12. ÓRA

Adatbázisok kezelése – MySQL

A PHP nyelv egyik meghatározó tulajdonsága, hogy nagyon könnyen képes adatbázisokhoz csatlakozni és azokat kezelni. Ebben az órában elsősorban a MySQL adatbázisokkal foglalkozunk, de amint majd észre fogjuk venni, a PHP által támogatott összes adatbázist hasonló függvények segítségével érhetjük el. Miért esett a választás éppen a MySQL-re? Mert ingyenes, ugyanakkor nagyon hatékony eszköz, amely képes megfelelni a valós feladatok által támasztott igényeknek is. Az sem elhanyagolható szempont, hogy többféle rendszerhez is elérhető.
A MySQL adatbáziskiszolgálót a `http://www.mysql.com` címről tölthetjük le.
Ebben az órában a következő témákkal foglalkozunk:

- Megnézünk néhány SQL példát.
- Csatlakozunk egy MySQL adatbáziskiszolgálóhoz.
- Kiválasztunk egy adatbázist.
- Tanulunk a hibakezelésről.
- Adatokat viszünk fel egy táblába.

- Adatokat nyerünk ki egy táblából.
- Megváltoztatjuk egy adattábla tartalmát.
- Megjelenítjük az adatbázis szerkezetét.

(Nagyon) rövid bevezetés az SQL nyelvbe

ÚJDONSÁG

Az *SQL* jelentése *Structured Query Language*, vagyis strukturált lekérdező nyelv. Az SQL szabványosított nyelvezete segítségével a különböző típusú adatbázisokat azonos módon kezelhetjük. A legtöbb SQL termék saját bővítésekkel látja el a nyelvet, ahogy a legtöbb böngésző is saját HTML nyelvjárást „beszél". Mindazonáltal SQL ismeretek birtokában nagyon sokféle adatbázist fogunk tudni használni, a legkülönbözőbb operációs rendszereken.

A könyvben – terjedelmi okok miatt – még bevezető szinten sem tárgyalhatjuk az SQL-t, de megpróbálunk megvilágítani egy-két dolgot a MySQL-lel és általában az SQL-lel kapcsolatban.

A MySQL kiszolgáló démonként fut a számítógépen, így a helyi vagy akár távoli gépek felhasználói bármikor csatlakozhatnak hozzá. Miután a csatlakozás megtörtént, ki kell választanunk a megfelelő adatbázist, ha van jogunk hozzá.

Egy adatbázison belül több adattáblánk is lehet. Minden tábla oszlopokból és sorokból áll. A sorok és oszlopok metszéspontjában tároljuk az adatokat. Minden oszlopba csak előre megadott típusú adatot tölthetünk, az `INT` típus például egész számot, míg a `VARCHAR` változó hosszúságú, de egy adott értéknél nem hosszabb karakterláncot jelent.

A kiválasztott adatbázisban a következő SQL utasítással hozhatunk létre új táblát:

```
CREATE TABLE entablam ( keresztnev VARCHAR(30),
 ➥ vezeteknev VARCHAR(30), kor INT );
```

Ez a tábla három oszlopot tartalmaz. A `keresztnev` és `vezeteknev` oszlopokba legfeljebb 30 karaktert írhatunk, míg a `kor` oszlopba egy egész számot.

A táblába új sort az `INSERT` paranccsal vehetünk fel:

```
INSERT INTO entablam ( keresztnev, vezeteknev, kor )
 ➥ VALUES ( 'János', 'Kovács', 36 );
```

A mezőneveket az első, zárójelek közti kifejezéssel adjuk meg, míg az értékeket a megfelelő sorrendben a második zárójelpár között soroljuk fel.

A táblából az összes adatot a `SELECT` paranccsal kaphatjuk meg:

```
SELECT * FROM entablam;
```

A „*” szokásos helyettesítő karakter, jelentése „az összes mező”. Ha nem az összes mező tartalmát akarjuk lekérdezni, írjuk be az érintett mezők neveit a csillag helyére:

```
SELECT kor, keresztnev FROM entablam;
```

Már létező bejegyzést az `UPDATE` paranccsal módosíthatunk.

```
UPDATE entablam SET keresztnev = 'Gábor';
```

Az utasítás az összes sorban `"Gábor"`-ra módosítja a `keresztnev` mező tartalmát. A `WHERE` záradékkal leszűkíthetjük a `SELECT` és `UPDATE` parancsok hatáskörét. Például:

```
SELECT * FROM entablam WHERE keresztnev = 'Gábor';
```

Ez az utasítás csak azokat a sorokat írja ki, amelyekben a keresztnev mező értéke `"Gábor"`.

A következő példa csak azokban a sorokban változtatja `"Gábor"`-ra a `keresztnev` mező értékét, ahol a `vezeteknev` mező a `"Szakács"` karakterláncot tartalmazza.

```
UPDATE entablam SET keresztnev = "Gábor" WHERE vezeteknev =
➥ = "Szakács";
```

Az SQL-ről további információkat találunk Ryan K. Stephens és szerzőtársai Teach Yourself SQL in 21 Days című könyvében.

Csatlakozás a kiszolgálóhoz

Mielőtt elkezdhetnénk dolgozni az adatbázissal, csatlakoznunk kell a kiszolgálóhoz. A PHP-ben erre a `mysql_connect()` függvény szolgál. A függvény három karakterláncot vár paraméterként: a gazdagép nevét, a felhasználó nevét és a jelszót. Ha ezek egyikét sem adjuk meg, a függvény feltételezi, hogy a kérés a `localhost`-ra (azaz a helyi gépre) vonatkozik és felhasználóként a PHP-t futtató felhasználót, jelszóként pedig egy üres karakterláncot ad át. Az alapértelmezés

a `php.ini` fájlban felülbírálható, de egy próbálkozásra szánt kiszolgálót kivéve nem bölcs dolog ezzel próbálkozni, ezért a példákban mindig használni fogjuk a felhasználónevet és a jelszót. A `mysql_connect()` függvény siker esetén egy kapcsolatazonosítót ad vissza, amelyet egy változóba mentünk, hogy a későbbiekben folytathassuk a munkát az adatbáziskiszolgálóval.

