AJAX i PHP. Ćwiczenia praktyczne

Autor: Marcin Lis
ISBN: 978-83-246-1176-8
Format: A5, stron: 160
Przykłady na ftp: 46 kB

Poznaj AJAX-a i twórz ergonomiczne witryny WWW!

- Transmisja danych w technologii AJAX
- Obsługa formularzy i danych
- Przetwarzanie plików XML

AJAX zrewolucjonizował witryny WWW. Połączenie języków JavaScript i XML z możliwością przesyłania danych w tle sprawiło, że strony internetowe nie muszą być przeładowywane po każdym kliknięciu łącza lub zmianie zawartości. Dzięki temu witryny i aplikacje internetowe coraz bardziej przypominają swoim działaniem normalne programy, a korzystanie z nich stało się zdecydowanie wygodniejsze. Jedną z najistotniejszych zalet AJAX-a jest to, że wykorzystuje on znane i sprawdzone rozwiązania, co sprawia, że użytkownicy przeglądarek internetowych nie są zmuszani do instalowania dodatkowych kontrolek, pluginów i innych modułów.

„AJAX i PHP. Ćwiczenia praktyczne" to zbiór przykładów, dzięki którym poznasz możliwości modelu AJAX w połączeniu ze skryptami PHP. Wykonując kolejne ćwiczenia zawarte w książce, dowiesz się, jak działają witryny WWW oparte na tym modelu, czym są żądania asynchroniczne oraz obiekt XMLHttpRequest. Nauczysz się tworzyć skrypty PHP tak, aby przesyłały i odbierały dane w sposób zgodny z wymaganiami AJAX-a. Poznasz metody weryfikacji formularzy, komunikacji z bazami danych i przetwarzania plików XML.

- Transmisja danych w modelu AJAX
- Obiektowy Model Dokumentu (DOM)
- Wysyłanie danych za pomocą metod GET i POST
- Przetwarzanie danych w skryptach PHP
- Walidacja danych z formularzy
- Generowanie kodu po stronie serwera
- AJAX, PHP i SQL
- Praca z plikami XML

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

AJAX, PHP i bazy danych

Tworzenie bazy danych

Każdy większy serwis internetowy korzysta z bazy danych. Do komunikacji z systemem bazodanowym z powodzeniem używa się skryptów PHP. W tym rozdziale przyjrzymy się więc, jak może współpracować AJAX, PHP i baza danych. Skorzystamy z produktu o nazwie SQLite, którego obsługa jest wbudowana bezpośrednio w PHP, począwszy od PHP5. Nie będziemy jednak omawiać zasad tworzenia baz danych, a także języka zapytań SQL, gdyż temu celowi służą inne publikacje, np. *SQL Ćwiczenia praktyczne* (*http://helion.pl/ksiazki/cwsql2.htm*)[1].

Aby PHP mogło współpracować z SQLite w systemie Windows, w pliku konfiguracyjnym *php.ini* należy dodać dwie linie (dokładnie w przedstawionej kolejności):

```
extension=php_pdo.dll
extension=php_sqlite.dll
```

[1] Tematyka współpracy PHP z bazami danych została również wyczerpująco omówiona m.in. w publikacjach *PHP i MySQL Dla każdego (http://helion.pl/ksiazki/phsqdk.htm* oraz *PHP5. Praktyczny kurs, http://helion.pl/ksiazki/php5pk.htm)*.

Użytkownicy Linuksa powinni natomiast skorzystać z instalatora PEAR, wydając na konsoli polecenie:

```
pear install sqlite
```

Do zarządzania bazą, np. przygotowania danych, można natomiast użyć działającego w wierszu poleceń klienta `sqlite`. Jest on dostępny w internecie pod adresem *http://www.sqlite.org*.

ĆWICZENIE

5.1 Uruchamianie SQLite i tworzenie bazy danych

Uruchom klient SQLite w wybranym systemie i utwórz nową bazę danych.

Aby uruchomić klienta, należy go skopiować do wybranego katalogu oraz wydać polecenie:

```
sqlite nazwa_pliku;
```

gdzie *nazwa_pliku* to określenie pliku z bazą danych. Jeśli pliku określonego przez *nazwa_pliku* nie będzie na dysku, zostanie on utworzony — powstanie w ten sposób nowa baza. Po uruchomieniu klienta (rysunek 5.1) można w nim wykonywać polecenia i zapytania SQL.

