

Gestiona bases de datos con MySQL

Día 3

Gerald Kogler

geraldo@servus.at

Instrucciones preliminares

Podéis descargar este documento aquí:

https://github.com/geraldo/curso_mysql/blob/main/mysql_dia3.pdf

Posteriormente hace falta descargar el instalador de XAMPP desde aquí:

<https://www.apachefriends.org/download.html>

Seguimos los pasos de instalación de XAMPP para el sistema operativo usado. Si usáis Windows entonces recomiendo instalarlo en la carpeta predefinida:

<c:\xampp>

Por cualquier problema que nos encontramos podemos consultar la ayuda online:

https://www.apachefriends.org/faq_windows.html

Temario día 3

7 Triggers, eventos y índices

7.1 Triggers

7.2 Eventos

8 Mantenimiento de MySQL

8.1 ANALYZE TABLE

8.2 CHECK TABLE

8.3 CHECKSUM TABLE

8.4 Defragmentar tabla

8.5 FLUSH

8.6 OPTIMIZE TABLE

9 Administración de la Base de Datos

9.1 Copias de seguridad y restauración

9.2 Permisos

9.3 Roles

10 Conexión de una base de datos con un sitio web

10.1 Introducción a PHP

10.2 Práctica para crear una web básica

Ejercicios SQL

Para terminar la sesión practicamos algunos comandos SQL que aprendimos. Para eso cargamos un nuevo fichero de datos con consumos energéticos llamado **consums_electrics.sql**.

1. ¿Qué consumo eléctrico tiene el municipio Prat de Llobregat?
2. ¿Cuál es el municipio de Cataluña que tiene más consumo eléctrico industrial?
3. ¿Cuánto consumo eléctrico tiene la industria en la comarca del Baix Llobregat?
4. Agrupa el consumo eléctrico de Cataluña por sectores.
5. Agrupa el consumo eléctrico de la comarca de Barcelona por municipios.
6. Agrupa el consumo eléctrico de la comarca de Barcelona por municipios y sectores.
7. Mostramos un listado de todos los consumos privados por provincia.
8. Mostramos un listado de todos los consumos privados por municipio y habitante.

4 – Triggers y eventos

MySQL – Triggers

Una trigger es un evento que se dispara cuando una condición se cumpla. Cada trigger ejecuta una función que es un poco especial cada vez que una condición se cumpla.

Son muy útiles para automatizar cambios, puede hacer desde calcular alguna columna hasta modificar otras tablas para hacer que los valores sean consistentes.

El trigger se puede activar de dos maneras (*trigger_time*):

- Antes de la instrucción, esto permite modificar los valores que esta aplica (*BEFORE*).
- Después de la instrucción, esto permite utilizar funciones que lean los cambios hechos (*AFTER*).

El trigger se ejecuta antes o después de cada operación insertado (*INSERT*), actualizado (*UPDATE*) o borrado (*DELETE*). Se puede ejecutar una vez por fila (*FOR EACH ROW*) o por el procedimiento completo.

MySQL – Triggers

La sintaxis del trigger tiene la siguiente estructura:

CREATE

```
[DEFINER = user]
TRIGGER [IF NOT EXISTS] trigger_name
trigger_time trigger_event
ON tbl_name FOR EACH ROW
[trigger_order]
trigger_body
```

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }

trigger_order: { FOLLOWS | PRECEDES } other_trigger_name

MySQL – Triggers

El siguiente ejemplo define un trigger que añade la fecha de creación:

```
DELIMITER $$

CREATE
  TRIGGER ultima_actualizacion
  BEFORE UPDATE
  ON cursos
  FOR EACH ROW
  BEGIN
    SET NEW.ultima_actualizacion = NOW();
  END$$

DELIMITER ;
```

Primero tendremos que añadir la columna nueva *last_update* a la tabla *cursos*:

```
ALTER TABLE cursos ADD ultima_actualizacion
DATETIME NULL DEFAULT NULL;
```