Az alábbi kódrészlet a `mysql_connect()` függvény segítségével kapcsolódik a MySQL adatbáziskiszolgálóhoz.

```
$kapcsolat =  mysql_connect( "localhost", "root", "jelszo" );
if ( ! $kapcsolat )
   die( "Nem lehet csatlakozni a MySQL kiszolgalohoz!" );
```

Ha a PHP-t az Apache kiszolgáló moduljaként használjuk, a `mysql_pconnect()` függvényt is használhatjuk az előzőekben megadott paraméterekkel. Fontos különbség a két függvény között, hogy a `mysql_pconnect()`-tel megnyitott adatbáziskapcsolat nem szűnik meg a PHP program lefutásával vagy a `mysql_close()` függvény hatására (amely egy szokásos MySQL kiszolgálókapcsolat bontására szolgál) , hanem továbbra is aktív marad és olyan programokra várakozik, amelyek a `mysql_pconnect()` függvényt hívják. Más szóval a `mysql_pconnect()` függvény használatával megtakaríthatjuk azt az időt, ami egy kapcsolat felépítéséhez szükséges és egy előzőleg lefutott program által hagyott azonosítót használhatunk.

Az adatbázis kiválasztása

Miután kialakítottuk a kapcsolatot a MySQL démonnal, ki kell választanunk, melyik adatbázissal szeretnénk dolgozni. Erre a célra a `mysql_select_db()` függvény szolgál, amelynek meg kell adnunk a kiválasztott adatbázis nevét és szükség szerint egy kapcsolatazonosító értéket. Ha ez utóbbit elhagyjuk, automatikusan a legutoljára létrehozott kapcsolat helyettesítődik be. A `mysql_select_db()` függvény igaz értéket ad vissza, ha az adatbázis létezik és jogunk van a használatára. A következő kódrészlet a `pelda` nevű adatbázist választja ki.

```
$adatbazis = "pelda";
mysql_select_db( $adatbazis ) or die ( "Nem lehet
 ➥ megnyitni a következő adatbázist: $adatbazis" );
```

Hibakezelés

Eddig ellenőriztük a MySQL függvények visszatérési értékét és hiba esetén a `die()` függvénnyel kiléptünk a programból. A hibakezeléshez azonban hasznosabb lenne, ha a hibaüzenetek valamivel több információt tartalmaznának. Ha valamilyen művelet nem sikerül, a MySQL beállít egy hibakódot és egy hibaüzenetet. A hibakódhoz a `mysql_errno()`, míg a hibaüzenethez a `mysql_error()` függvénnyel férhetünk hozzá. A 12.1 példa az eddigi kódrészleteinket teljes programmá kapcsolja össze, amely kapcsolódik az adatbáziskezelőhöz és kiválasztja az adatbázist. A hibaüzenetet a `mysql_error()` függvénnyel tesszük használhatóbbá.

12.1. program Kapcsolat megnyitása és az adatbázis kiválasztása

```
 1: <html>
 2: <head>
 3: <title>12.1. program Kapcsolat megnyitása és
 4: adatbázis kiválasztása</title>
 5: </head>
 6: <body>
 7: <?php
 8: $felhasznalo = "jozsi";
 9: $jelszo = "bubosvocsok";
10: $adatbazis = "pelda";
11: $kapcsolat =  mysql_connect( "localhost",
                  $felhasznalo, $jelszo );
12: if ( ! $kapcsolat )
13:     die( "Nem lehet kapcsolódni
              a MySQL kiszolgálóhoz!" );
14: print "Sikerült a kapcsolatfelvétel<P>";
15: mysql_select_db( $adatbazis )
16:     or die ( "Nem lehet megnyitni a $adatbázist: "
                 .mysql_error() );
17: print "Sikeresen kiválasztott adatbázis: \"
           $adatbazis\"<P>";
18: mysql_close( $kapcsolat );
19: ?>
20: </body>
21: </html>
```

Ha az `$adatbazis` változó értékét mondjuk `"nincsmeg"`-re módosítjuk, a program egy nem létező adatbázist próbál meg elérni. Ekkor a `die()` kimenete valami ilyesmi lesz:

```
Nem lehet megnyitni a nincsmeg adatbázist: Access denied
 ➥ for user: 'jozsi@localhost' to database 'nincsmeg'
```

Adatok hozzáadása táblához

Már sikerült hozzáférést szerezni az adatbázishoz, így ideje némi adatot is bevinni a táblákba. A következő példákhoz képzeljük el, hogy az általunk készítendő oldalon a látogatóknak lehetőségük lesz tartományneveket vásárolni.

A `pelda` adatbázisban készítsük el az öt oszlopot tartalmazó `tartomanyok` táblát. Az `azonosito` mező lesz az elsődleges kulcs, amely automatikusan növel egy egész értéket, ahogy újabb bejegyzések kerülnek a táblába, a `tartomany` mező változó számú karaktert tartalmazhat (`VARCHAR`), a `nem` mező egyetlen karaktert, az `email` mező pedig a felhasználó elektronikus levélcímét.
A táblát a következő SQL paranccsal készíthetjük el:

```
CREATE TABLE tartomanyok ( azonosito INT NOT NULL
 ➥ AUTO_INCREMENT,
                PRIMARY KEY( azonosito ),
                tartomany VARCHAR( 20 ),
                nem CHAR( 1 ),
                email VARCHAR( 20 ) );
```

Az adatok felviteléhez össze kell állítanunk és le kell futtatnunk egy SQL parancsot, erre a célra a PHP-ben a `mysql_query()` függvény szolgál. A függvény paraméterként egy SQL parancsot tartalmazó karakterláncot és szükség szerint egy kapcsolatazonosítót vár. Ha ez utóbbit elhagyjuk, a függvény automatikusan az utoljára megnyitott kapcsolaton keresztül próbálja meg kiadni az SQL parancsot. Ha a program sikeresen lefutott, a `mysql_query()` pozitív értéket ad vissza, ha azonban formai hibát tartalmaz vagy nincs jogunk elérni a kívánt adatbázist, a visszatérési érték hamis lesz. Meg kell jegyeznünk, hogy egy sikeresen lefutott SQL program nem feltétlenül okoz változást az adatbázisban vagy tér vissza valamilyen eredménnyel. A 12.2 példában kibővítjük a korábbi programot és a `mysql_query()` függvénnyel kiadunk egy `INSERT` utasítást a `pelda` adatbázis tartomanyok tábláján.

