MIB II.-92/2007/2012.

SZAKDOLGOZAT

KÁKONYI ROLAND

2012

Pécsi Tudományegyetem

Pollack Mihály Műszaki és Informatikai Kar

Mérnök Informatikus Szak

**SZAKDOLGOZAT**

Sugárzásmérési adatok megjelenítése térképen

Készítette: Kákonyi Roland

Témavezető: Zidarics Zoltán

Konzulens: Bertók Attila

Pécs

**2012**

PÉCSI TUDOMÁNYEGYETEM

**Pollack Mihály Műszaki   
ÉS INFORMATIKAI Kar Szakdolgozat száma:**

**Mérnök Informatikus Szak**

**MIB II.-92/2007/2012.**

**SZAKDOLGOZAT FELADAT**

**KÁKONYI ROLAND**

**hallgató részére**

A záróvizsgát megelőzően szakdolgozatot kell benyújtania, amelynek témáját és feladatait az alábbiak szerint határozom meg:

**Téma:** Sugárzásmérési adatok megjelenítése térképen

**Feladat:**

* Mikrokontrolleres adatgyűjtőből származó adathalmaz tárolása
* Adatgyűjtő paraméterezése
* Mért koordinátákhoz tartozó sugárzási érték pontonkénti megjelenítése
* Kvázi ekvivalens pontokra görbe illesztés
* Az így megkapott görbék alapján súlypont (sugárzás forrása) meghatározása

A szakdolgozat készítéséért felelős tanszék: Rendszer és Szoftvertechnológia Tanszék

Külső konzulens: Bertók Attila

munkahelye: Webstar Csoport Kft.

Témavezető: Zidarics Zoltán

munkahelye: PTE-PMMK Automatizálási Tanszék

Pécs, 2011. szeptember 26.

Dr. Szakonyi Lajos

op. szakvezető

HALLGATÓI NYILATKOZAT

Alulírott szigorló hallgató kijelentem, hogy a szakdolgozat saját munkám eredménye. A felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Egyéb jelentősebb segítséget nem vettem igénybe.

Az elkészült szakdolgozatban talált eredményeket a főiskola, a feladatot kiíró intézmény saját céljaira térítés nélkül felhasználhatja.

Pécs, 2011.

|  |  |
| --- | --- |
|  | .................................................  hallgató aláírása |

Köszönetnyilvánítás

Ezúton szeretném megragadni az alkalmat arra, hogy köszönetemet és tiszteletemet fejezzem ki mindenkinek, aki a diplomamunkám elkészítéséhez nagyban hozzájárult.

Szeretném kinyilvánítani köszönetemet Zidarics Zoltánnak, aki szakértelmével, végtelen hosszú türelmével nagyban hozzájárult ahhoz, hogy ez a dolgozat létrejöjjön.

Végül, de nem utolsó sorban szeretném kifejezni megbecsülésemet, szívből jövő szeretetemet és köszönetemet családtagjaimnak és a barátnőmnek, akik szeretetükkel és segítségükkel mindvégig támaszt nyújtottak PTE-s tanulmányaim során. Köszönöm!

# Tartalomjegyzék

[Tartalomjegyzék 6](#_Toc325658236)

[1. Feladat specifikáció 8](#_Toc325658237)

[2. Felhasznált eszközök, technológiák és fejlesztői környezet áttekintése 9](#_Toc325658238)

[2.1. Az alkalmazás technológiai alapjai 9](#_Toc325658239)

[2.2. Általánosan a fejlesztői környezetről 9](#_Toc325658240)

[2.3. Az adatbázis motor 9](#_Toc325658241)

[2.3.1. SQLite bemutatása 9](#_Toc325658242)

[2.3.2. Választás indoklása 10](#_Toc325658243)

[2.3.3. SQLite Manager, mint adatbázis kezelő felület 11](#_Toc325658244)

[2.4. Verziókövető rendszerek 13](#_Toc325658245)

[2.4.1. Verziókövető rendszerek általánosan 13](#_Toc325658246)

[2.4.2. Verziókövető rendszerek típusai 14](#_Toc325658247)

[2.4.3. Verziókövető rendszerek alapfogalmai 15](#_Toc325658248)

[2.4.4. A verziókövető rendszer kiválasztása 16](#_Toc325658249)

[2.4.5. Git-hez használatos eszközök bemutatása 16](#_Toc325658250)

[2.4.6. Git bash 16](#_Toc325658251)

[2.4.7. Gitk 17](#_Toc325658252)

[2.4.8. Git Gui 19](#_Toc325658253)

[2.4.9. Verziókövető használata és előnyei a szakdolgozatom fejlesztése során 19](#_Toc325658254)

[3. Az alkalmazás tervezése 21](#_Toc325658255)

[3.1. A tervezés és a kódolás során alkalmazott módszertanok 21](#_Toc325658256)

