PHP SQLite adatbázisháttérrel

Eddigi PHP alapokra épülő munkáim során ritka volt az olyan feladat, ahová ne jött volna jól egy kis adatbázis támogatás a hatékony adatkezelés érdekében. Ámde a PHP-ben legyártott alkalmazások nagy része olyan környezetben éli életét, ahol az adatbázis kiszolgáló ugyanazon a gépen található, vagy esetleg egy ugyanazon hálózaton figyelő másikon.

Bevezetés

Bár az adatbázis szerverek maguk lehetővé teszik, az ilyen kis és közepes projektek nem igényelnek többet egy darab adatbázis hozzáférést biztosító felhasználónév/jelszó párosnál. De mivel a nyers fájlokkal való izommunka mégsem olyan csábító választási lehetőség, marad az *SQL* adatbázis szerverek dolgoztatása, felesleges hálózati rétegekkel, széleskörű jogosultság rendszerekkel. Ez így is ment mindaddig, míg csak egy napon elém nem került az ***SQLite***, a beágyazható adatbáziskezelő motor. Bár nem lehet belőle nagyvállalati központi adattár kiszolgálót építeni, de saját kis webhelyeink blogjainak, fórumainak, on-line boltocskáinak meghajtására kiválóan alkalmas.
Ez a cikk azok számára lehet hasznos olvasmány, akik már foglalkoztak a *PHP* valamely adatbáziskezelő kiterjesztésével, mivel javarészt inkább csak a különbségek, egyedi megoldások bemutatására szorítkozom.

Telepítés, rendszerigény

A *PHP SQLite* kiterjesztésének birtokbavétele nem tartozik a keményebb rendszergazdai feladatok közé. Az *SQLite* tervezésekor az egyik fő szempont pont az volt, hogy könnyen illeszthető legyen bármihez, és lehetőleg ne függjön léte más függvénykönyvtáraktól. Így ahol a *PHP* futásképes, ott az *SQLite*-al sem lesz gond. Munkába fogásához csak a megfelelő *PHP* modul telepítése szükséges. A legtöbb kiterjesztéssel ellentétben ennek telepíthetősége nem függ külső függvénykönyvtár meglététől, mivel a *PHP* kiterjesztés magában foglalja a teljes *SQLite* függvénykönyvtárt. Nos, nem hiába hívják beágyazható adatbáziskezelőnek.
A *PHP 5* esetében az alaptelepítéssel már kezünkben is van a teljes *SQLite* eszköztár telepítve, élesítve. A 4-es változatok esetében a *PHP Pear* névre hallgató csomagkezelőjét kell kicsit munkára fognunk, a

```
pearinstall sqlite
```

utasítás parancssorba gépelésével, minek eredményeként letöltésre, majd gépünkön forrásból fordításra kerül a megfelelő PHP kiterjesztésmodul. A sikeres fordításhoz a gcc mellett szükség lesz az autotools (autoconf, automake) eszközökre is.
Korábbi *PHP 4*-es változatok esetén előfordulhat, hogy egy kellemetlenkedő,

```
No releases found for package 'sqlite'
```

üzenetet kapunk próbálkozásunkra. Semmi gond, csak a *PHP*-vel kapott *Pear* telepítő kicsit öreg, így előbb saját maga frissítésére kell rávennünk. A következő parancs hatására ez meg is történik, ami után az ***sqlite*** csomagot is meg fogja találni:

```
pear upgrade Console_Getopt PEAR Archive_Tar
```

Siker esetén helyére kerül az ***sqlite.so*** állomány, melyet vagy a ***php.ini*** egy megfelelő `extension=sqlite.so` sorával, vagy a *PHP* parancsállományaink elejére tett `dl('sqlite.so')` utasítással tudjuk a PHP részévé tenni. Innentől kezdve rendelkezésünkre áll az ***sqlite*** parancskészlet.