Rysunek 5.1. *Uruchomienie klienta SQLite*

Dane w bazie są przechowywane w tabelach; do ich tworzenia służy instrukcja SQL `CREATE TABLE` o ogólnej postaci:

```
CREATE TABLE nazwa_tabeli
(
  nazwa_kolumny_1 typ_kolumny_1 [atrybuty],
  nazwa_kolumny_2 typ_kolumny_2 [atrybuty],
...
  nazwa_kolumny_n typ_kolumny_n [atrybuty],
)
```

Dane do tabel wprowadza się za pomocą instrukcji `INSERT INTO`. Jej podstawowa, najczęściej wykorzystywana postać, jest następująca:

```
INSERT [INTO] tabela [(kolumna1, kolumna2, ..., kolumnaN)] VALUES
(wartość1, wartość2, ..., wartośćN)
```

Powoduje ona wprowadzenie do tabeli nowego wiersza, w którym w polu *kolumna1* została zapisana wartość *wartość1*, w polu *kolumna2* — wartość *wartość2* itd.

Powróćmy teraz do ćwiczeń 3.7 i 3.8 z rozdziału 3. Polegały one na pobraniu z serwera danych, które miały się pojawić jako opcje na liście rozwijanej. Opcje te były generowane przez skrypt PHP i zapisane w zwykłej tablicy. Spróbujmy więc wykonać to zadanie nieco inaczej, niech zawartość listy będzie pobierana z bazy danych. Trzeba więc przygotować odpowiednią tabelę i wypełnić ją danymi.

ĆWICZENIE

5.2 Tworzenie tabeli w bazie danych

Utwórz tabelę przechowującą dane dla listy rozwijanej i wypełnij ją przykładowymi danymi.

```
CREATE TABLE Opcje (
  ZESTAW_ID INTEGER,
  NAZWA VARCHAR(40)
);

INSERT INTO Opcje VALUES (1, "Java. Ćwiczenia praktyczne");
INSERT INTO Opcje VALUES (1, "Java. Ćwiczenia zaawansowane");
INSERT INTO Opcje VALUES (1, "Praktyczny kurs Java");

INSERT INTO Opcje VALUES (2, "PHP. 101 praktycznych skryptów");
INSERT INTO Opcje VALUES (2, "PHP i MySQL. Dla każdego");
INSERT INTO Opcje VALUES (2, "PHP5. Praktyczny kurs");
```

Tabela została utworzona za pomocą instrukcji `CREATE TABLE` i zawiera dwie kolumny. Pierwsza, o nazwie `ZESTAW_ID`, przechowuje liczby całkowite (typ danych `INTEGER`) określające, do jakiego zestawu należy dana opcja (jak pamiętamy z ćwiczenia 3.7, do dyspozycji będziemy mieli dwa zestawy danych), a druga, o nazwie `NAZWA` — ciągi maksymalnie 40 znaków (typ `VARCHAR(40)`) określające tekst danej opcji. Do tej tabeli za pomocą instrukcji `INSERT INTO` zostały wprowadzone dwa zestawy danych (o identyfikatorach `1` i `2`), każdy składający się z trzech pozycji.

Dane zapisane w tabelach bazy można pobierać za pomocą instrukcji SELECT. Co prawda, posiada ona wiele opcji i klauzul dodatkowych, jednak dla naszych potrzeb wystarczająca jest jej podstawową postać, która schematycznie wygląda następująco:

```
SELECT kolumna1, kolumna2, ..., kolumnaN
FROM tabela
[WHERE warunek]
[ORDER BY kolumna1, kolumna2, ..., kolumnaN [ASC | DEC]]
```

Oznacza ona: pobierz wartości wymienionych kolumn z tabeli *tabela* spełniających warunek *warunek*, a wyniki posortuj względem kolumn wymienionych w klauzuli ORDER BY rosnąco (ASC) lub malejąco (DESC).

ĆWICZENIE

5.3 Pobieranie danych z tabeli

Napisz instrukcję SQL pobierającą wszystkie dane z tabeli Opcje z ćwiczenia 5.2.

Zapytanie pobierające wszystkie dane z tabeli Opcje będzie miało postać:

```
SELECT ZESTAW_ID, NAZWA FROM Opcje;
```

ĆWICZENIE

5.4 Pobieranie danych z tabeli spełniających zadany warunek

Napisz instrukcję SQL pobierającą z tabeli Opcje wszystkie wiersze, dla których wartość w kolumnie ZESTAW_ID jest równa 2.