MySQL – Triggers

Revisamos algunos ejemplos con triggers del manual oficial en <https://dev.mysql.com/doc/refman/8.0/en/trigger-syntax.html>

También puedes consultar este manual muy interesante con casos prácticos: <https://www.mysqltutorial.org/mysql-triggers/>

Ejercicios:

- Crear un Trigger para actualizar la fecha de la última actualización.
- Crear un Trigger para añadir el usuario que ha hecho la última actualización.
- Crear un Trigger para guardar la suma de cursos de un participante (nuevo columna num_cursos).

MySQL – Events

Eventos son acciones que se ejecutarán en un momento específico definido por el evento. Con la siguiente consulta podemos mostrar un listado de eventos activos:

```
SHOW PROCESSLIST;
```

Para crear un evento nuevo usamos la orden **CREATE EVENT** de la siguiente manera:

```
CREATE EVENT [IF NOT EXIST] event_name  
ON SCHEDULE schedule  
DO  
event_body
```

El siguiente evento se ejecutará cada minuto durante una hora y dejará un mensaje en la tabla *messages*:

```
CREATE EVENT recurring_log  
ON SCHEDULE EVERY 1 MINUTE  
STARTS CURRENT_TIMESTAMP  
ENDS CURRENT_TIMESTAMP + INTERVAL 1 HOUR  
DO  
    INSERT INTO messages(message)  
    VALUES(CONCAT('Running at ', NOW()));
```

Para más información consulta este manual: <https://www.mysqltutorial.org/mysql-events/mysql-create-event/>

8 – Mantenimiento de MySQL

MySQL – Analyze Table

ANALYZE TABLE genera estadísticas de una tabla en MySQL.

```
ANALYZE TABLE `consums_electrics`;
```

MySQL – Check Table

CHECK TABLE comprueba que una tabla no tenga errores.

```
CHECK TABLE `consums_electrics`;
```

MySQL – Checksum Table

CHECKSUM TABLE devuelve un checksum, que es un número entero con la cantidad de bytes que ocupa una tabla. Puedes usar esta consulta para verificar que el contenido es exactamente el mismo antes y después de un backup, un rollback, o cualquier otra operación que está pensada para devolver los datos a su estado anterior.

```
CHECKSUM TABLE `consums_electrics`;
```

MySQL – Defragmentar tabla

Insertaciones random o eliminaciones de un índice secundario pueden causar el índice de estar fragmentado.

Fragmentación significa que el orden físico de las páginas índice en el disco duro no es el mismo que el orden de los índices en los registros de las tablas. También puede significar que haya muchos bloques sin usar que han sido alocados al índice.

Un síntoma de la fragmentación es que la tabla ocupa más espacio que debería. Otra síntoma es que un escaneo de la tabla dura mucho más de lo que debería.

```
SELECT COUNT(*) FROM t WHERE non_indexed_column <> 12345;
```

Esta consulta hace un escaneo completo de una tabla de MySQL, lo que es la operación más lenta en una tabla grande.

Para hacer más rápido los escaneos, se puede hacer periódicamente la siguiente operación que rehace la tabla:

```
ALTER TABLE tbl_name ENGINE=INNODB;
```

MySQL – Flush

Hay diferentes tipos de FLUSH de una tabla, lo que necesitan diferentes tipos de permisos. Se puede borrar y rehacer diferentes caches internos y también soltar bloqueos.

```
FLUSH TABLE `consums_electrics`;
```


MySQL – Optimize Table

OPTIMIZE TABLE reorganiza la memoria física de una tabla y sus índices. Eso reduce el espacio usado y mejora las operaciones de entrada y salida (I/O), es decir, cualquier consulta a la tabla. Los cambios aplicados dependen de la storage engine (en muchos casos es InnoDB, más info en https://dev.mysql.com/doc/refman/8.0/en/glossary.html#glos_storage_engine) utilizadas por la tabla.

```
OPTIMIZE TABLE `consums_electrics`;
```

9 – Administración de MySQL

MySQL – Backup and Restore

Hay diferentes formas de hacer backups de MySQL, dependiendo del cliente utilizado:

Una forma fácil y eficiente es usar Export y Import de phpMyAdmin, en este caso es recomendable utilizar el formato SQL ya que así guardamos también la estructura y los parámetros de las tablas de MySQL.



