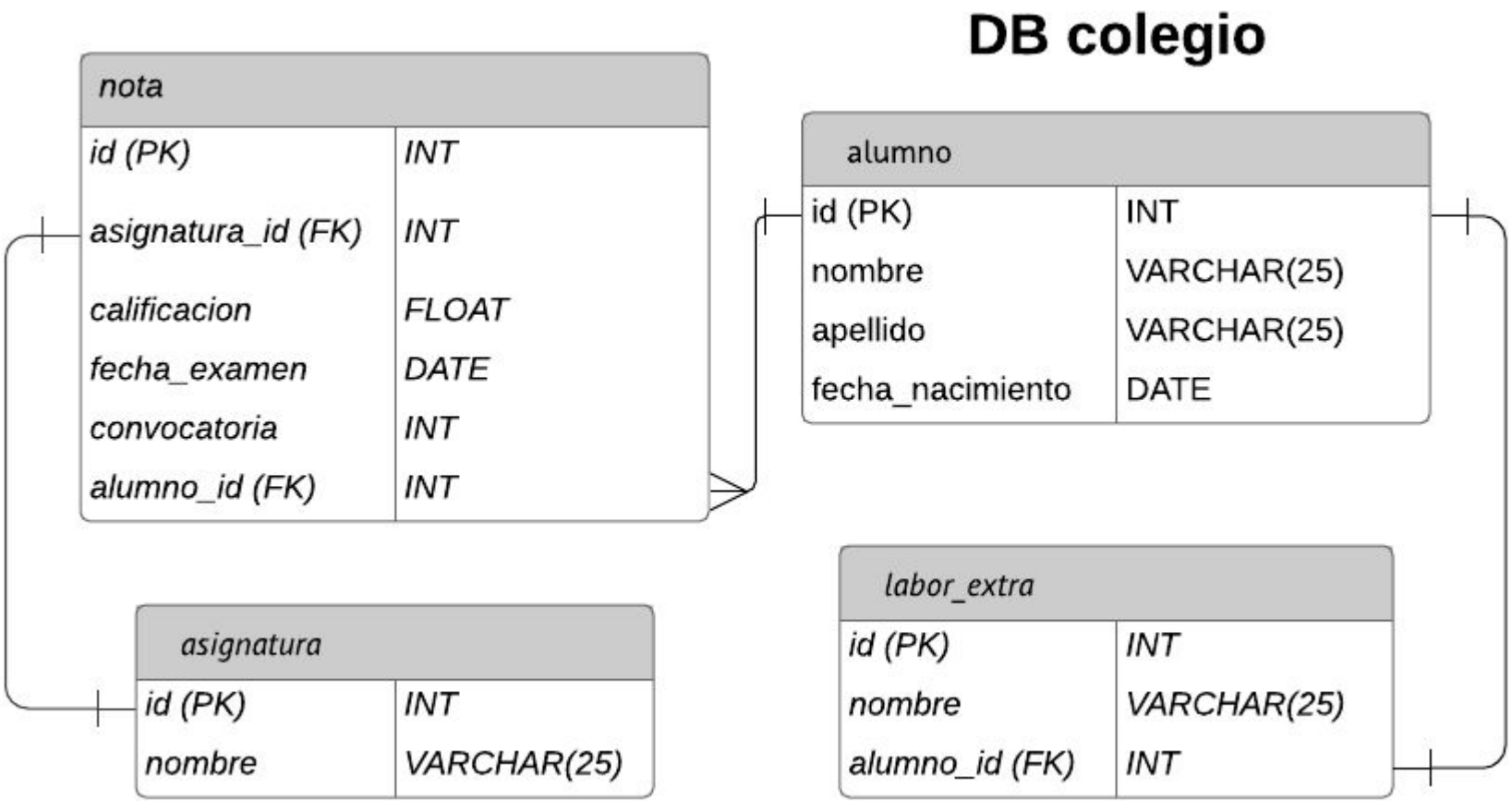
- **Consistencia:** Al realizar una lectura sobre una NoSQL va a devolver el resultado más reciente.
- **Disponibilidad:** Al realizar una lectura, va a recibir una respuesta correcta en un periodo de tiempo razonable, aunque puede no ser la más reciente.
- **Tolerancia a particiones:** Es posible tener datos repartidos entre distintas bases de datos y que el sistema siga funcionando.

El teorema CAP nos dice que un sistema de base de datos distribuido no puede garantizar estos tres puntos, **únicamente puede garantizarnos dos de ellos.**

¿Qué elementos forman las BD relacionales?

- **Datos**: Están organizados en tablas. En una tabla de base de datos que se parece a una simple hoja de cálculo.
- **Línea**: Es una fila (o tupla o registro) es un conjunto de datos relacionados, tales como datos de una dirección.
- **Columna**: Es un atributo de una tabla. Por ejemplo “color de pelo”.
- **Clave primaria**: La clave principal es única. Es una columna que identifica unívocamente a un registro (fila) en una tabla. Ejemplo: DNI.
- **Clave foránea:** La clave foránea se utiliza para enlazar dos tablas.
- **Clave compuesta:** Una clave compuesta son varias columnas que identifican de forma unívoca a una fila.

Veamos un ejemplo:


DB colegio
nota
id (PK) INT
asignatura_id (FK) INT
calificacion FLOAT
fecha_examen DATE
convocatoria INT
alumno_id (FK) INT
alumno
id (PK) INT
nombre VARCHAR(25)
apellido VARCHAR(25)
fecha_nacimiento DATE
asignatura
id (PK) INT
nombre VARCHAR(25)
labor_extra
id (PK) INT
nombre VARCHAR(25)
alumno_id (FK) INT

¿Qué es MySQL?