Zapytanie wykonujące postawione zadanie jest następujące:

```
SELECT ZESTAW_ID, NAZWA FROM Opcje WHERE ZESTAW_ID = 2;
```

Komunikacja z bazą danych

Skrypt PHP korzystający z bazy SQLite musi się z nią w jakiś sposób komunikować. Możemy tu wyróżnić trzy główne czynności:

- nawiązanie połączenia z bazą,
- wykonanie zapytania i odebranie wyników,
- zamknięcie połączenia.

Do nawiązania połączenia służy funkcja `sqlite_open`. Ma ona schematyczną postać:

```
sqlite_open("plik"[, "tryb"[, $msg]])
```

Parametr *plik* określa nazwę pliku, w którym znajduje się baza danych lub w którym ma zostać utworzona. *tryb* w założeniach określa atrybuty pliku, jednak obecnie jest on ignorowany i zaleca się wykorzystywanie wartości `0666`. Natomiast *$msg* to przekazana przez referencję zmienna, w której zostanie zapisany opis błędu, o ile taki wystąpi. Funkcja zwraca identyfikator nawiązanego połączenia, jeśli udało się je nawiązać, lub wartość `false` w przeciwnym przypadku. Jeśli zatem chcielibyśmy nawiązać połączenie z bazą zawartą w pliku */var/www/bazy/testphp.db*, możemy zastosować wywołanie:

```
sqlite_open("/var/www/bazy/testphp.db")
```

lub też:

```
sqlite_open("/var/www/bazy/testphp.db", 0666, $msg)
```

W tym drugim przypadku, jeśli wystąpi błąd (np. na dysku nie będzie katalogu */var/www/bazy/*), będziemy mogli uzyskać bardziej szczegółową informację o jego przyczynie.

Połączenie otwarte za pomocą `sqlite_open` po wykonaniu żądanych operacji na bazie powinno zostać zamknięte przez wywołanie `sqlite_close`. Schematyczne wywołanie tej funkcji ma postać:

```
sqlite_close(identyfikator)
```

gdzie *identyfikator* to identyfikator połączenia, które ma zostać zamknięte, zwrócony wcześniej przez funkcję `sqlite_open`. Funkcja `sqlite_close` zwraca wartość `true`, jeżeli operacja przez nią wykonywana zakończyła się sukcesem, lub `false` w przeciwnym przypadku.

Zapytania pobierające dane najprościej wykonywać za pomocą funkcji `sqlite_array_query`, która jednocześnie wysyła zapytanie do bazy oraz odbiera wyniki w postaci tablicy zawierającej odczytane rekordy[2].

[2] Funkcji tej nie należy jednak stosować do wykonywania zapytań zwracających większą liczbę wyników (dokumentacja PHP określa graniczną liczbę na 45 wierszy). W przypadku zapytań zwracających wiele wierszy danych, powinno się stosować funkcję sqlite_unbuffered_query wraz z jedną z funkcji typu sqlite_fetch_*. Bliższe informacje na ten temat można zaleźć w dokumentacji PHP oraz w publikacjach wymienionych w pierwszym przypisie.

Do dyspozycji mamy dwie postacie różniące się od siebie kolejnością argumentów:

```
sqlite_array_query(identyfikator, zapytanie[, typ_wyniku])
sqlite_array_query (zapytanie, identyfikator[, typ_wyniku])
```

Znacznie argumentów jest następujące:

identyfikator — identyfikator połączenia z bazą danych zwrócony przez funkcję `sqlite_open`;

zapytanie — zapytanie SQL wysyłane do bazy;

typ_wyniku — określa, w jaki sposób będzie indeksowana tablica zawierająca wynik zapytania; dopuszczalne wartości to: `SQLITE_ASSOC` — indeksowanie asocjacyjne, `SQLITE_NUM` — indeksowanie numeryczne; `SQLITE_BOTH` — oba typy indeksowania; jeśli argument ten zostanie pominięty, zastosowana będzie wartość `SQLITE_BOTH`.

Wynikiem działania `sqlite_array_query` jest tablica zawierająca tablice, z których każda zawiera jeden wiersz będący wynikiem zadanego zapytania.

Sprawdźmy więc, jak za pomocą skryptu PHP pobrać dane z bazy. Obrazuje to kod ćwiczenia 5.5.