Exporting rows from "courses" table

Export templates:

New template:

Existing templates:

Template:

Export method:

- ☒ Quick - display only the minimal options
- ☐ Custom - display all possible options

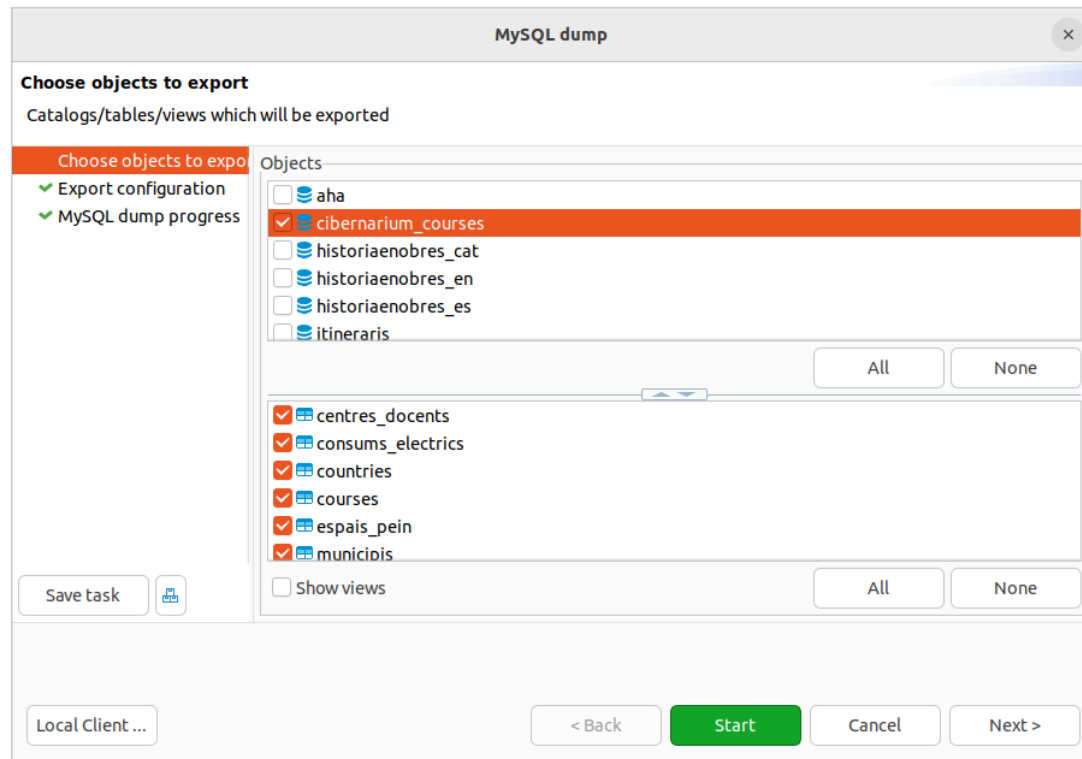
Format:

Rows:

- ☐ Dump some row(s)
- Number of rows:
- Row to begin at:
- ☒ Dump all rows

MySQL – Backup and Restore

La herramienta **mysqldump** está incluido en algunos clientes como MySQL Shell y DBeaver y ofrece más opciones que un simple export de phpMyAdmin.