12.2. program Új sor hozzáadása táblához

```
 1: <html>
 2: <head>
 3: <title>12.2. program Új sor hozzáadása
          táblához</title>
 4: </head>
 5: <body>
 6: <?php
 7: $felhasznalo = "jozsi";
 8: $jelszo = "bubosvocsok";
 9: $adatbazis = "pelda";
10: $kapcsolat =  mysql_connect( "localhost",
     $felhasznalo, $jelszo );
11: if ( ! $kapcsolat )
12:     die( "Nem lehet kapcsolódni
              a MySQL kiszolgálóhoz!" );
13: mysql_select_db( $adatbazis, $kapcsolat )
14:        or die ( "Nem lehet megnyitni a $adatbázist:
                    ".mysql_error() );
15: $parancs = "INSERT INTO tartomanyok ( tartomany,
              nem, email )
16:        VALUES ( '123xyz.com', 'F',
                    'okoska@tartomany.hu' )";
17: mysql_query( $parancs, $kapcsolat )
18: or die ( "Nem lehet adatot hozzáadni
             a \"tartomanyok\" táblához: "
19: .mysql_error() );
20: mysql_close( $kapcsolat );
21: ?>
22: </body>
23: </html>
```

Vegyük észre, hogy nem adtunk értéket az `azonosito` mezőnek. A mező értéke automatikusan növekszik az `INSERT` hatására.

Természetesen minden esetben, amikor a böngészőben újratöltjük a 12.2 példaprogramot, ugyanaz a sor adódik hozzá a táblához. A 12.3 példa a felhasználó által bevitt adatot tölti be a táblába.

12.3.program A felhasználó által megadott adatok beszúrása a táblába

```
 1: <html>
 2: <head>
 3: <title>12.3. program A felhasználó által megadott ada-
          tok beszúrása a táblába</title>
 4: </head>
 5: <body>
 6: <?php
 7: if ( isset( $tartomany ) && isset( $nem ) &&
         isset( $email ) )
 8:      {
 9:      // Ne feledjük ellenőrizni a felhasználó által
            megadott adatokat!
10:      $dbhiba = "";
11:      $vissza = adatbazis_bovit( $tartomany, $nem,
                                    $email, $dbhiba );
12:      if ( ! $vissza )
13:          print "Hiba: $dbhiba<BR>";
14:      else
15:          print "Köszönjük!";
16:      }
17: else     {
18:      urlap_keszit();
19:      }
20:
21: function adatbazis_bovit( $tartomany, $nem, $email,
                              &$dbhiba )
22:      {
23:      $felhasznalo = "jozsi";
24:      $jelszo = "bubosvocsok";
25:      $adatbazis = "pelda";
26:      $kapcsolat =  mysql_pconnect( "localhost",
                       $felhasznalo, $jelszo );
27:      if ( ! $kapcsolat )
28:          {
29:          $dbhiba = "Nem lehet kapcsolódni
                        a MySQL kiszolgálóhoz!";
30:          return false;
31:          }
32:      if ( ! mysql_select_db( $adatbazis, $kapcsolat ) )
33:          {
34:          $dbhiba = mysql_error();
```

12.3.program (folytatás)

```
35:          return false;
36:          }
37: $parancs = "INSERT INTO tartomanyok ( tartomany,
                nem, email )
38:      VALUES ( '$tartomany', '$nem', '$email' )";
39:      if ( ! mysql_query( $parancs, $kapcsolat ) )
40:          {
41:          $dbhiba = mysql_error();
42:          return false;
43:          }
44:      return true;
45:      }
46:
47: function urlap_keszit()
48:      {
49:      global $PHP_SELF;
50:      print "<form action=\"$PHP_SELF\"
                      method=\"POST\">\n";
51:      print "A kívánt tartomány<p>\n";
52:      print "<input type=\"text\" name=\"tartomany\"> ";
53:      print "Email cím<p>\n";
54:      print "<input type=\"text\" name=\"email\"> ";
55:      print "<select name=\"nem\">\n";
56:      print "\t<option value=\"N\"> Nő\n";
57:      print "\t<option value=\"F\"> Férfi\n";
58:      print "</select>\n";
59:      print "<input type=\"submit\"
               value=\"Elküld\">\n</form>\n";
60:      }
61: ?>
62: </body>
63: </html>
```

A tömörség érdekében a 12.3. példából kihagytunk egy fontos részt. Megbíztunk a felhasználókban: semmilyen formában nem ellenőriztük a felhasználó által bevitt adatokat. Ezt a feladatot a 17. órában tárgyalt karakterlánckezelő függvényekkel végezhetjük el. Meg kell jegyeznünk, hogy a beérkező adatok ellenőrzése mindig fontos feladat, itt csak azért maradt el, mivel így jobban összpontosíthattunk az óra témájára.

Ellenőrizzük a $tartomany, $nem és $email változókat. Ha megvannak, nyugodtan feltehetjük, hogy a felhasználó adatokat küldött az űrlap kitöltésével, így meghívjuk az adatbazis_bovit() függvényt.

Az adatbazis_bovit()-nek négy paramétere van: a $tartomany, a $nem és az $email változók, illetve a $dbhiba karakterlánc. Ez utóbbiba fogjuk betölteni az esetlegesen felbukkanó hibaüzeneteket, így hivatkozásként kell átadnunk. Ennek hatására ha a függvénytörzsön belül megváltoztatjuk a $dbhiba értékét, a másolat helyett tulajdonképpen az eredeti paraméter értékét módosítjuk.

Először megkísérlünk megnyitni egy kapcsolatot a MySQL kiszolgálóhoz. Ha ez nem sikerül, akkor hozzárendelünk egy hibaszöveget a $dbhiba változóhoz és false értéket visszaadva befejezzük a függvény futását. Ha a csatlakozás sikeres volt, kiválasztjuk a tartomany táblát tartalmazó adatbázist és összeállítunk egy SQL utasítást a felhasználó által küldött értékek felhasználásával, majd az utasítást átadjuk a mysql_query() függvénynek, amely elvégzi az adatbázisban a kívánt műveletet. Ha a mysql_select_db() vagy a mysql_query() függvény nem sikeres, a $dbhiba változóba betöltjük a mysql_error() függvény által visszaadott értéket és false értékkel térünk vissza. Minden egyéb esetben, vagyis ha a művelet sikeres volt, true értéket adunk vissza.

