MySQL

Esercizio 1

Implementare un servizio di API, che permette di accedere alle informazioni contenute nel dataset http://dati.toscana.it/dataset/lista-comuni-colpiti. In particolare, l'API deve permettere la consultazione delle seguenti informazioni:

- per ogni comune le coordinate geografiche
- per ogni provincia la popolazione prima del sisma

Soluzione

Come prima cosa, dobbiamo correggere alcuni problemi di codifica del dataset. Tramite Phpmyadmin modifichiamo i seguenti record:

- sostituire l'apostrofo di Sant'Angelo sul Nera con un normale apostrofo
- ripetere l'operazione anche per Sant'anatolia di Narco.

Per fare questo, accedere a Phpmyadmin, selezionare il database di interesse e la tabella contenente la lista dei comuni. Nel menu in alto selezionare il tasto mostra e scorrere l'elenco dei record fino a trovare il primo record di interesse. Premere sul tasto modifica in corrispondenza del record. Nel form che si apre correggere il campo Comune. Ripetere l'operazione anche per l'altro record.

Strutturiamo l'esercizio come un progetto, che poi utilizzeremo anche negli esercizi successivi. Nella htdocs di XAMPP creiamo una cartella progetto, dentro cui creiamo altre due cartelle:

- api, che conterrà tutte le api php che andremo a sviluppare
- libraries, che conterrà tutte le librerie php già esistenti

All'interno della cartella libraries posizioniamo la libreria dbLibrary.php, che permette la gestione dell'interazione tra PHP e MySQL.
Creiamo ora un file config.php che posizioniamo nella cartella api. Questo file contiene i parametri di configurazione della connessione al database e aprirà la connessione al database. Il file config.php dovrà essere incluso in tutti gli script php che effettueranno l'accesso al database.
Come prima cosa, il file config.php include la dbLibrary per poter usare la funzione openDB:

```
<?php
include('../libraries/dbLibrary.php');
?>
```

Abbiamo aggiunto al file dbLibrary.php il percorso per accedere al file. Il file config.php si trova nella directory api. Per accedere al file dbLibrary.php, dobbiamo tornare indietro di una directory e poi entrare nella cartella libraries. Il simbolo .. permette di tornare indietro di una directory, mentre lo / permette di accedere alla directory che segue. In ambiente Windows, occorre sostituire il carattere / con il carattere \:

```
include('..\libraries\dbLibrary.php');
```

A questo punto possiamo invocare la funzione openDB della dbLibrary, passando dei parametri che abbiamo settato in precedenza attraverso delle variabili:

```
$database = "Esercitazione";
$username = "root";
$password = "";
$servername = "localhost";
$db = openDB($database, $password, $username, $servername);
```

La connessione al database è ora aperta ed è memorizzata nella variabile $db. Il file completo è il seguente:

```
<?php
include('../libraries/dbLibrary.php');

$database = "Esercitazione";
$username = "root";
$password = "";
$servername = "localhost";
$db = openDB($database, $password, $username, $servername);

?>
```

Ricordiamoci che sotto Windows, dobbiamo sostituire il carattere / con \.
Ora possiamo passare all'implementazione vera e propria della nostra API. Come prima cosa, l'esercizio chiede di implementare una API che per ogni comune mostra le coordinate geografiche. Nella cartella API creiamo un file comuni.php che restituirà per ogni comune le coordinate geografiche. Come prima cosa, includiamo il file config.php che ci permette di aprire la connessione al database:

```
<?php
include('config.php');
?>
```

Visto che il file config.php si trova nella stessa cartella del file comuni.php, non c'è bisogno di aggiungere altri percorsi al file. Non c'è bisogno di includere la dbLibrary perché è già inclusa nel file config.php
Una volta aperta la connessione, ricordiamoci subito di chiuderla in fondo, altrimenti poi potremmo dimenticarcene:

```
<?php
include('config.php');
// corpo di comuni.php
closeDB($db);
?>
```