Más información: <https://dev.mysql.com/doc/refman/8.4/en/mysqldump.html>



MySQL – Permisos y roles

Por definición ya vienen algunos usuarios para gestionar MySQL. Por lo menos siempre va a existir un usuario de administración que tenga todos los permisos sobre las bases de datos, en nuestro caso llamado *root*:

[Browse](#) [Structure](#) [SQL](#) [Search](#) [Insert](#) [Export](#) [Import](#) [Privileges](#) [Operations](#) [Tracking](#) [Triggers](#)

 **Users having access to "cibernarium_courses.courses"**

	User name	Host name	Type	Privileges	Grant	Action
<input type="checkbox"/>	debian-sys-maint	localhost	global	ALL PRIVILEGES	Yes	 Edit privileges  Export
<input type="checkbox"/>	mysql.infoschema	localhost	global	SELECT	No	 Edit privileges  Export
<input type="checkbox"/>	root	localhost	global	ALL PRIVILEGES	Yes	 Edit privileges  Export

 ☐ [Check all](#) With selected:  [Export](#)

MySQL – Permisos y roles

Podemos crear un usuario que solamente tenga derechos de visualizar los datos, pero no modificarlos:

Server: localhost:3306

Databases SQL Status User accounts Export Import Settings Binary log Replication

Add user account

Login Information

User name: Use text field ▼ visor

Host name: Any host ▼ % ⓘ

Password: Use text field ▼ Strength:

Re-type:

Authentication plugin: Caching sha2 authentication ▼

Generate password:

Database for user account

- ☐ Create database with same name and grant all privileges.
- ☐ Grant all privileges on wildcard name (username_%).
- ☐ Grant all privileges on database cibernarium_courses.

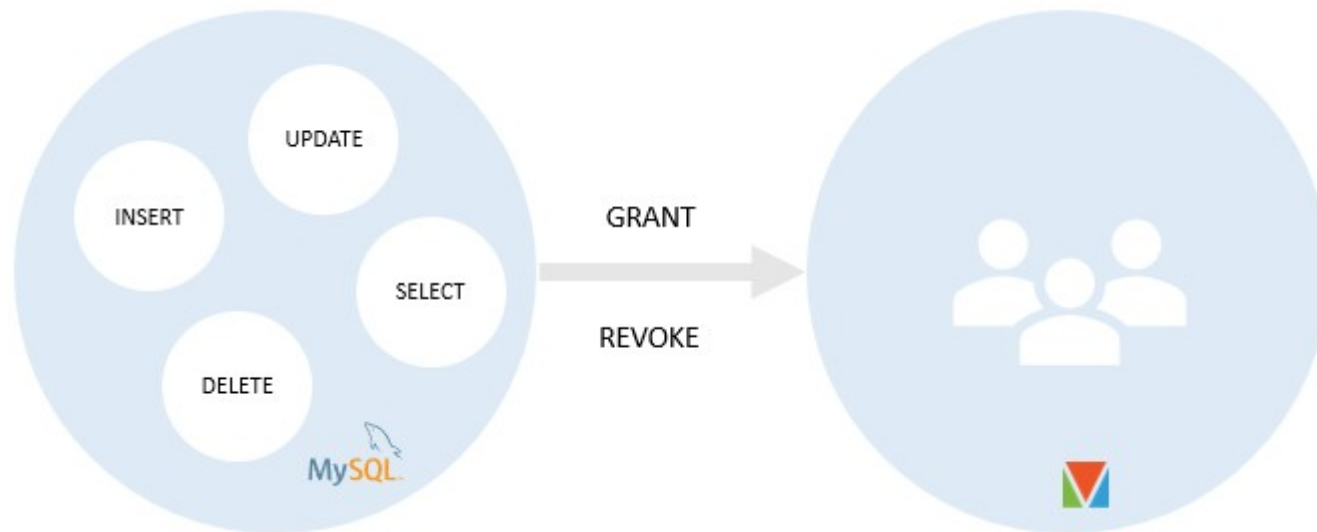
Global privileges ☒ Check all

Note: MySQL privilege names are expressed in English.