Miután az adatbazis_bovit() lefutott, megvizsgáljuk a visszatérési értéket. Ha true, az adatok bekerültek az adatbázisba, így üzenetet küldhetünk a felhasználónak. Egyébként ki kell írnunk a böngészőbe a hibaüzenetet. Az adatbazis_bovit() függvény által visszaadott $dbhiba változó most már hasznos adatokat tartalmaz a hiba természetét illetően, így ezt a szöveget is belefoglaljuk a hibaüzenetbe.

Ha a kezdeti if kifejezés nem találja meg a $tartomany, $nem vagy $email valamelyikét, feltehetjük, hogy a felhasználó nem küldött el semmilyen adatot, ezért meghívunk egy másik általunk írt függvényt, az urlap_keszit()-et, amely egy HTML űrlapot jelenít meg a böngészőben.

Automatikusan növekvő mező értékének lekérdezése

Az előző példákban úgy adtuk hozzá a sorokat a táblához, hogy nem foglalkoztunk az azonosito oszlop értékével, hiszen az adatok beszúrásával az folyamatosan növekedett, értékére pedig nem volt szükségünk. Ha mégis szükségünk lenne rá, egy SQL lekérdezéssel bármikor lekérdezhetjük az adatbázisból. De mi van akkor, ha azonnal szükségünk van az értékre? Fölösleges külön lekérdezést végrehajtani, hiszen a PHP tartalmazza a mysql_insert_id() függvényt, amely a legutóbbi INSERT kifejezés során beállított automatikusan növelt mező értékét adja vissza. A mysql_insert_id() függvénynek szükség esetén átadhatunk

egy kapcsolatazonosító paramétert, amelyet elhagyva a függvény alapértelmezés szerint a legutoljára létrejött MySQL kapcsolat azonosítóját használja.

Így ha meg akarjuk mondani a felhasználónak, hogy milyen azonosító alatt rögzítettük az adatait, az adatok rögzítése után közvetlenül hívjuk meg a `mysql_insert_id()` függvényt.

```
$parancs = "INSERT INTO tartomanyok ( tartomany, nem,
 ➥ email ) VALUES ( '$tartomany', '$nem', '$email' )";
mysql_query( $parancs, $kapcsolat );
$azonosito =  mysql_insert_id();
print "Köszönjük. Az Ön tranzakció-azonosítója:
 ➥ $azonosito. Kérjük jegyezze meg ezt a kódot.";
```

Adatok lekérdezése

Miután adatainkat már képesek vagyunk az adatbázisban tárolni, megismerkedhetünk azokkal a módszerekkel, amelyek az adatok visszanyerésére szolgálnak. Mint bizonyára már nyilvánvaló, a `mysql_query()` függvény használatával ki kell adnunk egy `SELECT` utasítást. Az azonban már korántsem ilyen egyszerű, hogyan jutunk hozzá a lekérdezés eredményéhez. Nos, miután végrehajtottunk egy sikeres `SELECT`-et, a `mysql_query()` visszaad egy úgynevezett eredményazonosítót, melynek felhasználásával elérhetjük az eredménytáblát és információkat nyerhetünk róla.

Az eredménytábla sorainak száma

A `SELECT` utasítás eredményeként kapott tábla sorainak számát a `mysql_num_rows()` függvény segítségével kérdezhetjük le. A függvény paramétere a kérdéses eredmény azonosítója, visszatérési értéke pedig a sorok száma a táblában. A 12.4. példaprogramban lekérdezzük a `tartomany` tábla összes sorát és a `mysql_num_rows()` függvénnyel megkapjuk a teljes tábla méretét.

12.4. program Sorok száma a SELECT utasítás eredményében.

```
1: <html>
2: <head>
3: <title>12.4. program A mysql_num_rows() függvény
          használata</title>
4: </head>
5: <body>
6: <?php
```

12.4. program (folytatás)

```
 7: $felhasznalo = "jozsi";
 8: $jelszo = "bubosvocsok";
 9: $adatbazis = "pelda";
10: $kapcsolat =  mysql_connect( "localhost",
                  $felhasznalo, $jelszo );
11: if ( ! $kapcsolat )
12:     die( "Nem lehet kapcsolódni
              a MySQL kiszolgálóhoz!" );
13: mysql_select_db( $adatbazis, $kapcsolat )
14:     or die ( "Nem lehet megnyitni a $adatbazis adatbá-
                  zist: ".mysql_error() );
15: $eredmeny = mysql_query( "SELECT * FROM tartomanyok" );
16: $sorok_szama = mysql_num_rows( $eredmeny );
17: print "Jelenleg $sorok_szama sor van a táblában<P>";
18: mysql_close( $kapcsolat );
19: ?>
20: </body>
21: </html>
```

A `mysql_query()` függvény egy eredményazonosítót ad vissza. Ezt átadjuk a `mysql_num_rows()` függvénynek, amely megadja a sorok számát.

Az eredménytábla elérése

Miután végrehajtottuk a `SELECT` lekérdezést és az eredményazonosítót tároltuk, egy ciklus segítségével férhetünk hozzá az eredménytábla soraihoz. A PHP egy belső mutató segítségével tartja nyilván, hogy melyik sort olvastuk utoljára. Ha kiolvasunk egy eredménysort, a program automatikusan a következőre ugrik.

A `mysql_fetch_row()` függvénnyel kiolvashatjuk a belső mutató által hivatkozott sort az eredménytáblából. A függvény paramétere egy eredményazonosító, visszatérési értéke pedig egy tömb, amely a sor összes mezőjét tartalmazza. Ha elértük az eredménykészlet végét, a `mysql_fetch_row()` függvény `false` értékkel tér vissza. A 12.5. példa a teljes `tartomany` táblát megjeleníti a böngészőben.