Ora, dobbiamo sostituire il commento // corpo di comuni.php con il corpo vero e proprio della api. Ricordiamoci di lasciare la closeDB in fondo allo script. Supponiamo che nel nostro database Esercitazione, abbiamo memorizzato la tabella con il nome ComuniTerremotati. Dalla tabella dobbiamo estrarre il nome del comune e le coordinate geografiche (latitudine e longitudine). La query che dobbiamo fare è la seguente:

```
SELECT Comune, latitude, longitude FROM ComuniTerremotati
```

Possiamo passare questa query alla funzione select della dbLibrary. Stampiamo anche il risultato tramite la funzione var_dump, che può essere utilizzata per stampare un array:

```
$sql = "SELECT Comune, latitude, longitude FROM ComuniTerremotati";
$recs = select($db, $sql);
var_dump($recs);
```

Il codice completo dello script comuni.php ottenuto fino ad ora è il seguente:

```
<?php
include('config.php');
$sql = "SELECT Comune, latitude, longitude FROM ComuniTerremotati";
$recs = select($db, $sql);
var_dump($recs);

closeDB($db);
?>
```

Guardando la stampa dell'array, si può notare che la latitudine e la longitudine sono stampati come stringhe. Questo non va bene, perché in realtà si tratta di decimali. Occorre quindi sistemare il codice in modo da convertire le stringhe in float. Per fare questo, dobbiamo scorrere l'array $recs, cercare le chiavi latitude e longitude e convertire il valore corrispondente in float.
L'array $recs è un array numerico, quindi, per accedere ai suoi elementi, possiamo usare un ciclo for:

```
for($i = 0; $i < count($recs); $i++)
{
      // corpo del for
}
```

Ogni campo dell'array $recs è a sua volta un array, questa volta di tipo associativo. Quindi per scorrere ogni elemento dell'array $recs possiamo usare un ciclo foreach:

```
for($i = 0; $i < count($recs); $i++)
{
      foreach($recs[$i] as $chiave => $valore)
      {
            // corpo del foreach
      }
}
```

Se il campo chiave coincide con latitude o longitude, occorre fare la conversione di valore a float, altrimenti non c'è bisogno di fare niente:

```
for($i = 0; $i < count($recs); $i++)
{
      foreach($recs[$i] as $chiave => $valore)
      {
            if(($chiave == 'latitude') || ($chiave == 'longitude'))
                  $recs[$i][$chiave] = floatval($valore);
      }
}
```

Da notare che il campo che vogliamo modificare si trova nella posizione i-esima dell'array $recs in corrispondenza della chiave che stiamo analizzando. Quindi dobbiamo sostituire al valore corrente di $recs[$i][$chiave] lo stesso valore convertito però in float.

Per rendere i dati accessibili a tutti i tipi di client, la nostra API dovrebbe fornire i risultati in JSON. JSON (JavaScript Object Notation) è un formato utilizzato per lo scambio dei dati in applicazioni client-server. Il JSON è un formato che serve per codificare un array in una stringa. In JSON un array numerico è rappresentato come un elenco di elementi separati da virgola e racchiusi tra parentesi quadre. Ad esempio:

['cane', 'gatto', 'topo']

Un array associativo invece è rappresentato da coppie chiave valore separate dalla virgola e racchiuse tra parentesi graffe. Ogni coppia chiave valore è rappresentata nel seguente modo:

chiave : valore

Un esempio di array associativo è il seguente:

{'animale' : 'cane', 'fiore' : 'rosa', 'frutto' : 'mela'}

In PHP, affinché una API ritorni un array in JSON è necessario eseguire due passi:

1. includere all'inizio dell'API la chiamata ad una funzione che setta il formato del risultato in json

```
header('Content-Type: application/json');
```

2. codificare l'array risultato in json attraverso la funzione json_encode e stampare il risultato:

```
echo json_encode($recs);
```