Gyorstalpaló

Az *SQLite* – hasonlóan a *PHP*-hez – nem kifejezetten típusérzékeny. Valójában jórészt magasról tesz rá, milyen típust adtunk meg melyik oszlopnál, egy `INT` típusú mezőbe büntetlenül tehetünk szöveget is. A szövegesnek megjelölt oszlopok méretmeghatározása is inkább csak arra jó, hogy magunknak dokumentáljuk, körülbelül milyen adatokat is szeretnénk tárolni. Hasznos információvá szinte csak a tábla neve és az oszlopnevek válnak. Bizonyos esetekben, például rendezések során persze szerepet kap a megadott oszloptípus, ami szerint számszerűleg, vagy szövegként kezelve rendezi sorba az elemeket.
Aki már foglalkozott a *PHP* és valamely adatbázis kiszolgálótípus házasításával, az *SQLite* modul parancskészlete sem fog nagy meglepődéseket okozni. Sajnos ez a modul is a maga útját járja, így egyik jelenlegi *SQL* motorral dolgozó kiterjesztésnek sem felel meg egy az egyben parancskészlete.

1. lista

```
<?php
if (!function_exists('sqlite_fetch_single')) {
    function sqlite_fetch_single($res) {
        $row = sqlite_fetch_array($res,
            ➥SQLITE_NUM);
        return $row[0];
    }
}
?>
```

Mivel nincs külön adatbázis szerver, kapcsolódnunk nem kell hozzá, valamint jelszó sem kell az adatbázis eléréséhez. Ez a művelet inkább hasonlatos ahhoz, ahogy egy fájlt megnyitunk:

```
$db = sqlite_open(<fájlnév>, [fájljogok],
➥[&hibaüzenet]);
```

Az egyetlen, amit mindenképp meg kell adnunk, az ***SQLite*** adatbázist megtestesítő fájl neve, esetlegesen az elérési útjával egyetemben. A második paraméter alapértéke a 0666 oktális érték, ami a `rw-rw-rw-` fájljogosultságnak felel meg. Jelenleg ezen paraméternek nincs jelentősége, bármit is adunk meg, az ***SQLite*** ezzel a móddal fog próbálkozni. Harmadik paraméterként megadhatunk egy változót, amibe az esetleges szöveges hibaüzenet kerül, ha az adatbázis megnyitása kudarcba fulladt volna. Ilyen esetekben az `sqlite_open()` amúgy `false` értékkel fog visszatérni. Amennyiben a megadott helyen nem található a keresett fájl, ezen utasítással létre is hozzuk azt, feltéve ha erre a könyvtárra a webszerver felhasználója megfelelő jogosultságot kap. Fontos tudni, hogy nem elég, ha a létrehozás idejére adunk csak írási jogot a hordozó könyvtárra. Az adatbázis módosításához, bővítéséhez ugyanis nem elég, ha magát a fájlt tudjuk írni, mivel ezen műveletek során ugyanitt ideiglenes állományok létrehozására is szüksége lesz ***SQLite*** motorunknak. Amint sikeresen rendelkezünk egy megnyitott adatbázissal, minden úgy megy, ahogy azt más adatbázisoknál is megszokhattuk. Az SQL kérések futtatására az `sqlite_query()` valamint az `sqlite_exec()` szolgál, míg az eredménylisták kisajtolására az `sqlite_fetch_*` függvények. Munkánk végeztével egy esetleges `sqlite_close()` által felszabadíthatjuk lefoglalt erőforrásainkat.