12.5. program Tábla összes sorának és oszlopának megjelenítése

```
 1: <html>
 2: <head>
 3: <title>12.5. program Tábla összes sorának és
          oszlopának megjelenítése</title>
 4: </head>
 5: <body>
 6: <?php
 7: $felhasznalo = "jozsi";
 8: $jelszo = "bubosvocsok";
 9: $adatbazis = "pelda";
10: $kapcsolat =  mysql_connect( "localhost",
                  $felhasznalo, $jelszo );
11: if ( ! $kapcsolat )
12:     die( "Nem lehet kapcsolódni
             a MySQL kiszolgálóhoz!" );
13: mysql_select_db( $db, $kapcsolat )
14:     or die ( "Nem lehet megnyitni a $adatbazis
                  adatbázist: ".mysql_error() );
15: $eredmeny = mysql_query( "SELECT * FROM tartomanyok"
);
16: $sorok_szama = mysql_num_rows( $eredmeny );
17: print "Jelenleg $sorok_szama sor van a táblában<P>";
18: print "<table border=1>\n";
19: while ( $egy_sor = mysql_fetch_row( $eredmeny ) )
20:     {
21:     print "<tr>\n";
22:     foreach ( $egy_sor as $mezo )
23:         print "\t<td>$mezo</td>\n";
24:     print "</tr>\n";
25:     }
26: print "</table>\n";
27: mysql_close( $kapcsolat );
28: ?>
29: </body>
30: </html>
```

Kapcsolódunk a kiszolgálóhoz és kiválasztjuk az adatbázist, majd a `mysql_query()` függvénnyel egy `SELECT` lekérdezést küldünk az adatbázis kiszolgálójához. Az eredményt az `$eredmeny` változóban tároljuk, amit az eredménysorok számának lekérdezéséhez használunk, ahogy korábban láttuk.

A `while` kifejezésben a `mysql_fetch_row()` függvény visszatérési értékét az `$egy_sor` változóba töltjük. Ne feledjük, hogy az értékadó kifejezés értéke megegyezik a jobb oldal értékével, ezért a kiértékelés során a kifejezés értéke mindig igaz lesz, amíg a `mysql_fetch_row()` függvény nem nulla értékkel tér vissza. A ciklusmagban kiolvassuk az `$egy_sor` változóban tárolt elemeket és egy táblázat celláiként jelenítjük meg azokat a böngészőben.

A mezőket név szerint is elérhetjük, méghozzá kétféle módon.
A `mysql_fetch_array()` függvény egy asszociatív tömböt ad vissza, ahol a kulcsok a mezők nevei. A következő kódrészletben módosítottuk a 12.5. példa `while` ciklusát, a `mysql_fetch_array()` függvény felhasználásával:

```
print "<table border=1>\n";
while ( $egy_sor = mysql_fetch_array( $eredmeny ) )
     {
     print "<tr>\n";
     print
"<td>".$egy_sor["email"]."</td><td>".$egy_sor["tartomany"].
 ➥ "</td>\n";
     print "</tr>\n";
     }
print "</table>\n";
```

A `mysql_fetch_object()` függvény segítségével egy objektum tulajdonságaiként férhetünk hozzá a mezőkhöz. A mezőnevek lesznek a tulajdonságok nevei. A következő kódrészletben ismét módosítottuk a kérdéses részt, ezúttal a `mysql_fetch_object()` függvényt használtuk.

```
print "<table border=1>\n";
while ( $egy_sor = mysql_fetch_object( $eredmeny ) )
     {
     print "<tr>\n";
     print "<td>".$egy_sor->email."</td><td>".$egy_sor->
             ➥ tartomany."</td>\n";
     print "</tr>\n";
     }
print "</table>\n";
```

A `mysql_fetch_array()` és `mysql_fetch_object()` függvények lehetővé teszik számunkra, hogy a sorból kinyert információkat szűrjük és végrehajtásuk sem kerül sokkal több időbe, mint a `mysql_fetch_row()` függvényé.

Adatok frissítése

Az adatokat az `UPDATE` utasítással frissíthetjük, amelyet természetesen a `mysql_query()` függvénnyel kell átadnunk az adatbázis kiszolgálójának. Itt is igazak a korábban elmondottak, azaz egy sikeres `UPDATE` utasítás nem feltétlenül jelent változást az adatokban. A megváltozott sorok számát a `mysql_affected_rows()` függvénnyel kérdezhetjük le, amelynek szükség szerint egy kapcsolatazonosítót kell átadnunk. Ennek hiányában a függvény – mint azt már megszokhattuk – a legfrissebb kapcsolatra értelmezi a műveletet. A `mysql_affected_rows()` függvényt bármely olyan SQL lekérdezés után használhatjuk, amely feltehetően módosított egy vagy több sort az adattáblában.

A 12.6. példa egy olyan programot tartalmaz, amely lehetővé teszi az adatbázis karbantartója számára, hogy bármelyik sor `tartomany` mezőjét megváltoztassa.

12.6. program Sorok frissítése az adatbázisban

```
 1: <html>
 2: <head>
 3: <title>12.6. program Sorok frissítése az adatbázisban
 4: </title>
 5: </head>
 6: <body>
 7: <?php
 8: $felhasznalo = "jozsi";
 9: $jelszo = "bubosvocsok";
10: $adatbazis = "pelda";
11: $kapcsolat =  mysql_connect( "localhost",
$felhasznalo, $jelszo );
12: if ( ! $kapcsolat )
13:      die( "Nem lehet kapcsolódni
               a MySQL kiszolgálóhoz!" );
14: mysql_select_db( $adatbazis, $kapcsolat )
15:      or die ( "Nem lehet megnyitni a $adatbazis
                   adatbázist: ".mysql_error() );
16: if ( isset( $tartomany ) && isset( $azonosito ) )
17:      {
18:      $parancs = "UPDATE tartomanyok SET tartomany =
                     '$tartomany' WHERE
                     azonosito=$azonosito";
19:      $eredmeny = mysql_query( $parancs );
```