Nel nostro caso, togliamo la stampa fatta con la var_dump e inseriamo il codice appena definito, facendo attenzione ad inserire la chiamata alla header all'inizio del documento e la json_encode al posto della var_dump. Il risultato è il seguente:

```
<?php
```

```
header('Content-Type: application/json');
include('config.php');
$sql = "SELECT Comune, latitude, longitude FROM ComuniTerremotati";
$recs = select($db, $sql);
for($i = 0; $i < count($recs); $i++)
{
      foreach($recs[$i] as $chiave => $valore)
      {
            if(($chiave == 'latitude') || ($chiave == 'longitude'))
                  $recs[$i][$chiave] = floatval($valore);
      }
}

echo json_encode($recs);

closeDB($db);
?>
```

Visualizzando il risultato dal browser, notiamo che non è molto chiaro. Per visualizzare meglio un json, possiamo installare un plugin per il nostro browser. Il plugin si chiama JSON View.

Per installare il plugin su Google Chrome, andiamo nel menu Impostazioni, selezioniamo estensioni, poi prova altre estensioni. Nella barra di ricerca digitiamo jsonview e poi installiamolo.

Per installare il plugin su Safari, selezionare il menu Safari, poi estensioni, cercare JSON ed installare json-lite.

Per installare il plugin su Firefox, dal menu in alto a destra selezionare componenti aggiuntivi, quindi estensioni e cercare jsonview.

A questo punto possiamo passare alla seconda parte dell'esercizio, che richiede la creazione di una API che mostri per ogni provincia la popolazione prima del sisma. Per prima cosa rinominiamo il campo della tabella 'Popolazione prima del sisma' in popolazione, per evitare problemi dovuti agli spazi. Per fare questo, usiamo Phpmyadmin, selezioniamo la tabella e quindi in corrispondenza del campo 'Popolazione prima del sisma' selezionare modifica e cambiare il nome in Popolazione. Premere il tasto Salva.

Ora scriviamo la query, che deve fare la somma di tutte le popolazioni prima del sisma per ogni provincia. Quindi, raggruppiamo i record per provincia e facciamo la somma del campo popolazione, usando l'istruzione GROUP BY:

```
SELECT SUM(Popolazione) AS Popolazione, Provincia FROM ComuniTerremotati GROUP BY Provincia
```

Dentro la cartella api creiamo un file popolazione.php che mostrerà il risultato della nostra query. Come per il file comuni.php, includiamo il file config.php e impostiamo come json il tipo di risultato :

```
<?php
header('Content-Type: application/json');
include('config.php');
// corpo di popolazione.php
closeDB($db);
?>
```

Ora possiamo eseguire la query:

```
$sql = "SELECT SUM(Popolazione) AS Popolazione, Provincia FROM ComuniTerremotati GROUP BY Provincia";
$recs = select($db, $sql);
```

e stampare il risultato:

```
echo json_encode($recs);
```

Come nel caso della latitudine e della longitudine per l'API comuni.php, anche in popolazione.php il campo popolazione deve essere un intero. Pertanto va fatta la conversione da stringa a intero attraverso la funzione intval. Copiamo il codice scritto per comuni.php, avendo l'accortezza di modificare latitudine e longitudine con popolazione e la funzione floatval con intval:

```
for($i = 0; $i < count($recs); $i++)
{
      foreach($recs[$i] as $chiave => $valore)
      {
            if(($chiave == 'Popolazione'))
                  $recs[$i][$chiave] = intval($valore);
      }
}
```

Il codice completo dello script popolazione.php è il seguente:

```
<?php
header('Content-Type: application/json');
include('config.php');
$sql = "SELECT SUM(Popolazione), Provincia FROM ComuniTerremotati GROUP BY Provincia";
$recs = select($db, $sql);
for($i = 0; $i < count($recs); $i++)
{
      foreach($recs[$i] as $chiave => $valore)
      {
            if(($chiave == 'Popolazione'))
                  $recs[$i][$chiave] = intval($valore);
      }
}
echo json_encode($recs);
```

```
closeDB($db);
?>
```

Esercizio 2

Modificare il file comuni.php in modo da ricevere in ingresso tramite GET un parametro type. In base al valore del campo type, il file restituisce un risultato diverso:

- se type = coordinate, lo script restituisce le coordinate geografiche
- se type = url, lo script restituisce l'url del comune