ĆWICZENIE

5.5 Komunikacja z bazą danych

Napisz skrypt PHP, który odczyta całą zawartość tabeli `Opcje` i wyświetli ją na ekranie.

```
<?php
if (!$db_lnk = sqlite_open("./baza.sqlite", 0666, $msg)){
  echo('error:Wystąpił błąd podczas próby połączenia z bazą danych.');
  exit;
}

$query = "SELECT ZESTAW_ID, NAZWA FROM Opcje";

if(!$arr = sqlite_array_query($db_lnk, $query)){
  sqlite_close($db_lnk);
  echo('error:Wystąpił błąd: nieprawidłowe zapytanie...');
  exit;
}
```

```
foreach($arr as $row){
 echo "$row[0] : $row[1]\n";
}
sqlite_close($db_lnk);
?>
```

Ten skrypt to połączenie elementów opisanych na wcześniejszych stronach. Połączenie z bazą danych jest otwierane za pomocą funkcji `sqlite_open`, a jego identyfikator zapisywany w zmiennej `$db_lnk`. W przypadku gdyby czynność ta nie zakończyła się sukcesem, wyświetlana jest informacja o błędzie, a skrypt kończy działanie. Jeśli połączenie udało się utworzyć, zmiennej `$query` przypisywana jest treść zapytania, które jest takie samo jak w przypadku ćwiczenia 5.3. Zmienne `$db_lnk` oraz `$query` są następnie używane w wywołaniu funkcji `sqlite_array_query`, która wysyła zapytanie do bazy i pobiera jego wyniki w postaci tablicy. Wynik działania tej funkcji jest przypisywany zmiennej `$arr`. A zatem jeśli zapytanie udało się wykonać, zmienna `$arr` będzie zawierała tablicę wynikową, a jeśli nie — wartość `false`.

Wyniki zapytania przetwarzane są w pętli `foreach`. W każdym jej przebiegu zmienna `$row` otrzymuje wartość kolejnego klucza (komórki) tabeli `$arr`, a więc zawiera tablicę przechowującą kolejny wiersz danych. To oznacza, że pod indeksem `0` (`$row[0]`) zapisana jest wartość kolumny `ZESTAW_ID`, a pod indeksem `1` (`$row[1]`) wartość kolumny `NAZWA` z tabeli `Opcje`. Zatem każda instrukcja:

```
echo "$row[0] : $row[1]\n";
```

powoduje wyświetlenie kolejnego wiersza wyników na ekranie.

Po zakończeniu pętli `foreach` połączenie z bazą jest zamykane za pomocą funkcji `sqlite_close`.

Powróćmy teraz do wypełniania danymi listy rozwijanej. W ćwiczeniu 5.2 przygotowaliśmy odpowiednią tabelę i wypełniliśmy ją danymi — były to dwa zestawy opcji. Zatem należy teraz napisać skrypt PHP, który na podstawie przekazanego mu parametru pobierze dane z wybranego zestawu i wyśle je do przeglądarki. Format wysyłanych danych będzie taki jak w przypadku ćwiczenia 3.7, czyli poszczególne opcje będą od siebie oddzielone znakiem dwukropka.

ĆWICZENIE

5.6 Uzależnienie wyników zapytania od zewnętrznego parametru

Napisz skrypt PHP, który na podstawie przekazanego mu parametru pobierze właściwy zestaw opcji z tabeli `Opcje` i wyśle go w odpowiednim formacie do przeglądarki.

```
<?php
if(isSet($_GET['zestaw'])){
  $id = intval($_GET['zestaw']);
  if($id != 1 && $id != 2){
    echo "error:Nieprawidłowy parametr.";
    exit;
  }
  if (!$db_lnk = @sqlite_open("./baza.sqlite", 0666, $msg)){
    echo('error:Wystąpił błąd podczas próby połączenia z bazą danych.');
    exit;
  }

  $query = "SELECT nazwa FROM opcje WHERE ZESTAW_ID=$id";

  if(!$arr = sqlite_array_query($db_lnk, $query)){
    @sqlite_close($db_lnk);
    echo('error:Wystąpił błąd: nieprawidłowe zapytanie...');
    exit;
  }

  $str = "";
  foreach($arr as $row){
   $str .= $row[0] . ":";
  }
  echo $str;

  @sqlite_close($db_lnk);
}
else{
  echo "error:Nieprawidłowe wywołanie skryptu.";
  exit;
}
?>
```