[3.1.1. Az Objektum Orientált Programozás alapjai 21](#_Toc325658257)

[3.1.2. Magyar jelölés 22](#_Toc325658258)

[3.1.3. E-K diagram 22](#_Toc325658259)

[3.2. Az adatbázis szerkezete 23](#_Toc325658260)

[3.2.1. Adattáblák fizikai terve 24](#_Toc325658261)

[3.2.2. Adattáblák bemutatása 24](#_Toc325658262)

[3.2.3. Kapcsolatok az adatbázis táblái között 25](#_Toc325658263)

[3.3. Kapcsolat az alkalmazás és az adatbázis között 26](#_Toc325658264)

[3.3.1. Mi az a PDO? 26](#_Toc325658265)

[3.3.2. Használata, legfontosabb metódusai 26](#_Toc325658266)

[3.3.3. Miért választottam? 28](#_Toc325658267)

[4. Az alkalmazás megoldásainak bemutatása 29](#_Toc325658268)

[4.1. Mikrokontrolleres adatgyűjtőből származó adathalmaz tárolása 29](#_Toc325658269)

[4.2. Mért koordinátákhoz tartozó sugárzási érték pontonkénti megjelenítése 29](#_Toc325658270)

[4.3. Kvázi ekvivalens pontokra görbe illesztés 29](#_Toc325658271)

[4.4. Az így megkapott görbék alapján súlypont (sugárzás forrása) meghatározása 29](#_Toc325658272)

[5. Működés bemutatása teszteseteken keresztül 30](#_Toc325658273)

[5.1. Adatok feltöltése fájlból 30](#_Toc325658274)

[5.2. Adatok betöltése adatbázisból 30](#_Toc325658275)

[6. Továbbfejlesztési lehetőségek 31](#_Toc325658276)

[7. Összefoglalás 32](#_Toc325658277)

[8. Mellékletek 33](#_Toc325658278)

[9. Irodalomjegyzék 34](#_Toc325658279)

# 

# Feladat specifikáció

# Felhasznált eszközök, technológiák és fejlesztői környezet áttekintése

## Az alkalmazás technológiai alapjai

Az alkalmazás kódja 2 fő részre osztható, kliens és szerver[1] oldali kódra.

A szerver oldali kódja a PHP[2] programozási nyelv 5.2.17-es verziójában készült, amit egy Apache HTTP Szerver[3] szolgál ki a kapcsolódó klienseknek.

A kliens oldali kód JavaScript nyelven íródott, ezt a felhasználó által használt web böngésző dolgozza fel.

Az alkalmazás megjelenítéséhez HTML5[4] jelölő nyelvet használtam.

A felhasználói élmény növeléséhez kettő izgalmas JavaScript plugin-t használtam,

a jQuery[5] 1.7-es és a jQuery-impromptu[6] 3.2-es verzióját. Ezek segítségével az alkalmazás teljes funkcionalitása kihasználható az oldal újratöltése nélkül.

Az adatok tárolását egy SQLite relációs adatbázis támogatja.

## Általánosan a fejlesztői környezetről

A fejlesztői környezetet egy Windows 7 operációs rendszert futtató számítógépen építettem fel. A kódolást a NetBeans[7] integrált fejlesztői környezet 7.1.2-es verziójában végeztem. Az adatbázis kezelésére, az SQLite Manager nevű Firefox kiegészítőt használtam - ezt az eszközt a későbbiekben részletesen bemutatom. A verziókövetésre a git[8] nevű verziókövető keretrendszert használtam.

## Az adatbázis motor

### SQLite bemutatása

Az SQLite[9] egy beágyazott (embedded) relációs adatbázis kezelő, azaz az őt használó szoftverhez linkelve lehet használni, nem egy külön SQL szerver. Nincs felhasználó adatbázisa. Ettől eltekintve nagyrészt megvalósítja az SQL-92[10] szabványt. Olyan esetekben érdemes megfontolni az alkalmazását, amikor a helyi rendszeren szeretnénk az adatainkat tárolni. Az SQLite-ot neves szoftverek és eszközök használják: MAC OS, Solaris 10, Skype, iPhone, Firefox. A fejlesztők szlogenje szerint használjuk ott ezt a könyvtárat, ahol egyébként a fopen()[11] parancsot használnánk. Gondoljunk arra, hogy egy hordozható eszközhöz írt szoftvernek általában pont ilyen adatbázis kezelőre van szüksége.

Adatbázisok esetén az ACID[12] mozaikszó az Atomicity (atomicitás), Consistency (konzisztencia), Isolation (izoláció), és Durability (tartósság) rövidítése. Ezek az adatbázis kezelő rendszer tranzakció feldolgozó képességeinek alapelemei. Enélkül az adatbázis integritása nem garantálható, így a tranzakció kezelés támogatott ebben a környezetben is.