12.6. program (folytatás)

```
20:      if ( ! $eredmeny )
21:           die ("Nem sikerült a módosítás: "
                   .mysql_error());
22: print "<h1>A tábla módosítva, ".
          mysql_affected_rows() .
23:      " sor változott</h1><p>";
24:      }
25: ?>
26: <form action="<? print $PHP_SELF ?>" method="POST">
27: <select name="azonosito">
28: <?
29: $eredmeny = mysql_query( "SELECT tartomany, azonosito
                              FROM tartomanyok" );
30: while( $egy_sor = mysql_fetch_object( $eredmeny ) )
31:      {
32:      print "<option value=\"$egy_sor->azonosito\"";
33:      if ( isset($azonosito) && $azonosito ==
               $egy_sor->azonosito )
34:           print " selected";
35:      print "> $egy_sor->tartomany\n";
36:      }
37: mysql_close( $kapcsolat );
38: ?>
39: </select>
40: <input type="text" name="tartomany">
41: <input type="submit" value="Frissítés">
42: </form>
43: </body>
44: </html>
```

A művelet a szokásos módon indul: kapcsolódunk az adatbázis kiszolgálójához és kiválasztjuk az adatbázist. Ezek után megvizsgáljuk a `$tartomany` és az `$azonosito` változók meglétét. Ha mindent rendben találunk, összeállítjuk az SQL utasítást, amely a `$tartomany` változó értékének megfelelően frissíti azt a sort, ahol az `azonosito` mező tartalma megegyezik az `$azonosito` változó értékével. Ha nem létező `azonosito` értékre hivatkozunk vagy a kérdéses sorban a `$tartomany` változó értéke megegyezik a `tartomany` mező tartalmával, nem kapunk hibaüzenetet, de a `mysql_affected_rows()` függvény visszatérési értéke 0 lesz. A módosított sorok számát kiírjuk a böngészőbe.

A karbantartó számára készítünk egy HTML űrlapot, amelyen keresztül elvégezheti a változtatásokat. Ehhez ismét a `mysql_query()` függvényt kell használnunk: lekérdezzük az `azonosito` és `tartomany` oszlopok összes elemét és beillesztjük azokat egy HTML `SELECT` listába. A karbantartó ezáltal egy lenyíló menüből választhatja ki, melyik bejegyzést kívánja módosítani. Ha a karbantartó már elküldte egyszer az űrlapot, akkor az utoljára hivatkozott `azonosito` érték mellé a `SELECTED` módosítót is kitesszük, így a módosítások azonnal látszódni fognak a menüben.

Információk az adatbázisokról

Mindezidáig egyetlen adatbázisra összpontosítottunk, megtanulva, hogyan lehet adatokkal feltölteni egy táblát és az adatokat lekérdezni. A PHP azonban számos olyan eszközzel szolgál, amelyek segítségével megállapíthatjuk, hány adatbázis hozzáférhető az adott kapcsolaton keresztül és milyen ezek szerkezete.

Az elérhető adatbázisok kiíratása

A `mysql_list_dbs()` függvény segítségével listát kérhetünk az összes adatbázisról, melyek az adott kapcsolaton keresztül hozzáférhetők. A függvény paraméterre szükség szerint egy kapcsolatazonosító, visszatérési értéke pedig egy eredményazonosító. Az adatbázisok neveit a `mysql_tablename()` függvénnyel kérdezhetjük le, amelynek paramétere ez az eredményazonosító és az adatbázis sorszáma, visszatérési értéke pedig az adatbázis neve. Az adatbázisok sorszámozása `0`-val kezdődik. Az összes adatbázis számát a `mysql_list_dbs()` függvény után meghívott `mysql_num_rows()` függvény adja meg.

12.7. program Adott kapcsolaton keresztül elérhető adatbázisok

```
 1: <html>
 2: <head>
 3: <title>12.7. program Adott kapcsolaton keresztül
           elérhető adatbázisok
 4: </title>
 5: </head>
 6: <body>
 7: <?php
 8: $felhasznalo = "jozsi";
 9: $jelszo = "bubosvocsok";
10: $kapcsolat =  mysql_connect( "localhost",
                  $felhasznalo, $jelszo );
```

12.7. program (folytatás)

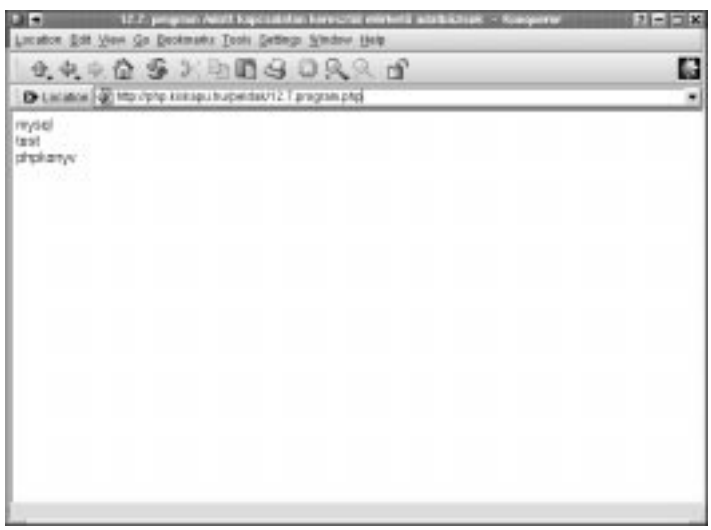
```
11: if ( ! $kapcsolat )
12:      die( "Nem lehet kapcsolódni
                a MySQL kiszolgálóhoz!" );
13: $adatbazis_lista = mysql_list_dbs( $kapcsolat );
14: $szam = mysql_num_rows( $adatbazis_lista );
15: for( $x = 0; $x < $szam; $x++ )
16:      print mysql_tablename( $adatbazis_lista,
                                $x )."<br>";
17: mysql_close( $kapcsolat );
18: ?>
19: </body>
20: </html>
```

A `mysql_list_dbs()` függvény visszaad egy eredményazonosítót, amelyet paraméterként átadunk a `mysql_num_rows()` függvénynek. Az így megkapott adatbázisok számát a `$szam` változóba töltjük és felhasználjuk a `for` ciklusban.

Az `$x` indexet minden lépésben eggyel növeljük, `0`-tól a talált adatbázisok számáig, majd az eredményazonosítóval együtt átadjuk a `mysql_tablename()` függvénynek az adatbázis nevének lekérdezéséhez. A 12.1 ábrán a 12.7 program eredménye látható egy böngészőablakban.