Kod rozpoczyna się od sprawdzenia, czy w tablicy `$_GET` znajduje się klucz o nazwie `zestaw`, czyli czy do skryptu został przekazany parametr o nazwie `zestaw`. Jeśli nie, generowana jest jedynie informacja o błędzie,

jeśli tak, otrzymany ciąg jest podawany działaniu funkcji `intval`, czyli przekształcany na wartość całkowitą. Wynik przekształcenia zapisywany jest w zmiennej `$id`:

```
$id = intval($_GET['zestaw']);
```

Ponieważ wiemy, że jedyne prawidłowe wartości parametru to `1` i `2`, badamy następnie, czy zmienna `$id` zawiera jedną z nich:

```
if($id != 1 && $id != 2){
```

Jeśli nie, przeglądarka otrzymuje komunikat o błędzie. Jeżeli dane są prawidłowe, konstruowane jest zapytanie SQL przypisywane następnie zmiennej `$query`:

```
$query = "SELECT nazwa FROM opcje WHERE ZESTAW_ID=$id";
```

Zapytanie to ma postać taką jak w ćwiczeniu 5.4, z tą różnicą że warunek `WHERE` jest tworzony na podstawie wartości zmiennej `$id`.

Wykonanie zapytania oraz odebranie wyników zapewnia funkcja `sqlite_array_query`. Tablica wynikowa jest odczytywana w pętli `foreach`, podobnie jak miało to miejsce w przypadku ćwiczenia 5.5. W każdym przebiegu tej pętli pod `$row[0]` podstawiana jest wartość kolejnej opcji pobranej z tabeli `Opcje`. Wartość ta jest dodawana do ciągu `$str` wraz ze znakiem dwukropka. Dzięki temu ciąg wynikowy, wysyłany do przeglądarki za pomocą instrukcji `echo`, będzie miał taki sam format jak w przypadku ćwiczenia 3.7.

Do wykonania pozostał jeszcze strona WWW współpracująca ze skryptem PHP z ćwiczenia 5.6.

ĆWICZENIE

5.7 Umieszczanie wyników zapytania na stronie WWW

Napisz kod strony WWW wysyłającej dane sterujące do skryptu PHP z ćwiczenia 5.6 i odbierającej dane wynikowe.

```
XMLHttpRequestObject.onreadystatechange = function()
{
  if (XMLHttpRequestObject.readyState == 4){
    if(XMLHttpRequestObject.status == 200){
      var tekst = XMLHttpRequestObject.responseText;
      var arr = tekst.split(':');
      if(arr[0] != 'error'){
```

```
            var listaOpcji = document.getElementById("listaOpcji");
            listaOpcji.options.length = 0;
            for(i = 0; i < arr.length; i++){
              listaOpcji[i] = new Option(arr[i], arr[i]);
            }
          }
          else{
            if(arr.length == 2){
              alert(arr[1]);
            }
            else{
              alert("Wystąpił błąd podczas przetwarzania danych.");
            }
          }
        }
        przycisk1El.disabled = false;
        przycisk2El.disabled = false;
      }
```

Kod tego ćwiczenia będzie prawie identyczny jak ten przedstawiony w ćwiczeniu 3.8. Niewielkim zmianom uległa jedynie widoczna wyżej funkcja anonimowa przypisana właściwości `onreadystatechange` obiektu `XMLHttpRequestObject`. W inny sposób obsługuje ona sytuacje błędne. Otóż tym razem, w przypadku stwierdzenia, że pierwszym wyrazem ciągu wynikowego zapisanego w tablicy `arr` jest ciąg `error`, przyjmujemy, że drugi wyraz zawiera komunikat o błędzie, który ma zostać wyświetlony użytkownikowi i to jest cała zawartość tej tablicy. Badane jest zatem, czy długość `arr` równa się `2`. Jeśli tak, tekst z drugiej komórki tej tabeli jest wyświetlany w oknie dialogowym:

```
alert(arr[1]);
```

Jeśli nie, na ekranie pojawi się standardowy komunikat o błędzie:

```
alert("Wystąpił błąd podczas przetwarzania danych.");
```

Pobieranie treści witryny z bazy danych

W ćwiczeniach 5.5, 5.6 i 5.7 powstał system generowania danych dla formularza (listy rozwijanej) znajdującego się na stronie WWW. Teraz postarajmy się wykonać przykład, w którym w bazie danych przechowywana jest sama treść prezentowana na witrynie. Użytkownik,

klikając przyciski, będzie decydował o tym, co ma zostać wyświetlone; oczywiście transmisja danych będzie się odbywała w tle. W przeciwieństwie jednak do przykładów z poprzedniego podrozdziału tym razem część kodu witryny będzie generowana na podstawie danych z bazy.