<input checked="" type="checkbox"/> Data	<input type="checkbox"/> Structure	<input type="checkbox"/> Administration	<input type="checkbox"/> Resource limits
<input checked="" type="checkbox"/> SELECT <input type="checkbox"/> INSERT <input type="checkbox"/> UPDATE <input type="checkbox"/> DELETE <input type="checkbox"/> FILE	<input type="checkbox"/> CREATE <input type="checkbox"/> ALTER <input type="checkbox"/> INDEX <input type="checkbox"/> DROP <input type="checkbox"/> CREATE TEMPORARY TABLES	<input type="checkbox"/> GRANT <input type="checkbox"/> SUPER <input type="checkbox"/> PROCESS <input type="checkbox"/> RELOAD <input type="checkbox"/> SHUTDOWN	<p><i>Note: Setting these options to 0 (zero) removes the limit.</i></p> <p>MAX QUERIES PER HOUR <input type="text" value="0"/></p> <p>MAX UPDATES PER HOUR <input type="text" value="0"/></p>

MySQL – Permisos y roles

Existe el concepto de los roles para facilitar el proceso de gestionar los permisos de los usuarios. Eso permite dar y quitar los mismo permisos a multiples usuarios dentro de un mismo rol:



Aquí un tutorial que explica la gestión de los permisos y roles con SQL: <https://www.mysqltutorial.org/mysql-administration/mysql-roles/>

10 – Conexión de una base de datos con un sitio web

MySQL – PHP

PHP es un lenguaje de programación muy popular para conectarse a bases de datos y manipularlos. En combinación con MySQL es un sistema muy potente para programar cualquier aplicación web. Esta combinación se usa en sistemas tan populares como Wordpress y muchas de las plataformas grandes en la web.

Haremos nuestro primer programa de PHP escribiendo lo siguiente a un fichero de texto que llamamos **index.php**:

```
<?php echo "hola mundo"; ?>
```

Lo guardamos en la carpeta **C:\xampp\htdocs** y lo llamamos desde el navegador con la URL

http://localhost/index.php

Para mostrar toda la información sobre nuestro servidor, hacemos un fichero **info.php** con el siguiente contenido:

```
<?php phpinfo(); ?>
```

PHP corre en el servidor y produce código **HTML** (y opcionalmente **CSS** y **JavaScript**) que luego se envía al cliente donde se renderiza en el navegador web.

MySQL – PHP

A partir de PHP 5 se puede trabajar con MySQL usando:

- **MySQLi** extensión
- **PDO** (PHP Data Objects)

PDO tiene la ventaja que es un sistema que también funciona con otros bases de datos. Es decir, en caso de cambiar a otra base de datos solamente hace falta cambiar la cadena de texto que configura la conexión. Aquí se explica la instalación de PDO (que en el caso de XAMPP ya está instalado):

<https://www.php.net/manual/en/pdo.installation.php>

MySQL – PHP

Los ficheros **.php** para programar una página web habitualmente son una mezcla de HTML con PHP. Modificamos *index.html* y ponemos de nuevo el siguiente contenido:

```
<!DOCTYPE html>
<html>

<head>
  <title>Hola Mundo</title>
</head>

<body>
  <h1>Hola mundo</h1>
  <?php echo "hola mundo"; ?>
</body>

</html>
```

Para detectar los fallos en PHP podemos mirar el log de Apache.

MySQL – PHP

A partir de ahora usaremos el fichero *database.php* donde se define un objeto *Database* que gestiona la conexión (en vez del usuario *root* también se puede usar uno específico que solamente tenga derechos de SELECT):

```
class Database {
    private $servername = "localhost";
    private $username = "root";
    private $password = "";
    private $db = "cibernarium_courses";

    public $conn;

    public function getConnection() {
        $this->conn = null;
        try {
            $this->conn = new PDO("mysql:host=$this->servername;dbname=$this->db", $this->username, $this->password);
            $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
        }
        catch(PDOException $exception){
            echo "Database could not be connected: " . $exception->getMessage();
        }
        return $this->conn;
    }
}
```

...