Az `sqlite_query()` és `sqlite_exec()` is ***SQL*** parancsok végrehajtására szolgál, de csak az első ad vissza eredménylistát. Az `sqlite_exec()` csupán igaz/hamis értéket ad attól függően, hogy a parancs futtatása sikeres volt, vagy sem. Ezen függvények két adatot várnak, a megnyitott adatbázisra mutató erőforrás azonosítót, valamint magát az ***SQL*** kérést. A két paraméter tetszőleges sorrendben megadható, a kézikönyv az adatbázis azonosító előre vételét javasolja. Aki sokat dolgozott például ***mysql***-el, esetleg kényelmesebbnek találhatja az ott megszokott sorrendet. Egy paranccsal egyszerre több ***SQL*** kérést is futtathatunk, ezeket az `sqlite_exec()` minden esetben futtatni fogja maradéktalanul. Az `sqlite_query()` csak akkor, ha a visszakapott eredményazonosítót a későbbiek folyamán nem használjuk fel. Ha mégis, akkor csak az első kérés fog végrehajtódni. Sikeres lekérdezés után jön az adatkinyerés az eredménylistából. Teljes, több mezős adatsort, az `sqlite_fetch_array()` függvénnyel tudunk olvasni. Első és egyetlen kötelező paraméterként az eredménylista azonosítót várja. Ellentétben más adatbázis kiterjesztésekkel, itt nem áll rendelkezésre `sqlite_fetch_row()` függvény. Erre is az előbbi alkalmazandó, a második paraméterben megadva, milyen formában kérjük az adatsort. Ehhez az ***SQLite*** kiterjesztés három előre megadott állandót alkalmaz:

2. lista

```
<?php

/* Ideiglenes adatbázis létrehozása tesztadatokkal */
$db = sqlite_open(':memory:');
sqlite_exec($db, 'CREATE TABLE ertekek (szam)');
foreach (array(14, 33, 62, 732, 1, 57, 21) as
        ➥$szam) {
    sqlite_exec($db, 'INSERT INTO ertekek VALUES
            ➥(' . $szam .')');
}

/* Saját (lentebb megvalósított) függvényeink
átadása SQLite-nak */
sqlite_create_aggregate($db, 'avg',
➥'avg_feldolgozo', 'avg_osszegzo');
sqlite_create_function($db, 'dup', 'dup');

/* Lekérdezés, melyben használjuk mindkét függvényt */
$res = sqlite_query($db, 'SELECT avg(dup(szam))
                ➥FROM ertekek');
echo sqlite_fetch_single($res);
sqlite_close($db);

/* Az SQLite számára elállított függvények */
function dup($ertek) {
    return $ertek * 2;
}

function avg_feldolgozo($kornyezet,
                    ➥$aktualis_ertek) {
    $kornyezet['osszeg'] += $aktualis_ertek;
    $kornyezet['elemek'] += 1;
}

function avg_osszegzo($kornyezet) {
    return ($kornyezet['osszeg'] /
            ➥$kornyezet['elemek']);
}

?>
```