Naszym zadaniem będzie prezentowanie tytułów książek należących do poszczególnych kategorii tematycznych. Na stronie znajdą się przyciski z nazwami kategorii, a poniżej prezentowane będą poszczególne tytuły. Pracę zaczniemy od przygotowania tabel bazy danych i wypełnienia ich danymi. To zadanie realizowane jest w ćwiczeniu 5.8.

ĆWICZENIE

5.8 Relacje między tabelami bazy danych

Utwórz i wypełnij przykładowymi danymi tabele bazy danych, umożliwiające przechowywanie danych o tytułach książek, z podziałem na kategorie tematyczne.

```
CREATE TABLE Kategorie (
  ID INTEGER PRIMARY KEY,
  NAZWA VARCHAR(40)
);

CREATE TABLE Tytuly (
  ID INTEGER PRIMARY KEY,
  KATEGORIA_ID INTEGER,
  NAZWA VARCHAR(40)
);

INSERT INTO Kategorie VALUES (1, "Java");
INSERT INTO Kategorie VALUES (2, "JavaScript");
INSERT INTO Kategorie VALUES (3, "PHP");
INSERT INTO Kategorie VALUES (4, "AJAX");

INSERT INTO Tytuly VALUES (NULL, 1, "Java. Ćwiczenia praktyczne");
INSERT INTO Tytuly VALUES (NULL, 1, "Java. Ćwiczenia zaawansowane");
INSERT INTO Tytuly VALUES (NULL, 1, "Praktyczny kurs Java");

INSERT INTO Tytuly VALUES (NULL, 2, "JavaScript. Ćwiczenia praktyczne");
INSERT INTO Tytuly VALUES (NULL, 2, "Tablice informatyczne.
JavaScript");
INSERT INTO Tytuly VALUES (NULL, 2, "101 praktycznych skryptów na stronę
WWW");
```

```
INSERT INTO Tytuly VALUES (NULL, 3, "PHP. 101 praktycznych skryptów");
INSERT INTO Tytuly VALUES (NULL, 3, "PHP i MySQL. Dla każdego");
INSERT INTO Tytuly VALUES (NULL, 3, "PHP5. Praktyczny kurs");

INSERT INTO Tytuly VALUES (NULL, 4, "AJAX. Ćwiczenia");
INSERT INTO Tytuly VALUES (NULL, 4, "AJAX. 101 praktycznych skryptów");
INSERT INTO Tytuly VALUES (NULL, 4, "AJAX. Praktyczny kurs");
```

Powstały tu dwie tabele. Pierwsza przechowuje identyfikatory i nazwy kategorii, a druga dane dotyczące tytułów. Tabela `Kategorie` składa się z dwóch kolumn:

- `ID` — typu `INTEGER`, będąca kluczem podstawowym (atrybut `PRIMARY KEY`), przechowująca identyfikatory;
- `NAZWA` — typu `VARCHAR(40)`, przechowująca nazwy kategorii.

Tabela `Tytuly` zawiera trzy kolumny:

- `ID` — typu `INTEGER`, będąca kluczem podstawowym (atrybut `PRIMARY KEY`), przechowująca identyfikatory tytułów;
- `KATEGORIA_ID` — typu `INTEGER`, przechowująca identyfikator kategorii, do której należy dany tytuł;
- `NAZWA` — typu `VARCHAR(40)`, przechowująca same tytuły.

Za pomocą instrukcji `INSERT INTO` do tabeli `Kategorie` zostały wprowadzone 4 kategorie, a do tabeli `Tytuly` 12 tytułów, po trzy w każdej kategorii. Ponieważ w bazie SQLite pole typu `INTEGER` z atrybutem `PRIMARY KEY` jest jednocześnie polem autoinkrementacyjnym, ta właściwość została wykorzystana przy wprowadzaniu danych dotyczących tytułów, tak by ich identyfikatory zostały wygenerowane automatycznie.

Dane z tabel utworzonych w ćwiczeniu 5.8 będą pobierane przez skrypt PHP na podstawie przekazanego mu ze strony WWW parametru. Będzie to parametr o nazwie `kategoria` przesyłany za pomocą metody `GET`. Napiszmy więc teraz kod takiego skryptu.