- MySQL es un sistema de gestión de base de datos relacional de Oracle Corporation.
- MySQL es de código abierto.
- MySQL soporta grandes bases de datos.
- Es posible usar el lenguaje de consultas SQL.
- MySQL permite en varios sistemas, y soporta múltiples idiomas. Estos lenguajes de programación, incluyendo C, C ++, Python, Java, Perl, PHP, Eiffel, Ruby y Tcl.

https://dev.mysql.com/downloads/installer/

¿Qué es MySQL Workbench?

- MySQL Workbench permite diseñar visualmente, modelar, generar y administrar bases de datos.
- El software nos va a proporcionar un conjunto de herramientas para mejorar el rendimiento de las aplicaciones MySQL.
- Aporta informes de rendimiento que nos van a proporcionar la fácil identificación y acceso a puntos de acceso, declaraciones SQL y más.
- Además, con un solo clic, es posible ver dónde optimizar sus consultas.

https://dev.mysql.com/downloads/workbench/

Welcome to MySQL Workbench

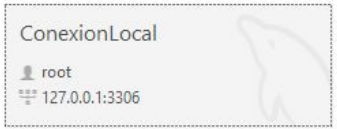
MySQL Workbench is the official graphical user interface (GUI) tool for MySQL. It allows you to design, create and browse your database schemas, work with database objects and insert data as well as design and run SQL queries to work with stored data. You can also migrate schemas and data from other database vendors to your MySQL database.