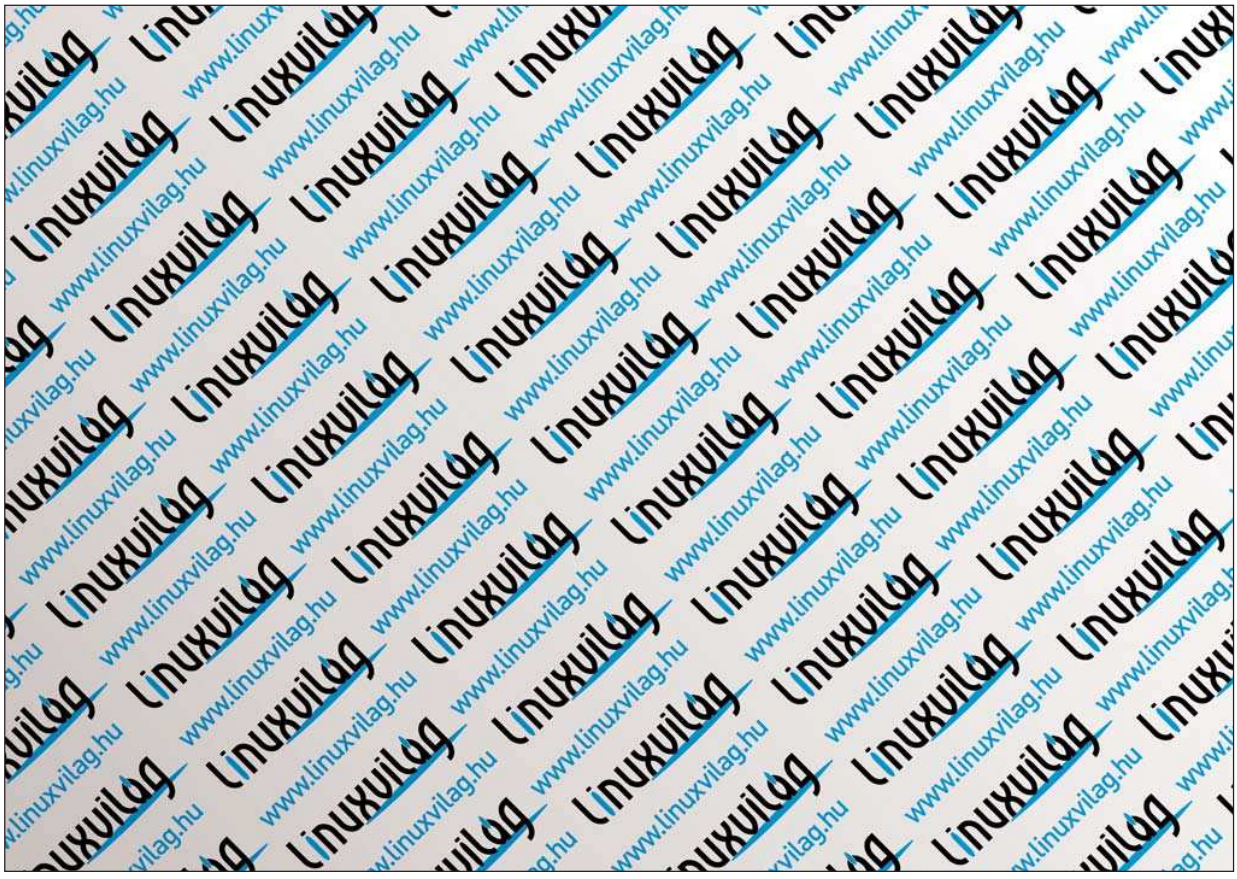
- `SQLITE_ASSOC` – a mezőnévnek megfelelő indexre teszi az értékeket
- `SQLITE_NUM` – a mező sorszámának megfelelő indexre teszi az értékeket
- `SQLITE_BOTH` – mind név, mind oszlopszám szerint is elhelyezi az adatokat

Ha amúgy is egy nagy tömbbe szeretnénk ciklikusan beolvasni a teljes eredménylistát, érdemes `sqlite_fetch_all()`-ra bízni ezt. Ez a teljes eredménylistát beolvassa egy tömbbe, melynek minden eleme egy adott sor lesz. Paraméterezésre teljesen megegyezik az előzővel. Mivel ekkor a teljes visszakapott adatsor a memóriába kerül, nagy listák lekérésekor ez a módszer nem javallott. Általában azonban igaz, hogy 50 sornyi adatnál többet amúgy sem illik a látogató orra alá dörgölni egyszerre.
Az ilyen tömbbe lekérdezésre van viszont az ***SQLite*** kiterjesztésnek egy még kényelmesebb változata. Ezt `sqlite_array_query()` néven érhetjük el és egyesíti az `sqlite_query()` valamint az `sqlite_fetch_all()` függvényeket. Első két paramétere a kérésfuttató függvényekével egyezik meg, harmadikként pedig megmondhatjuk, milyen formában kérjük az adatokat a tömbbe helyezni.
Számomra szimpatikus, hogy az egyetlen adat lekérését megkönnyítendő, létezik egy `sqlite_fetch_single()` függvény. Ez sajnos a ***PHP 4***-es változatokhoz telepíthető kiterjesztésben nincs benne, de a ***PHP 5*** tartalmazza.
Az *1. lista* bemutat egy egyszerű kódocskát mellyel eme hiány 4-es változatokban áthidalható.