12.1. ábra

Az elérhető adatbázisok listája



A `mysql_list_dbs()` függvény által visszaadott eredményazonosítót a `mysql_query()` eredményéhez hasonlóan is felhasználhatnánk, vagyis a tényleges neveket a `mysql_fetch_row()` függvénnyel is lekérdezhetnénk. Ekkor eredményül egy tömböt kapnánk, melynek első eleme tartalmazná az adatbázis nevét.

Adatbázistáblák listázása

A `mysql_list_tables()` függvénnyel egy adott adatbázis tábláinak nevét soroltathatjuk fel. A függvény paraméterei az adatbázis neve és szükség szerint a kapcsolat azonosítója. A visszatérési érték egy eredményazonosító, ha az adatbázis létezik és jogunk van hozzáférni. Az eredményazonosítót `mysql_tablename()` vagy `mysql_fetch_row()` hívásokhoz használhatjuk fel, a fent látott módon.

A következő kódrészlet a `mysql_list_tables()` függvény segítségével az adatbázis összes tábláját kiírja.

```
$eredmeny = mysql_list_tables( "pelda", $kapcsolat );
while ( $tabla_sor =  mysql_fetch_row( $eredmeny ) )
         print $tabla_sor[0]."<BR>\n";
```


Információk a mezőkről

A `SELECT` lekérdezés után az eredménytábla mezőinek számát a `mysql_num_fields()` függvénnyel kaphatjuk meg. A függvény paramétere egy eredményazonosító, visszatérési értéke pedig egy egész szám, amely a mezők számát adja meg.

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );
$mezok_szama = mysql_num_fields( $eredmeny );
```

A mezőkhöz sorszámokat rendelünk, a számozás `0`-val kezdődik. A sorszám és az eredményazonosító alapján számos információt lekérdezhetünk a mezőről, beleértve a nevét, típusát, legnagyobb hosszát és egyéb jellemzőit is.

A mező nevének lekérdezésére a `mysql_field_name()` függvény szolgál.

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );
$mezok_szama = mysql_num_fields( $eredmeny );
for ( $x=0; $x<$mezok_szama; $x++ )
      mysql_field_name( $eredmeny, $x ) . "<br>\n";
```

A mező legnagyobb hosszát a `mysql_field_len()` függvénnyel kaphatjuk meg.

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );
$mezok_szama = mysql_num_fields( $eredmeny );
for ( $x=0; $x<$mezok_szama; $x++ )
     mysql_field_len( $eredmeny, $x ) . "<BR>\n";
```

A mező egyéb jellemzőit a `mysql_field_flags()` függvénnyel olvashatjuk ki:

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );
$mezok_szama = mysql_num_fields( $eredmeny );
for ( $x=0; $x<$mezok_szama; $x++ )
     mysql_field_flags( $eredmeny, $x ) . "<BR>\n";
```

Lehetőségünk van még a mező típusának megállapítására,
a `mysql_field_type()` függvénnyel:

```
$eredmeny = mysql_query( "SELECT * from tartomanyok" );
$mezok_szama = mysql_num_fields( $eredmeny );
for ( $x=0; $x<$mezok_szama; $x++ )
     mysql_field_type( $eredmeny, $x ) . "<BR>\n";
```

Az adatbázis szerkezete – összeáll a kép

A 12.8 példa egyesíti a fenti eljárásokat és megjelenít minden elérhető adatbázist, táblát és mezőt.

12.8 program Minden adatbázis, tábla és mező megjelenítése

```
1: <html>
2: <head>
3: <title>12.8. program Minden adatbázis, tábla és
          mező megjelenítése</title>
4: </head>
5: <body>
6: <?php
7: $felhasznalo = "root";
8: $jelszo = "titkos";
9: $kapcsolat =  mysql_connect( "localhost",
                  $felhasznalo, $jelszo );
```

12.8 program (folytatás)

```
10: if ( ! $kapcsolat )
11:      die( "Nem lehet kapcsolódni
               a MySQL kiszolgálóhoz!" );
12: $adatbazis_lista = mysql_list_dbs( $kapcsolat );
13: while ( $adatbazisok = mysql_fetch_row
          ( $adatbazis_lista ) )
14:      {
15:      print "<b>".$adatbazisok[0]."</b>\n";
16:      if ( !@mysql_select_db( $adatbazisok[0],
$kapcsolat ) )
17:           {
18:           print "<dl><dd>Nem lehet kiválasztani – " .
                    mysql_error() ." </dl>";
19:           continue;
20:           }
21:      $tablak = mysql_list_tables( $adatbazisok[0],
                    $kapcsolat );
22:      print "\t<dl><dd>\n";
23:      while ( $tabla_sor =  mysql_fetch_row( $tablak )
)
24:           {
25:           print "\t<b>$tabla_sor[0]</b>\n";
26:           $eredmeny = mysql_query( "SELECT * from "
                         .$tabla_sor[0] );

27:          $mezok_szama = mysql_num_fields( $eredmeny );
28:          print "\t\t<dl><dd>\n";
29:          for ( $x=0; $x<$mezok_szama; $x++ )
30:               {
31:               print "\t\t<i>";
32:               print mysql_field_type( $eredmeny, $x );
33:               print "</i> <i>";
34:               print mysql_field_len( $eredmeny, $x );
35:               print "</i> <b>";
36:               print mysql_field_name( $eredmeny, $x );
37:               print "</b> <i>";
38:               print mysql_field_flags( $eredmeny, $x );
39:               print "</i><br>\n";
40:               }
41:          print "\t\t</dl>\n";
42:          }
```

12.8 program (folytatás)

```
43:      print "\t</dl>\n";
44:      }
45: mysql_close( $kapcsolat );
46: ?>
47: </body>
48: </html>
```

Először is a szokásos módon a MySQL kiszolgálóhoz kapcsolódunk, majd meghívjuk a `mysql_list_dbs()` függvényt. A kapott eredményazonosítót átadjuk a `mysql_fetch_row()` függvénynek és az `$adatbazisok` tömb első elemébe betöltjük az aktuális adatbázis nevét.

Ezután megpróbáljuk kiválasztani az adatbázist a `mysql_select_db()` függvénnyel. Ha nincs hozzáférési jogunk, akkor megjelenítjük a hibaüzenetet a böngészőablakban, majd a `continue` utasítással továbblépünk a következő adatbázisra. Ha ki tudjuk választani az adatbázist, a műveletet a táblanevek kiírásával folytatjuk.