Részlegesen megvalósítja a triggereket és a legtöbb komplex/összetett lekérdezést. Az SQLite szokatlan típuskezelést használ az SQL adatbázis-kezelőhöz: egy adattípus nem a táblaoszlopaihoz, hanem egyedi értékekhez van hozzárendelve, más szóval dinamikus típuskezelést használ, gyengén típusos adatkezelés mellett. Amennyiben string típusú adat beilleszthető integer oszlopba, akkor a SQLite először a stringet integerre konvertálja, ha az oszlop preferált típusa integer. Ez nagyobb rugalmasságot ad az oszlopoknak, ami hasznos lehet dinamikus típuskezelésű script-nyelvekben való alkalmazás esetén, azonban ez a technika nem vihető át más SQL adatbázis-kezelőkbe. Az SQLite képtelen atipikus adatbázisokban található szigorúan típusos oszlopok kezelésére.

Ugyanazt az adatbázist több processz és szál használhatja egyidejűleg problémamentesen. Az olvasási kérelmek kiszolgálása párhuzamosan történik. Az írási kérelmek végrehajtása akkor történik meg, amikor nincs folyamatban más kérelem kiszolgálása, egyébként az írási kérelem sikertelen lesz és hibakóddal tér vissza, illetve lehetőség van egy beállítható várakozási időelteltével a kérelem ismétlésére. Ez a konkurens hozzáférési állapot megváltozható ideiglenes táblák használata esetén.

Az SQLite számos programnyelvből használható, így BASIC, C, C++, Common Lisp, Java,C#, Visual Basic .NET, Delphi, Curl, Lua, Tcl,R, PHP, Perl, Ruby, Objective-C (Mac OS Xen), Python, newLisp, Haskell, OCaml, Smalltalk és Scheme nyelvekhez rendelkezik illesztő felülettel.