ĆWICZENIE

5.9 Pobieranie z bazy danych należących do określonej kategorii

Napisz kod skryptu PHP, który na podstawie przekazanego mu parametru pobierze z bazy i wyśle do przeglądarki nazwy tytułów należących do określonej kategorii tematycznej.

```
<?php
if(isSet($_GET['kategoria'])){
  $id = intval($_GET['kategoria']);

  if (!$db_lnk = @sqlite_open("./baza.sqlite", 0666, $msg)){
    echo('error:Wystąpił błąd podczas próby połączenia z bazą danych.');
    exit;
  }

  $query = "SELECT NAZWA FROM Tytuly WHERE KATEGORIA_ID=$id";

  if(!$arr = sqlite_array_query($db_lnk, $query)){
    @sqlite_close($db_lnk);
    echo('error:Wystąpił błąd: nieprawidłowe zapytanie...');
    exit;
  }
?>

<table border="0">

<?php
  $str = "";
  foreach($arr as $row){
   echo "<tr><td>$row[0]</td></tr>";
  }
?>

</table>

<?

  @sqlite_close($db_lnk);
}
else{
  echo "error:Nieprawidłowe wywołanie skryptu.";
  exit;
}
?>
```

Ponieważ skrypt ma za pomocą metody `GET` otrzymać parametr o nazwie `kategoria`, kod zaczyna się typowo od sprawdzenia, czy w tablicy `$_GET` znajduje się klucz o takiej nazwie. Jeśli istnieje, są wykonywane dalsze czynności, a jeśli nie, skrypt kończy działanie, wysyłając do przeglądarki informację o błędzie. Wartość parametru konwertowana jest do typu całkowitoliczbowego i zapisywana w zmiennej `$id`.

Połączenie z bazą danych otwierane jest, tak jak we wcześniej prezentowanych przykładach, za pomocą funkcji `sqlite_open`, a zapytanie wykonywane za pomocą funkcji `sqlite_array_query`. Wynik działania jest przypisywany zmiennej `$arr`. Zapytanie SQL ma postać:

```
SELECT NAZWA FROM Tytuly WHERE KATEGORIA_ID=$id
```

a zatem jest to pobranie z tabeli `Tytuly` takich wierszy, które w kolumnie `KATEGORIA_ID` zawierają taką wartość, jaką ma zmienna `$id`.

Pobrane dane są formowane w typową tabelę HTML o jednej kolumnie i wysyłane do przeglądarki. Każdy wiersz tworzony jest w pętli `foreach` przeglądającej tablicę `$arr`. Schematyczna konstrukcja pojedynczego wiersza to:

```
<tr>
  <td>tytuł</td>
</tr>
```

a więc za jego wygenerowanie odpowiada instrukcja:

```
echo "<tr><td>$row[0]</td></tr>";
```

Skoro zarówno baza danych, jak i skrypt PHP ją obsługujący są gotowe, pozostało nam utworzenie samej strony WWW. Odmiennie jednak od przykładu z ćwiczenia 5.8, w którym dane sterujące (wartości parametru przekazywanego do skryptu PHP) wpisane były bezpośrednio w kodzie HTML, tym razem będą one również pobierane z bazy danych. Dzięki temu liczba, kolejność oraz nazwy kategorii będą mogły być dowolnie zmieniane, a kod witryny nie będzie wymagał przy tym żadnych modyfikacji.

ĆWICZENIE

5.10 Generowanie treści strony na podstawie danych z bazy

Napisz kod witryny współpracującej ze skryptem PHP z ćwiczenia 5.9 oraz bazą danych z ćwiczenia 5.8.

```
<?php
function generujPrzyciski()
{
  $db_lnk = @sqlite_open("./baza.sqlite", 0666, $msg);
  $query = "SELECT id, nazwa FROM kategorie";
  $arr = @sqlite_array_query($db_lnk, $query);
  @sqlite_close();
```

```
  if($db_lnk && $arr){
    foreach($arr as $row){
      echo "<input type='button' value='{$row[1]}' ";
      echo "onclick='wyslijDane({$row[0]});' ";
      echo "style='width:100px;'/>  ";
    }
  }
}
?>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Ajax</title>
<script type="text/javascript">

function getXMLHttpRequestObject(){
  //tutaj treść funkcji getXMLHttpRequestObject
}

function wyslijDane(val)
{
  var XMLHttpRequestObject = getXMLHttpRequestObject();

  if(XMLHttpRequestObject){
    var url = "http://localhost/dane.php?kategoria=" + val;

    XMLHttpRequestObject.open("GET", url);
    XMLHttpRequestObject.onreadystatechange = function()
    {
      if (XMLHttpRequestObject.readyState == 4 &&
          XMLHttpRequestObject.status == 200){
        var tekst = XMLHttpRequestObject.responseText;
        var div = document.getElementById('warstwaDanych');
        div.innerHTML = tekst;
      }
    }
    XMLHttpRequestObject.send(null);
  }
}
</script>
</head>
<body>
<div>
<?php generujPrzyciski() ?>
<hr />
```

```
</div>
<div id="warstwaDanych">
</div>
</body>
</html>
```