MySQL – PHP

Además define tres funciones que nos permiten hacer las consultas. La primera llamada *query()* se usa internamente para ejecutar las consultas:

```
private function query($sqlQuery) {  
    $stmt = $this->conn->prepare($sqlQuery);  
    $stmt->setFetchMode(PDO::FETCH_ASSOC);  
    $stmt->execute();  
    return $stmt;  
}
```

La función *queryOne()* hace consultas que devuelven exactamente 1 resultado:

```
public function queryOne($sqlQuery) {  
    $stmt = $this->query($sqlQuery);  
    return $stmt->fetch();  
}
```

La función *queryAll()* hace consultas que devuelven un listado de resultados:

```
public function queryAll($sqlQuery) {  
    $stmt = $this->query($sqlQuery);  
    return $stmt->fetchAll();  
}
```

MySQL – PHP

Ahora insertaremos el fichero *database.php* a *index.php* y creamos la conexión a MySQL usando el siguiente script:

```
include_once 'database.php';  
$database = new Database();  
$db = $database->getConnection();
```

Ahora creamos la primera consulta desde PHP a MySQL mostrando la cantidad de provincias que tiene Cataluña:

```
echo "Cataluña tiene ";  
$sqlQuery = 'SELECT count(distinct(nomprov)) AS num FROM municipis';  
$result = $database->queryOne($sqlQuery);  
echo $result['num'];  
echo " provincias:";
```

Para mostrar todo el listado de provincias, hacemos lo siguiente:

```
$sqlQuery = 'SELECT distinct(nomprov) FROM municipis';  
$result = $database->queryAll($sqlQuery);  
echo "<ul>";  
foreach ($result as $row) {  
    echo "<li>" . $row['nomprov'] . "</li>";  
}  
echo "</ul>";
```

MySQL – PHP

Ahora enlazaremos cada provincia con otro fichero llamado *provincia.php* para mostrar los datos de cada provincia cambiando la línea 24 a:

```
echo "<li><a href='provincia.php?codiprov='" . $row['codiprov'] . "'>" . $row['nomprov'] . "</a></li>";
```

Pasaremos un parámetro get al fichero que contiene el código de la provincia. La podemos recuperar usando la función `$_GET['codiprov']`:

```
include_once 'database.php';
$database = new Database();
$db = $database->getConnection();

if (!isset($_GET['codiprov']) || empty($_GET['codiprov']) || !is_numeric($_GET['codiprov'])) {
    echo "Tienes que añadir un parámetro con un 'codiprov' válido para ver los datos de la provincia.";
    exit();
}

$sqlQuery = 'SELECT distinct(nomprov) FROM municipios WHERE codiprov=' . $_GET['codiprov'];
$result = $database->queryOne($sqlQuery);
if (!$result) {
    echo "No existe ninguna provincia con el 'codiprov='" . $_GET['codiprov'];
    exit();
}
```

MySQL – PHP

Ahora mostramos el listado de comarcas que contiene esta provincia:

```
$nomprov = $result['nomprov'];
echo "<h1>" . $nomprov . "</h1>";

echo $nomprov;
echo " tiene ";
$sqlQuery = 'SELECT count(distinct(nomcomar)) AS num FROM municipis WHERE codiprov=' . $_GET['codiprov'];
$result = $database->queryOne($sqlQuery);
echo $result['num'];
echo " comarcas:";

$sqlQuery = 'SELECT distinct(nomcomar), codicomar FROM municipis WHERE codiprov=' . $_GET['codiprov'];
$result = $database->queryAll($sqlQuery);
echo "<ul>";
foreach ($result as $row) {
    echo "<li><a href='comarca.php?codicomar=" . $row['codicomar'] . "'>" . $row['nomcomar'] . "</a></li>";
}
echo "</ul>";
```