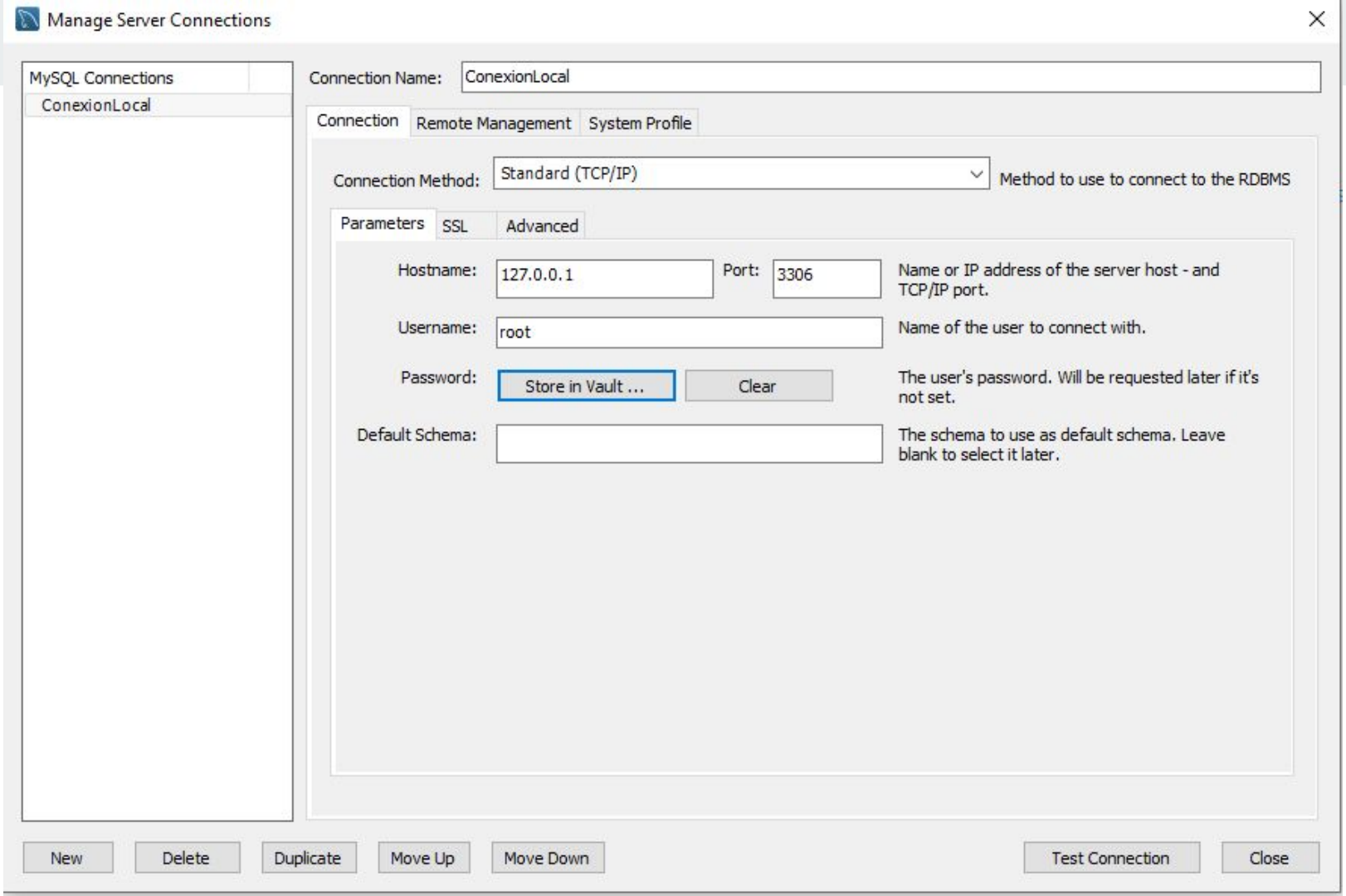
Browse Documentation >

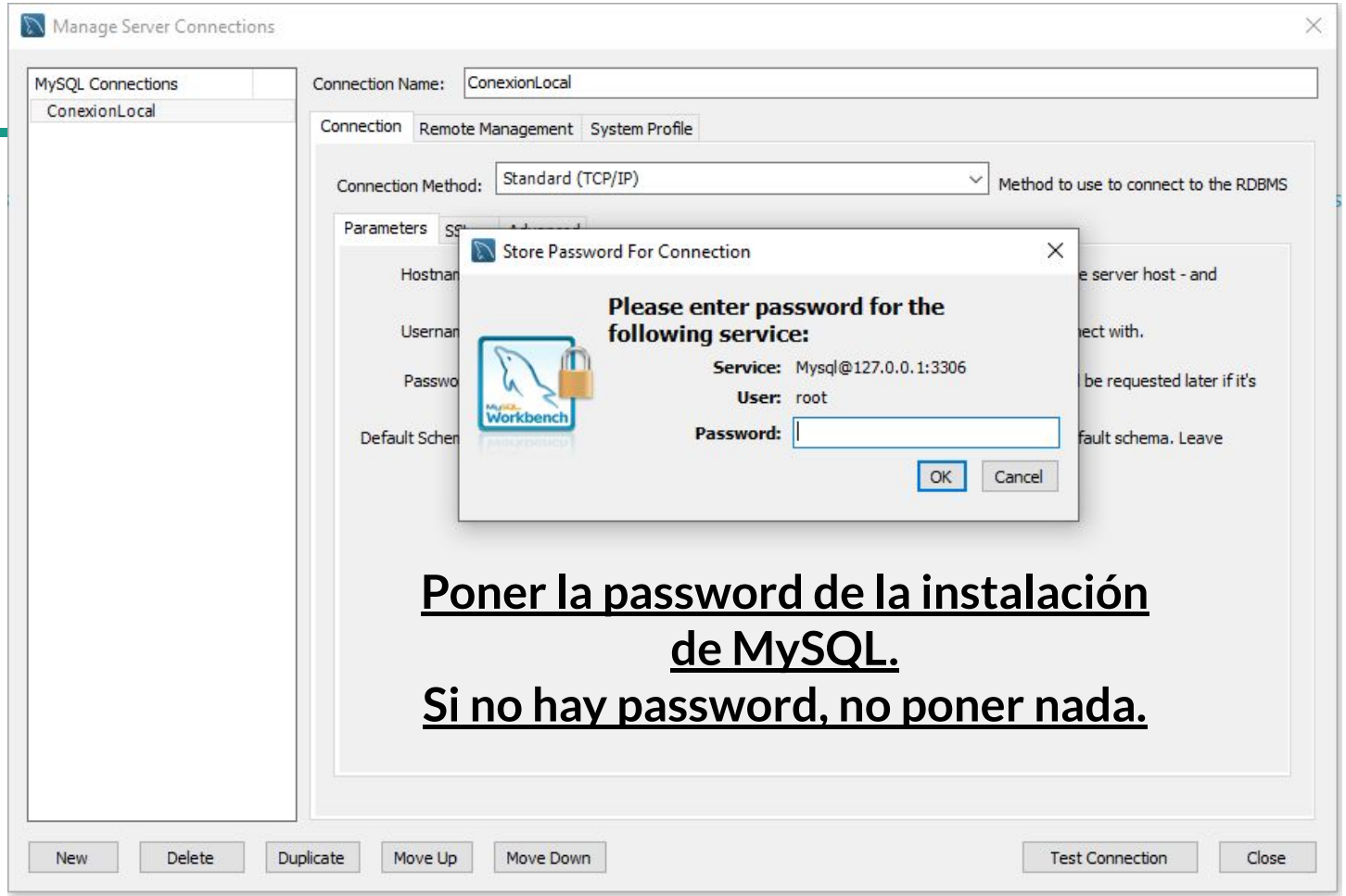
Read the Blog >

Discuss on the Forums >

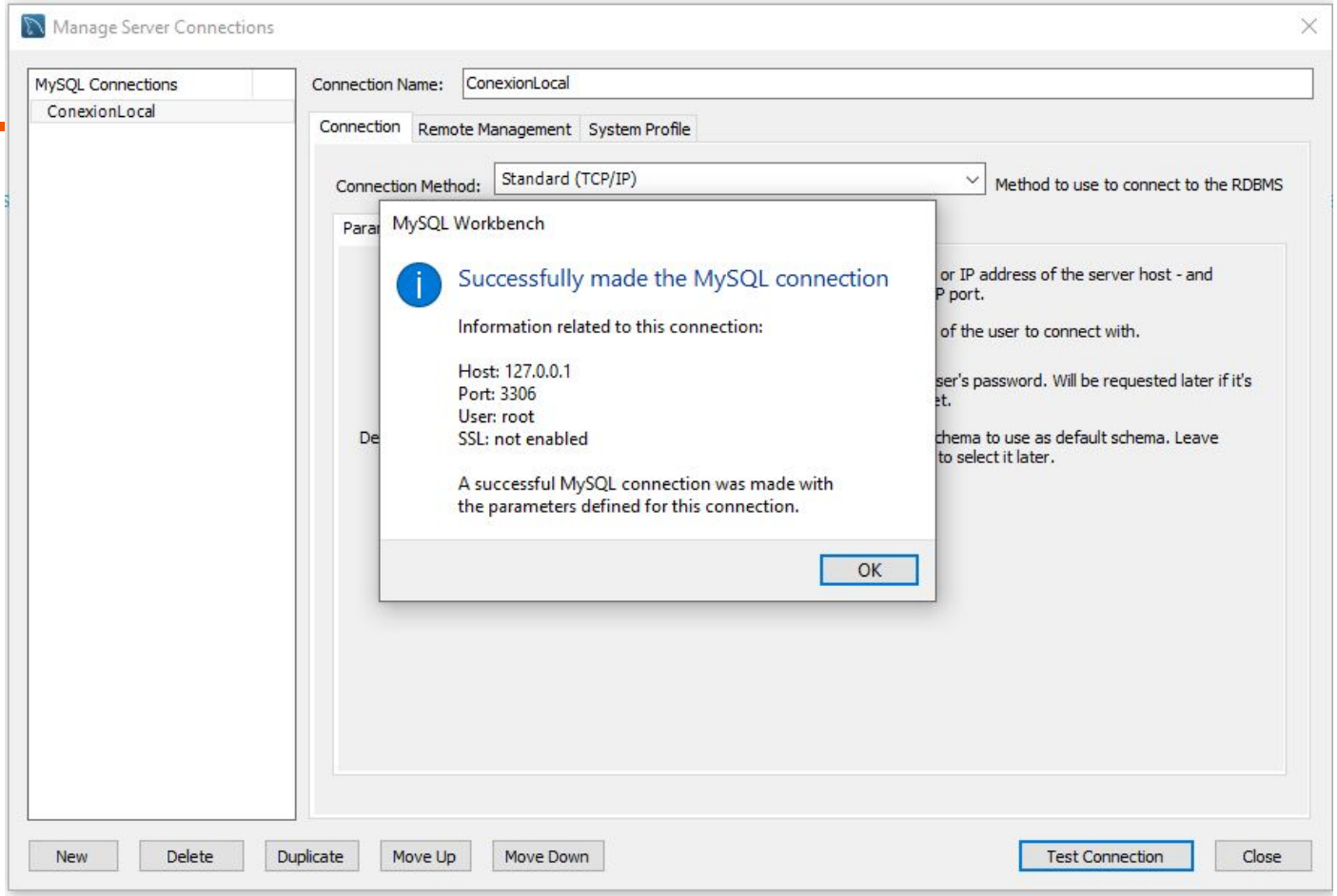
MySQL Connections




Manage Server Connections
MySQL Connections
ConexionLocal
Connection Name:
ConexionLocal
Connection
Remote Management
System Profile
Connection Method:
Standard (TCP/IP)
Method to use to connect to the RDBMS
Parameters
SSL
Advanced
Hostname:
127.0.0.1
Port:
3306
Name or IP address of the server host - and TCP/IP port.
Username:
root
Name of the user to connect with.
Password:
Store in Vault ...
Clear
The user's password. Will be requested later if it's not set.
Default Schema:
The schema to use as default schema. Leave blank to select it later.
New
Delete
Duplicate
Move Up
Move Down
Test Connection
Close



Poner la password de la instalación de MySQL.
Si no hay password, no poner nada.


Manage Server Connections
MySQL Connections
ConexionLocal
Connection Name:
ConexionLocal
Connection
Remote Management
System Profile
Connection Method:
Standard (TCP/IP)
Method to use to connect to the RDBMS
MySQL Workbench
Successfully made the MySQL connection
Information related to this connection:
Host: 127.0.0.1
Port: 3306
User: root
SSL: not enabled
A successful MySQL connection was made with the parameters defined for this connection.
OK
New
Delete
Duplicate
Move Up
Move Down
Test Connection
Close

Tipos de datos básicos - PT 1

- **TinyInt**: Es un número 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. Con signo el rango de valores va desde -32768 a 32767. Sin signo, el rango de valores es de 0 a 65535.
- **MediumInt**: Número entero con o sin signo. Con signo el rango de valores va desde -8.388.608 a 8.388.607. Sin signo el rango va desde 0 a16777215.
- **Integer**, **Int**: Número entero con o sin signo. Con signo el rango de valores va desde -2147483648 a 2147483647. Sin signo el rango va desde 0 a 429.4967.295

Tipos de datos básicos - PT 2

- **BigInt**: Número entero con o sin signo. Con signo el rango de valores va desde -9.223.372.036.854.775.808 a 9.223.372.036.854.775.807. Sin signo el rango va desde 0 a 18.446.744.073.709.551.615.