Az adatbázis nevét átadjuk a `mysql_list_tables()` függvénynek, tároljuk az új eredményazonosítót, majd a `mysql_fetch_row()` függvénnyel kiolvassuk a táblák neveit. Minden táblán a következő műveletsort hajtjuk végre: kiírjuk a tábla nevét, majd összeállítunk egy SQL utasítást, amely az összes oszlopot lekérdezi az adott táblában. A `mysql_query()` függvénnyel végrehajtjuk a lekérdezést, majd az eredményazonosító alapján a `mysql_num_fields()` függvénnyel meghatározzuk a mezők számát.

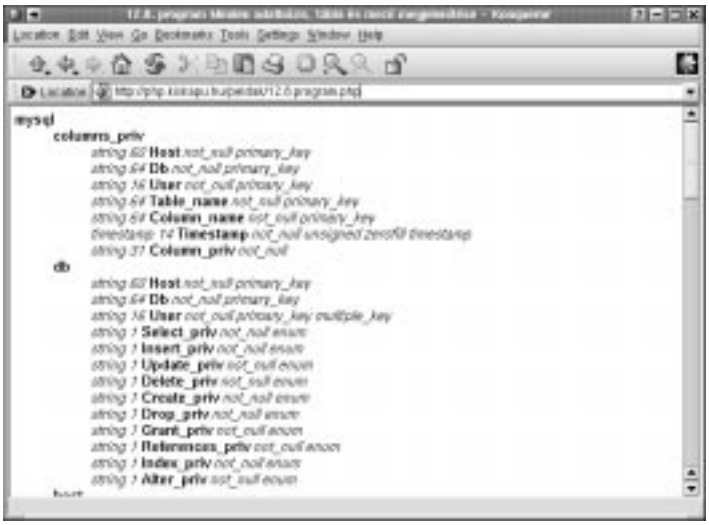
A `for` ciklusban minden mezőn elvégezzük a következő műveletsort: beállítjuk az `$x` változót, hogy az aktuális mező sorszámát tartalmazza (ez kezdetben `0`). Ezután a sorszám ismeretében meghívjuk az előző részben tárgyalt összes ellenőrző függvényt, eredményüket pedig átlátható formában, táblázatba rendezve elküldjük a böngészőnek.

Ezeket a műveleteket minden elérhető adatbázison elvégezzük.

A program egy áttekinthető szerkezeti vázlatot jelenít meg a böngészőablakban, amely az adott kapcsolaton keresztül elérhető összes adatbázist, adattáblát és mezőt tartalmazza. A 12.2. ábra a program kimenetét mutatja.

12.2. ábra

Az összes elérhető adatbázis, tábla és mező listája



Összefoglalás

Ebben az órában a MySQL adatbáziskezelés alapjait tanultuk meg: az adatok tárolását és visszanyerését.

Megtanultuk, hogyan építsük fel a kapcsolatot a MySQL kiszolgálóval a `mysql_connect()` és `mysql_pconnect()` függvények segítségével.

Az adatbázist a `mysql_select_db()` függvénnyel választottuk ki. Ha a kiválasztás nem sikerült, lekérdeztük a hibát a `mysql_error()` függvénnyel.

SQL utasításokat adtunk ki a `mysql_query()` függvény segítségével és a függvény által visszaadott eredményazonosító segítségével elértük az adatokat vagy megtudtuk a módosított sorok számát.

A PHP MySQL függvényeinek segítségével kiírattuk az elérhető adatbázisokat, táblákat és mezőket, és az egyes mezőkről is bővebb információkat szereztünk.

Kérdések és válaszok

Ez az óra szigorúan a MySQL-re vonatkozik. Hogyan ültethetjük át ezeket a fogalmakat más adatbázissal működő rendszerekre?
Az mSQL-kezelő függvények például szinte teljesen megegyeznek a MySQL függvényekkel, de más SQL kiszolgálóknak is megvannak a saját PHP függvényeik. SQL utasításokat minden adatbáziskiszolgálónak küldhetünk. Ha szabványos ANSI SQL-lel dolgozunk, nem lesz különösebb problémánk az adatbáziskiszolgálók közötti átálláskor.

Hogyan írhatunk olyan kódot, amelyet könnyű átültetni egy másik adatbázis kiszolgáló környezetbe?
Gyakran jó ötlet az adatbázis-lekérdező függvényeket egy egyszerű osztályba vagy könyvtárba csoportosítani. Így ha fel kell készíteni a kódot egy másik adatbáziskezelővel való együttműködésre, a kódnak csak kis részét kell átírni.

Műhely

A műhelyben kvízkérdések találhatók, melyek segítenek megszilárdítani az órában szerzett tudást. A válaszokat az A függelékben helyeztük el.

Kvíz

1. Hogyan kell csatlakozni a MySQL adatbáziskiszolgálóhoz?

2. Melyik függvény szolgál az adatbázis kiválasztására?

3. Melyik függvénnyel küldhetünk SQL utasítást a kiszolgálónak?

4. Mit csinál a `mysql_insert_id()` függvény?

5. Tegyük fel, hogy küldtünk egy `SELECT` lekérdezést a MySQL-nek. Nevezzünk meg három függvényt, amelyekkel hozzáférhetünk az eredményhez.

6. Tegyük fel, hogy küldtünk egy `UPDATE` lekérdezést a MySQL-nek. Melyik függvénnyel állapíthatjuk meg, hány sort érintett a módosítás?

7. Melyik függvényt kellene használnunk, ha az adatbáziskapcsolaton keresztül elérhető adatbázisok nevét szeretnénk megtudni?

8. Melyik függvénnyel kérdezhetjük le az adatbázisban található táblák neveit?

Feladatok

1. Hozzunk létre egy adattáblát három mezővel: `email` (legfeljebb 70 karakter), `uzenet` (legfeljebb 250 karakter) és `datum` (a UNIX időbélyegzőt tartalmazó egész szám). Tegyük lehetővé a felhasználóknak, hogy feltöltsék az adatbázist.

2. Készítsünk programot, amely megjeleníti az 1. pontban elkészített tábla tartalmát.