Ennyivel a mindennapokban nagyjából el lehet boldogulni, csupán csak a hibakeresésről nem esett szó. Ez itt két lépésben történik, először meg kell tudnunk a feltételezett hiba kódját. Erre az `sqlite_last_error()` függvénnyel nyílik lehetőség. Egyetlen paraméterében sajnos mindenképp meg kell adni, mely adatbázissal kapcsolatban érdekel ez az adat. Máshol, például a ***MySQL*** kiterjesztés esetében alapértelmezésként a legutóbb megnyitott adatbázist veszi elő ilyen adat hiányában. Persze egy hibát azonosító szám nem sokat segít a hiba felderítésében. Többre jutunk, ha a hibakódot az `sqlite_error_string()` függvénnyel lefordíttatjuk emberi nyelvre. A hibakódért cserébe ez annak szöveges változatát adja vissza.

Szorosabb együttműködés

A kisebb erőforrás igények mellett a beágyazottságnak van még egy nagy előnye. ***SQLite*** kéréseinkben tetszőleges bonyolultságú, ***PHP***-ben írt függvényeinket is alkalmazhatjuk. Néhány alapvető függvénnyel az ***SQLite*** is rendelkezik, de akár ezek felülírására is képesek vagyunk. Ehhez először is létre kell hoznunk egy, a célt megvalósító ***PHP*** függvényt, mely a bejövő adat(ok)on tetszőleges műveleteket végezve végül visszaad egy értéket. Ez csak az ***SQLite*** által is értelmezhető adat lehet, azaz nem adhat vissza tömböt, objektumot, erőforrás hivatkozást. Ha függvényünk harcra kész, az `sqlite_create_function()` függvény segítségével tudathatjuk, hogy azt az ***SQLite*** lekéréseiben alkalmazni szeretnénk. Első paramétereként az adatbázis azonosítót, másodikként és harmadikként pedig egy-egy függvény nevet.



Először a függvény lekérésekben használatos nevét, majd a már létrehozott *PHP* megvalósítás nevét kell megadni. Negyedik adatként megadhatjuk az *SQLite* motornak, hány paramétert várjon el ez a függvény.
Az már csak a fantázián múlik, a függvényen belül mihez kezdünk a kapott adattal. Akár más típusú adatbázisokhoz is hozzáférhetünk, IP cím alapján kideríthetjük a hozzá tartozó gépnevet. Persze a lekérés hatékonysága a feladatok bonyolultságának növekedtével alaposan visszaeshet.
Nem csupán az egyes értékek módosítására van lehetőségünk, de a `COUNT()`, `SUM()` és hasonló, a teljes eredménylistát átfogó függvényeket is létrehozhatunk *PHP* nyelven. Az `sqlite_create_aggregate()` által hozhatjuk ezt adatbázisunk tudtára. Elsőként ez is az adatbázist kéri beazonosítani, másodikként pedig a majdani *SQL* kérésekben alkalmazott függvénynevet kell meghatározni. Az átfogó függvény megvalósításához azonban két különálló függvényre lesz szükség. Ezeket adhatjuk meg a harmadik, illetve a negyedik paraméterekben. Így ötödik helyre csúszik annak megadási lehetősége hány paramétert is vár függvényünk.
A megadandó függvénypár első tagja minden egyes kapott soron végrehajtásra kerül, majd végül az adatgyűjtés végeztével a második függvény fogja megmondani a választ. Első paraméterként mindkét függvény egy tetszőlegesen felhasználható, referenciaként kezelt változót kap, nevezzük ezt „környezetnek”. Ebben a sorok feldolgozása idején tetszőleges méretű adathalmazt tárolhatunk. Végül az összegző függvényebből okoskodhatja ki az eredményt. Ha lehetséges, érdemes amit csak lehet már menet közben kiszámítani és minél kevesebb nyers adatot tárolni feleslegesen. Ugyanis nagy lekéréseknél egy hatalmasra növekvő tömb csak kiverné a biztosítékot. Az összegző függvénynek nincs is szüksége más adatra, mint erre a környezetleíró adat(halmaz)ra. A soronként meghívott függvény ezután minimum egy, de akár több adatot is várhat, szükség szerint. A 2. *lista* egy nem túl bonyolult példát mutat mind az egyedi adatokkal dolgozó, mind az átfogó függvények létrehozására és alkalmazására.