- **Float**: Número pequeño en coma flotante de precisión simple. Los valores válidos van desde -3.402823466E+38 a -1.175494351E-38, 0 y desde 1.175494351E-38 a 3.402823466E+38.
- **xReal, Double**: Número en coma flotante de precisión doble. Los valores permitidos van desde -1.7976931348623157E+308 a -2.2250738585072014E-308, 0 y desde 2.2250738585072014E-308 a 1.7976931348623157E+308.

- **A más precisión, más bytes ocupan en la base de datos.**

Tipos de datos - PT 3

- **Date**: Tipo fecha, almacena una fecha. El rango de valores va desde el 1 de enero del 1001 al 31 de diciembre de 9999. El formato de almacenamiento es de año-mes-dia.
- **DateTime**: Combinación de fecha y hora. El rango de valores va desde el 1 de enero del 1001 a las 0 horas, 0 minutos y 0 segundos al 31 de diciembre del 9999 a las 23 horas, 59 minutos y 59 segundos. El formato de almacenamiento es de año-mes-dia horas:minutos:segundos.
- **TimeStamp**: Combinación de fecha y hora. El rango va desde el 1 de enero de 1970 al año 2037.
- **Time**: Almacena una hora. El rango de horas va desde -838 horas, 59 minutos y 59 segundos a 838, 59 minutos y 59 segundos. El formato de almacenamiento es de 'HH:MM:SS'.
- **Year**: Almacena un año. El rango de valores permitidos va desde el año 1901 al año 2155. El campo puede tener tamaño dos o tamaño 4 dependiendo de si queremos almacenar el año con dos o cuatro dígitos.

Tipos de datos - PT 4

- **Char(n):** Almacena una cadena de **longitud fija**. La cadena podrá contener desde 0 a 255 caracteres.
- **VarChar(n):** Almacena una cadena de **longitud variable**. La cadena podrá contener desde 0 a 255 caracteres.

Dentro de los tipos de cadena se pueden distinguir otros dos subtipos, los tipo Test y los tipo BLOB (Binary large Object). La diferencia entre un tipo y otro es el tratamiento que reciben a la hora de realizar ordenamientos y comparaciones.

Mientras que el tipo test se ordena sin tener en cuenta las Mayúsculas y las minúsculas, el tipo BLOB se ordena teniéndose en cuenta.