MySQL – PHP

Ahora enlazaremos cada comarca con otro fichero llamado *comarca.php* para mostrar los datos de cada comarca usando el código de la comarca que recogemos usando la función `$_GET['codicomar']`:

```
include_once 'database.php';
$database = new Database();
$db = $database->getConnection();

if (!isset($_GET['codicomar']) || empty($_GET['codicomar']) || !is_numeric($_GET['codicomar'])) {
    echo "Tienes que añadir un parámetro con un 'codicomar' válido para ver los datos de la comarca.";
    exit();
}

$sqlQuery = 'SELECT distinct(nomcomar) FROM municipis WHERE codicomar=' . $_GET['codicomar'];
$result = $database->queryOne($sqlQuery);
if (!$result) {
    echo "No existe ninguna comarca con el 'codicomar'=" . $_GET['codicomar'];
    exit();
}
```

MySQL – PHP

Ahora mostramos el listado de municipios que contiene esta comarca:

```
$nomcomar = $result['nomcomar'];
echo "<h1>Comarca . $nomcomar . "</h1>";

echo $nomcomar;
echo " tiene ";
$sqlQuery = 'SELECT count(distinct(nommuni)) AS num FROM municipis WHERE codicomar=' . $_GET['codicomar'];
$result = $database->queryOne($sqlQuery);
echo $result['num'];
echo " municipios:";

$sqlQuery = 'SELECT distinct(nommuni), codimuni FROM municipis WHERE codicomar=' . $_GET['codicomar'];
$result = $database->queryAll($sqlQuery);
echo "<ul>";
foreach ($result as $row) {
    echo "<li><a href='municipio.php?codimuni=' . $row['codimuni'] . '>' . $row['nommuni'] . "</a></li>";
}
echo "</ul>";
```

MySQL – PHP

Ahora creamos un último archivo *municipios.php* donde mostramos los datos de cada municipio. De nuevo usamos el código de la municipio que recogemos usando la función `$_GET['codimuni']`:

```
<?php
    include_once 'database.php';
    $database = new Database();
    $db = $database->getConnection();

    if (!isset($_GET['codimuni']) || empty($_GET['codimuni']) || !is_numeric($_GET['codimuni'])) {
        echo "Tienes que añadir un parámetro con un 'codimuni' válido para ver los datos de la municipio.";
        exit();
    }

    $sqlQuery = 'SELECT * FROM municipis WHERE codimuni=' . $_GET['codimuni'];
    $result = $database->queryOne($sqlQuery);
    if (!$result) {
        echo "No existe ninguna municipio con el 'codimuni'=" . $_GET['codimuni'];
        exit();
    }
?>

<h1>Municipio <?php echo $result['nommuni']; ?></h1>

<p>Área: <?php echo $result['areapol']; ?></p>
<p>Habitantes: <?php echo $result['habitants']; ?></p>
<p>Altitud: <?php echo $result['altitud']; ?></p>
```

MySQL – PHP

Tenéis todos los archivos de nuestro mini proyecto en la carpeta *php-final* del repositorio Github. Para seguir investigando PHP recomiendo este tutorial: <https://www.w3schools.com/php/>

Hay muchos tutoriales que introducen a la temática, también recomiendo visitar los siguientes cursos en el **Cibernarium**:

- **MySQL avançat - Preparació per al certificat 1Z0-909 MySQL 8.0 Database Developer d'Oracle:**
<https://cibernarium.barcelonactiva.cat/ficha-actividad?activityId=1422924>
- **Construcció d'APIs en PHP i MySQL per a persones programadores:**
<https://cibernarium.barcelonactiva.cat/ficha-actividad?activityId=1388997>

Espero que os ha agradado esta sesión.
Muchas gracias por vuestra asistencia.

Gerald Kogler

geraldo@servus.at

<https://geraldo.github.io>

<https://github.com/geraldo>

<https://gitlab.com/geraldo1>