Tippek, tanácsok

Az egyik legszembetűnőbb hiányosság a jelenlegi *SQLite* motornak az `ALTER TABLE` megvalósításának teljes hiánya. Meglévő tábláink felépítésének módosítása így nem egy egy sorban kifejezhető hétköznapi feladat. A dolgot kézzel kell megcsinálnunk, és mivel a `RENAME TABLE` is hiányzik a megvalósított műveletek közül, ez csak hat lépésben oldható meg:

- Létrehozunk az eredetivel megegyező felépítésű átmeneti táblát.
- Az eredeti tábla tartalmát átmásoljuk ebbe az átmeneti táblába.
- Töröljük az eredeti táblát.
- Létrehozzuk az eredeti nevén az új táblát, új felépítésében.
- Visszamásoljuk az adatokat az eredeti táblából.
- Töröljük az átmeneti táblát.

Érdemes odafigyelni, hova kerül webszerverünkön az sqlite adatbázisfájl. Hasonlóan ugyanis, mint egy rossz helyre tett *inc* állomány, ez is könnyen rossz kezekbe kerülhet. Érdemes tehát azt olyan helyre tenni, amelyet a webszerver közvetlenül nem tud kifelé közvetíteni, de a *PHP* hozzáfér.
Mint a 2. listában is látható, létrehozható ideiglenes, csak a memóriában élő adatbázis is. Ez ellentétben a *MySQL* `HEAP` típusával szemben a létrehozó program futásának végeztével megszűnik létezni. Ha ilyen átmeneti adatbázisra van szükség, fájlnév helyett adjuk fájlnév helyett `:memory:` értéket megnyitáskor.
Az automatikusan növekvő értékű mezők (mint a *MySQL* `AUTO_INCREMENT`-je, vagy a *PostgreSQL* sorozatai) létrehozása *SQLite*-ban roppant egyszerű. Elég, ha a mező az elsődleges kulcs (`PRIMARY KEY`), típusa pedig számszerű, egész (`INT`). Az ilyen mezőknek `NULL` értéket adva azok időben növekvő, egyedi értéket kapnak.
Az *SQLite* nem rendelkezik külön beállításokat tartalmazó állománnyal, telepítése után azonnal üzemkész. Természetesen egyedi finomításokra, beállítgatásokra sokszor szükség lehet. Az ilyen feladatok elvégzésére szolgál az *SQLite* `PRAGMA` parancsa, mellyel egyedi beállításokat érvényesíthetünk, de ez által gyűjthetők be bizonyos információk is.
A `PRAGMA` parancsok a többi *SQL* kéréssel teljesen azonos mód hívhatók. Néhány olyan, amelyre hamar szükség lehet a mindennapi munka során:

```
PRAGMA table_info(tábla_neve)
```

Az adott tábla szerkezetét kapjuk vissza, minden mezőre megkapva az oszlop nevét, típusát, hordozhat-e `NULL` értéket, valamit az alapértelmezett értékét.

```
PRAGMA cache_size = érték
```

Az alapértelmezett 2000 lapos gyorstár méretet bírálhatjuk felül vele. Egy lap mérete körülbelül 1,5 K. Nagyobb értéket adva nő a memóriafogyasztás, de cserébe gyorsabb működésre bírhatjuk lekérdezéseinket bizonyos esetekben.

```
PRAGMA short_column_names=ON/OFF
```

Ellentétben a korábbiakkal, az aktuális *SQLite* változatok több táblás lekérés esetén a a mező nevet a tábla nevével együtt, ponttal elválasztva adják vissza. A fenti kapcsolót `ON`-ra állítva az *SQLite* rávehető, hogy csakis az oszlopnevet adja vissza.

Heilig Szabolcs

KAPCSOLÓDÓ CÍMEK

Michael Owens: SQL adatbázis beágyazása az SQLite segítségével (Linuxvilág, 2003. augusztus, 26-29. oldal)

➲ http://hu.php.net/sqlite

➲ http://sqlite.org