Los tipos BLOB se utilizan para almacenar datos binarios como pueden ser ficheros.

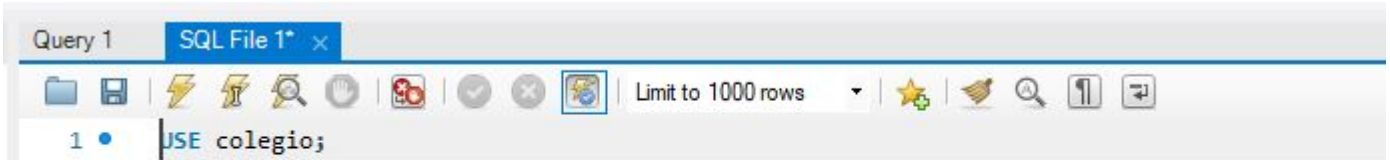
- **TinyText y TinyBlob:** Columna con una longitud máxima de 255 caracteres.
- **Blob y Text:** Un texto con un máximo de 65535 caracteres.

Vamos a crear la base de datos

CREATE DATABASE colegio CHARACTER SET UTF8mb4 COLLATE utf8mb4_bin;

USE colegio;

- CREATE DATABASE sirve para crear la base de datos.
- CHARACTER SET sirve para dotar de un rango de lenguaje.
- UTF8MB4: Acepta palabras de origen latino y emoticonos.



- USE: selecciona la base de datos.

Crear tabla asignatura de la base de datos

```
CREATE TABLE asignatura (
 id INT(6) NOT NULL auto_increment,
 nombre VARCHAR(25) NOT NULL,
 PRIMARY KEY  (id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Preguntas:

- ¿Qué significa auto_increment?
- ¿Y not null? ¿Y primary key? e ¿innodb?

Crear tabla alumno

```
CREATE TABLE alumno (
  id INT(6) NOT NULL auto_increment,
  nombre VARCHAR(25) NOT NULL,
  apellido VARCHAR (25) NOT NULL,
  fecha_nacimiento DATE NOT NULL,
  PRIMARY KEY  (id)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Crear tabla nota

```
CREATE TABLE nota (
 id INT(6) NOT NULL auto_increment,
 asignatura_id INT,
 calificacion FLOAT NOT NULL,
 fecha_examen DATE NOT NULL,
 convocatoria INT(6),
 alumno_id INT,
  INDEX alum_ind (alumno_id),
  FOREIGN KEY (alumno_id)
    REFERENCES alumno(id)
     ON DELETE CASCADE,
  INDEX asignat_ind (asignatura_id),
  FOREIGN KEY (asignatura_id)
    REFERENCES asignatura(id)
     ON DELETE CASCADE,
 PRIMARY KEY (id)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Investigación ¿Qué significa en MySQL?

- ¿Delete cascade?
- ¿Update cascade?
- ¿Qué son los índices?

Crear tabla labor_extra

```
CREATE TABLE labor_extra (
 puesto VARCHAR(50) NOT NULL,
 alumno_id INT NOT NULL,
  INDEX alum_ind (alumno_id),
  FOREIGN KEY (alumno_id)
    REFERENCES alumno(id)
     ON DELETE CASCADE,
 PRIMARY KEY (puesto, alumno_id)
) ENGINE=InnoDB  DEFAULT CHARSET=latin1
AUTO_INCREMENT=1;
```

Restricciones en las columnas

NOT NULL: Establece la obligatoriedad de que esta columna tenga un valor no nulo. Se debe especificar junto a la columna a la que afecta. Los valores nulos no ocupan espacio, y son distintos a 0 y al espacio en blanco.

UNIQUE: Evita valores repetidos en una columna, admitiendo valores nulos.

DEFAULT: Establece un valor por defecto para esa columna, si no se le asigna ninguno.

CHECK: Comprueba que se cumpla una condición determinada al rellenar esa columna.

PRIMARY KEY: Establece el conjunto de columnas que forman la clave primaria de esa tabla. Se comporta como única y obligatoria sin necesidad de explicitarlo. Sólo puede existir una clave primaria por tabla. Puede ser referenciada como clave ajena por otras tablas. Crea un índice automáticamente cuando se habilita o se crea esta restricción.

FOREIGN KEY: Establece que el contenido de esta columna será uno de los valores contenidos en una columna de otra tabla maestra. Esta columna marcada como clave ajena puede ser NULL. No hay límite en el número de claves ajenas. La clave ajena puede ser otra columna de la misma tabla.

Vamos a insertar datos - PT 1

```
-- insertar asignatura

INSERT INTO asignatura(nombre) VALUES ('Matemáticas');

INSERT INTO asignatura(nombre) VALUES ('Lengua');
```

Vamos a insertar datos - PT 2

```
-- insertar alumno

INSERT INTO alumno(nombre, apellido, fecha_nacimiento) VALUES ('Juan', 'Quesada', '1980-09-03');

INSERT INTO alumno(nombre, apellido, fecha_nacimiento) VALUES ('Manuel', 'Rico', '1992-11-10');

INSERT INTO alumno(nombre, apellido, fecha_nacimiento) VALUES ('Pedro', 'Riesgo', '1980-01-05');

INSERT INTO alumno(nombre, apellido, fecha_nacimiento) VALUES ('Maria', 'Valenzuela', '1986-12-19');
```

Vamos a insertar datos - PT 3