Ponieważ zaprezentowany kod zawiera części skryptu PHP, tym razem należy go zapisać w pliku *index.php* (a nie *index.html* — chyba że serwer został skonfigurowany w taki sposób, że pliki z rozszerzeniem *.html* są przetwarzane przez aparat wykonawczy PHP). Najpierw przyjrzyjmy się jednak części HTML. Otóż na stronie zostały umieszczone dwie warstwy. Treść pierwszej z nich jest generowana przez funkcję PHP o nazwie `generujPrzyciski`:

```
<?php generujPrzyciski() ?>
```

Treść drugiej (o identyfikatorze `warstwaDanych`) będzie tworzona dynamicznie w odpowiedzi na działania użytkownika.

Kliknięcie wybranego przycisku będzie powodowało wywołanie funkcji `wyslijDane`. Otrzymuje ona argument `val` określający, który przycisk został kliknięty, a dokładniej, zawierający wartość odpowiadającą danej kategorii tytułów. Zatem argument funkcji `wyslijDane` to wartość parametru kategoria, który ma być wysłany do skryptu PHP. Dlatego też adres URL tworzony jest za pomocą instrukcji:

```
var url = "http://localhost/dane.php?kategoria=" + val;
```

Inicjalizacja żądania, a także odbiór danych, są realizowane w standardowy sposób. Odebrany z serwera tekst jest odczytywany z właściwości `responseText` obiektu `XMLHttpRequestObject` i przypisywany zmiennej `tekst`. Zawartość tej zmiennej jest z kolei umieszczana na warstwie danych:

```
div.innerHTML = tekst;
```

Pozostała nam zatem do omówienia funkcja PHP generująca przyciski odpowiadające poszczególnym kategoriom. Nazwy kategorii i przypisane im identyfikatory są pobierane z tabeli `Kategorie` bazy danych. Niezbędne jest więc wykonanie zapytania SQL o postaci:

```
SELECT id, nazwa FROM Kategorie
```

Otwierane jest więc połączenie z bazą danych, a jego identyfikator zapisywany w zmiennej `$db_lnk`:

```
$db_lnk = @sqlite_open("./baza.sqlite", 0666, $msg);
```

a następnie wykonywane wspomniane zapytanie, którego wynik jest zapisywany w zmiennej `$arr`:

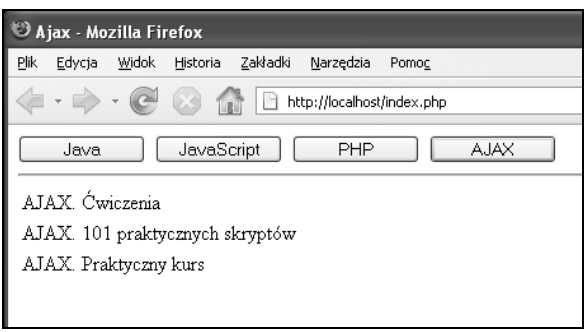
```
$arr = @sqlite_array_query($db_lnk, $query);
```

Następnie, jeśli obie te operacje zakończyły się sukcesem:

```
if($db_lnk && $arr){
```

wyniki są przetwarzane w pętli `foreach`, w której też generowane są znaczniki `<input>`. Dzięki nim na stronie pojawią się przyciski (tak jak jest to widoczne na rysunku 5.2). Pojedynczy znacznik ma ogólną postać:

```
<input type='button' value='nazwa_kategorii'
      "onclick='wyslijDane(id_kategorii);'
      "style='width:100px;'/>
```



Rysunek 5.2. *Po kliknięciu przycisku na stronie pojawiły się tytuły należące do danej kategorii*

Zatem na przyciskach pojawią się nazwy kategorii (odczytywane z tablicy `$row` z indeksu `1`), a jako argument funkcji `wyslijDane` przypisanej zdarzeniu `onclick` zostanie użyty identyfikator danej kategorii (odczytany z tablicy `$row` z indeksu `0`).