```
-- insertar nota
INSERT INTO nota(asignatura_id, calificacion, fecha_examen,convocatoria, alumno_id) VALUES (1, 7, '2018-12-19', 1, 1);
INSERT INTO nota(asignatura_id, calificacion, fecha_examen,convocatoria, alumno_id) VALUES (2, 5, '2018-11-03', 2, 1);
INSERT INTO nota(asignatura_id, calificacion, fecha_examen,convocatoria, alumno_id) VALUES (1, 3, '2018-11-03', 3, 2);
INSERT INTO nota(asignatura_id, calificacion, fecha_examen,convocatoria, alumno_id) VALUES (2, 8, '2018-11-03', 1, 2);
INSERT INTO nota(asignatura_id, calificacion, fecha_examen,convocatoria, alumno_id) VALUES (1, 2, '2018-07-05', 2, 3);
INSERT INTO nota(asignatura_id, calificacion, fecha_examen,convocatoria, alumno_id) VALUES (2, 5, '2018-11-03', 1, 3);
INSERT INTO nota(asignatura_id, calificacion, fecha_examen,convocatoria, alumno_id) VALUES (1, 9, '2018-09-13', 3, 4);
INSERT INTO nota(asignatura_id, calificacion, fecha_examen,convocatoria, alumno_id) VALUES (2, 5, '2018-11-23', 1, 4);
```

Vamos a insertar datos - PT 4

```
-- insertar labor extra

INSERT INTO labor_extra(puesto, alumno_id) VALUES ('Delegado', 1);

INSERT INTO labor_extra(puesto, alumno_id) VALUES ('Director', 2);
```

Selección de datos - PT 1

La recuperación de los datos en el lenguaje SQL se realiza mediante la sentencia SELECT.Esta sentencia permite indicar la información que se quiere recuperar. Esta es la sentencia SQL, con diferencia, más habitual. La sentencia SELECT consta de cuatro partes básicas:

- La cláusula **SELECT** seguida de la descripción de lo que se desea ver, los nombres de las columnas a seleccionar. Esta parte es obligatoria.
- La cláusula **FROM** seguida de la especificación de las tablas de las que se han de obtener los datos. Esta parte es obligatoria.
- La cláusula **WHERE** seguida por un criterio de selección, una condición. Esta parte es opcional.
- La cláusula **ORDER BY** seguida por el criterio de ordenación. Esta parte es opcional.

Selección de datos - PT 2

- select * from asignatura; //Selecciona todos los campos de asignatura
- select nombre, apellido from alumno //Es mejor anotar qué atributos mostrar.
- select a.nombre, a.apellido from alumno a //Podemos nombrar la tabla

Modificar una tabla - Alter Table

Para insertar a la tabla alumno el campo apellido2 bastaría con lanzar la instrucción:

- ALTER TABLE alumno ADD COLUMN apellido2 VARCHAR(25) AFTER apellido;

Actualizar el valor de un campo de una tabla

Mediante la instrucción UPDATE es posible actualizar un conjunto de datos sobre una tabla.

Hay que tener mucho cuidado con filtrar de manera adecuada cuáles serán las filas a actualizar.

Antes de lanzar la consulta hay que comprobar que la condición WHERE es la adecuada.

Para localizar las filas a editar es mejor hacerlo usando la clave primaria dentro del WHERE.

Veamos varios ejemplos

Ejemplos de actualizar

UPDATE alumno SET apellido2 = 'Palazon' WHERE id = 1;

UPDATE alumno SET apellido2 = 'Gomez' WHERE id = 2;

UPDATE alumno SET apellido2 = 'Leost' WHERE id = 3;

// UPDATE alumno SET apellido2 = 'Martin' WHERE nombre = 'Maria' and apellido = 'Gutierrez';

// Si actualizamos usando los campos nombre y apellido puede ocurrir que actualicemos varios alumnos que se llamen igual. Es mejor localizar a los registros por su clave primaria.

Inner Join - Varios ejemplos

En este ejemplo se muestran las calificaciones de los alumnos. Es necesario usar las tablas alumno, nota y asignatura . Se ha ordenado por apellido mediante ORDER BY. Por defecto el resultado obtenido es en orden ascendente.

```
SELECT a.nombre, asig.nombre as asignatura,
n.calificacion
FROM alumno a
        INNER JOIN nota n ON a.id =
n.alumno_id
        INNER JOIN asignatura asig ON asig.id
= n.asignatura_id
ORDER BY apellido
```

```
SELECT a.nombre, asig.nombre as asignatura,
n.calificacion
FROM alumno a, asignatura asig, nota n
WHERE
        a.id = n.alumno_id
        AND
        asig.id = n.asignatura_id
ORDER BY apellido
```

Uso de “IN” **Ojo: No es muy eficiente

Se puede seleccionar mediante IN alumnos que tengan alguna de las notas deseados. En este ejemplo se quieren obtener los alumnos que tengan de nota de examen 5,6,7 ó 8.

```
SELECT a.nombre, asig.nombre as asignatura, n.calificacion

FROM alumno a

    INNER JOIN nota n ON a.id = n.alumno_id

    INNER JOIN asignatura asig ON asig.id = n.asignatura_id

WHERE n.calificacion in (5, 6, 7, 8)

ORDER BY calificacion
```

Uso de Like

Sobre los campos de texto se puede emplear el comando LIKE. En la siguiente consulta se muestran los alumnos cuyo nombre empieza por 'Ma' o bien su apellido contiene los caracteres 'ar' .

SELECT a.nombre, a.apellido

FROM alumno a

WHERE a.nombre LIKE 'Ma%' OR a.apellido LIKE '%ar%'

Left Outer Join

En ocasiones queremos unir el contenido de 2 tablas.

Aunque la clave de la primera tabla no exista en la segunda queremos sacar todo el contenido de la primera. En estos casos se utiliza LEFT OUTER JOIN.

En el ejemplo queremos sacar todos los alumnos y además indicar la labor extra que realicen.

Si no disponen de ninguna labor los queremos mostrar igualmente.

En ese caso el valor del puesto será NULL como se muestra en la salida obtenida.

Left Outer Join - Ejemplo

```
SELECT a.nombre, l.puesto

FROM alumno a

        LEFT OUTER JOIN labor_extra l ON a.id = l.alumno_id
```

Group By - PT 1

- Mediante el uso de GROUP BY podemos agrupar por nombre y apellido y calcular la media de las calificaciones de cada alumno (empleando la función agregada AVG).
- Hay que tener en cuenta que los campos que se encuentran en la sección SELECT sólo pueden ser los campos que se encuentran en la sección GROUP BY (aparte de la función agregada).
- La consulta ordena de manera descendente los resultados mediante el uso de la palabra DESC en ORDER BY.
- Las funciones agregadas disponibles en mysql son:
    - SUM Realiza la suma de los valores agrupados
    - AVG Media de los valores agrupados
    - COUNT Cuenta los valores agrupados

Group By - Pt 2

```
SELECT a.nombre, a.apellido, avg(n.calificacion) as media
FROM alumno a
        INNER JOIN nota n ON a.id = n.alumno_id
        INNER JOIN asignatura asig ON asig.id = n.asignatura_id
GROUP BY a.nombre, a.apellido
ORDER BY media DESC
```

Filtrados usando Having

- Mediante el uso de HAVING podemos filtrar los resultados obtenidos mediante la función agregada AVG.
- La consulta devuelve las medias de notas de los alumnos cuya media es mayor ó igual de 5.

```
SELECT a.nombre, a.apellido, avg(n.calificacion) as media

FROM alumno a

        INNER JOIN nota n ON a.id = n.alumno_id

        INNER JOIN asignatura asig ON asig.id = n.asignatura_id

GROUP BY a.nombre, a.apellido

HAVING media >= 6

ORDER BY media desc
```

Having Vs Where

- Usar having es muy costoso.
- Investigue cuales son las diferencias entre usar HAVING y usar WHERE.
- ¿Es posible realizar la consulta anterior usando “WHERE avg(n.calificacion) > 6”? ¿Qué ocurre?

Delete

- Para el borrado de datos en una tabla se utiliza DELETE.
- La condición WHERE será necesaria para borrar solamente los datos deseados.
- En este caso se desea borrar aquellos alumnos que tengan el puesto de 'Director' en la tabla labor_extra

// Opción menos recomendable - DELETE FROM labor_extra WHERE puesto = 'Director'

// Opción recomendable - DELETE FROM labor_extra WHERE id = 1

Borrar tablas - Drop Table

- Para borrar tablas se usa el comando DROP TABLE.
- Una vez que se disponen de claves foráneas (FK) el borrado de tablas se debe realizar en un orden concreto ya que existen dependencias entre tablas.
- En este caso he borrado las tablas en el orden inverso de creación para evitar estos problemas.

```
DROP TABLE labor_extra;

DROP TABLE nota;

DROP TABLE alumno;

DROP TABLE asignatura;
```

Borrado de la base de datos

Si se disponen de los permisos necesarios el borrado de la base de datos se realiza mediante DROP DATABASE. En este caso sería:

DROP DATABASE colegio;

Extra - Comandos para usar en Where

=	Igualdad	SELECT * FROM nota WHERE calificacion = 6;
!=, <>, ^=	Desigualdad	SELECT * FROM nota WHERE calificacion != 6;
<	Menor que	SELECT * FROM nota WHERE calificacion < 6;
>	Mayor que	SELECT * FROM nota WHERE calificacion > 6;
<=	Menor o igual que	SELECT * FROM nota WHERE calificacion <= 6;

Extra - Comandos para usar en Where

>=	Mayor o igual que	SELECT * FROM nota WHERE calificacion >= 6;
not in	Distinto a cualquiera de los miembros entre paréntesis	SELECT * FROM nota WHERE calificacion not in (2,3,4);
between	Contenido en el rango	SELECT * FROM nota WHERE calificacion between 2 and 4;
not between	Fuera del rango	SELECT * FROM nota WHERE calificacion not between 2 and 4;

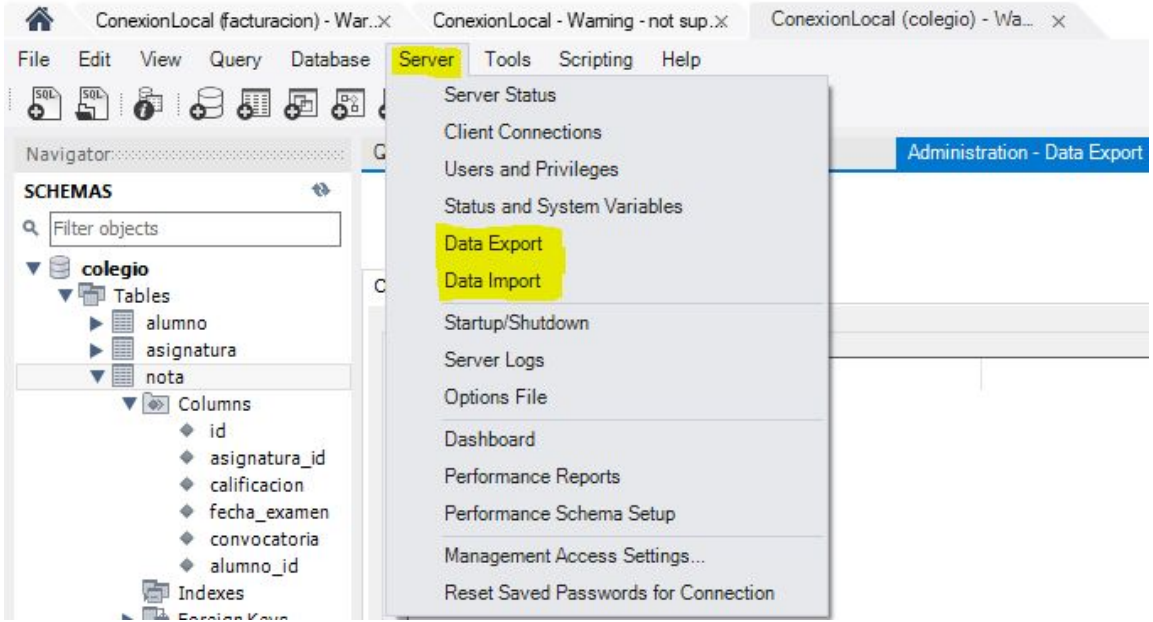
Extra - Usar Distinct

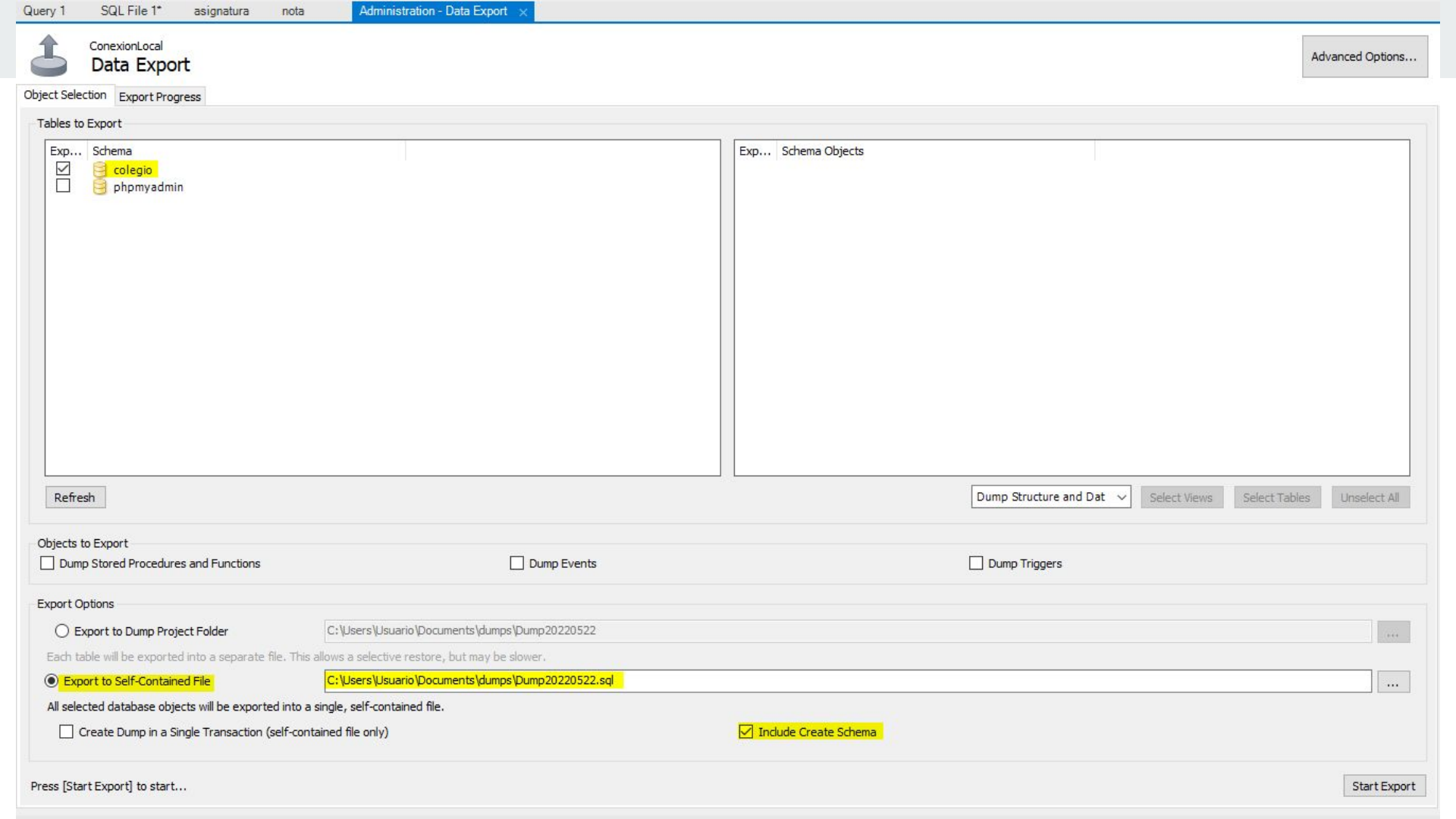
- Cuando se realiza una consulta sobre una tabla en la que se extrae información de varias columnas, puede ocurrir que, si no incluimos la/s columna/s que forman la clave principal, obtengamos filas repetidas en la respuesta.
- Si este comportamiento no nos resulta satisfactorio podemos utilizar la cláusula DISTINCT para eliminar las filas duplicadas obtenidas como respuesta a una consulta.

SELECT distinct calificacion, id, asignatura_id FROM nota;

En caso de haber notas repetidas no las mostrará.

Exportar/ Importar una BD en Workbench




Query 1
SQL File 1*
asignatura
nota
Administration - Data Export
ConexionLocal
Data Export
Advanced Options...
Object Selection
Export Progress
Tables to Export
Exp...
Schema
colegio
phpmyadmin
Exp...
Schema Objects
Refresh
Dump Structure and Dat
Select Views
Select Tables
Unselect All
Objects to Export
Dump Stored Procedures and Functions
Dump Events
Dump Triggers
Export Options
Export to Dump Project Folder
C:\Users\Usuario\Documents\dumps\Dump20220522
Each table will be exported into a separate file. This allows a selective restore, but may be slower.
Export to Self-Contained File
C:\Users\Usuario\Documents\dumps\Dump20220522.sql
All selected database objects will be exported into a single, self-contained file.
Create Dump in a Single Transaction (self-contained file only)
Include Create Schema
Press [Start Export] to start...
Start Export