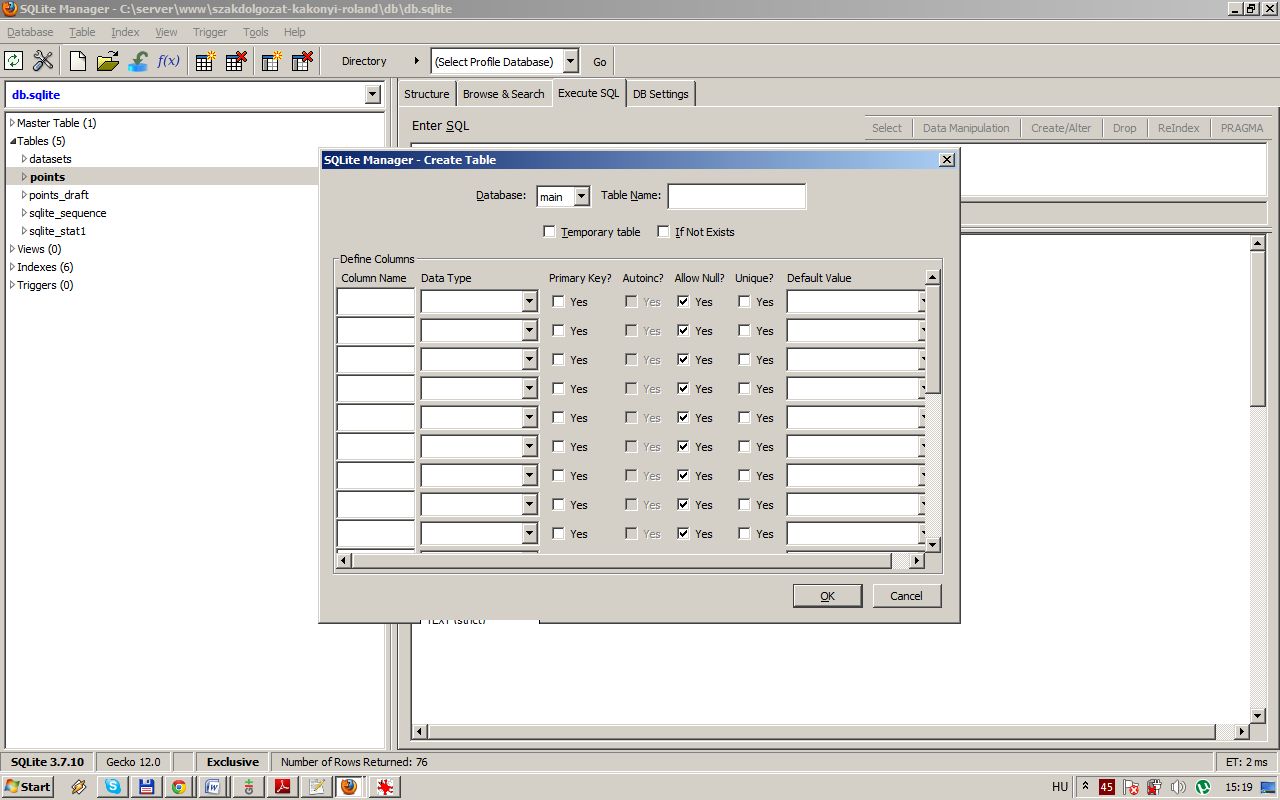
### Választás indoklása

Egy olyan adatbázis motorra volt szükség, amely kis erőforrásigénnyel rendelkezik, de képes a megfelelő szolgáltatások biztosítására. A választás az SQLite adatbázis motorra esett. A használt php interpreterhez elérhető legmagasabb verziószámú, 2.8.17-es SQLite adatbázis motort választottam. Az SQLite választása mellett döntött továbbá a hordozhatósága, egyszerű kezelhetősége, külön szolgáltatástól, szervertől való függetlensége. Továbbá a szakdolgozatom keretein belül fejlesztett alkalmazás teljesítmény és a képesség igényeinek teljesen megfelel.

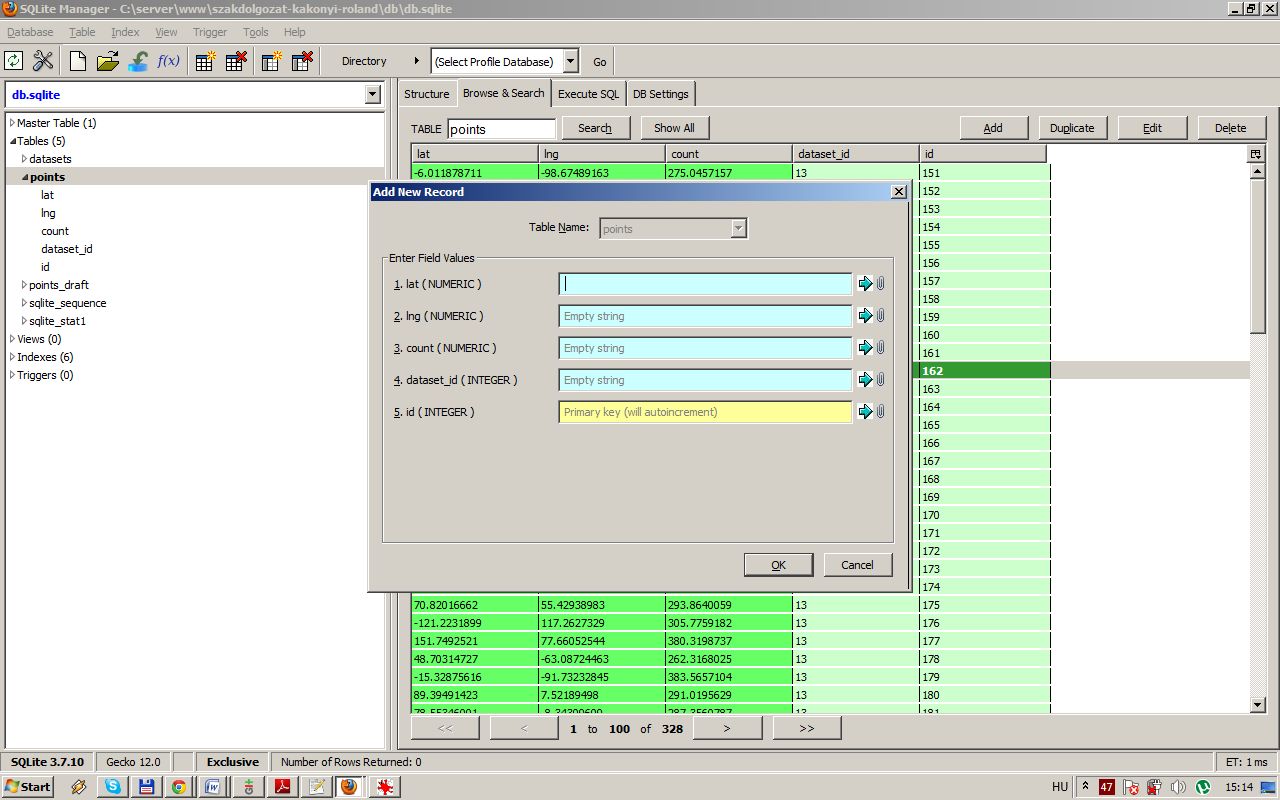
### SQLite Manager, mint adatbázis kezelő felület

A szakdolgozatban bemutatott alkalmazás fejlesztése során a használt adatbázis létrehozására, módosítására, monitorozására és ellenőrzésére az SQLite Manager nevű ingyenes Firefox bővítmény 0.7.7-es változatát használtam [13][14]. Teljes körű támogatást nyújt az SQLite adatbázisok létrehozására és kezelésére.

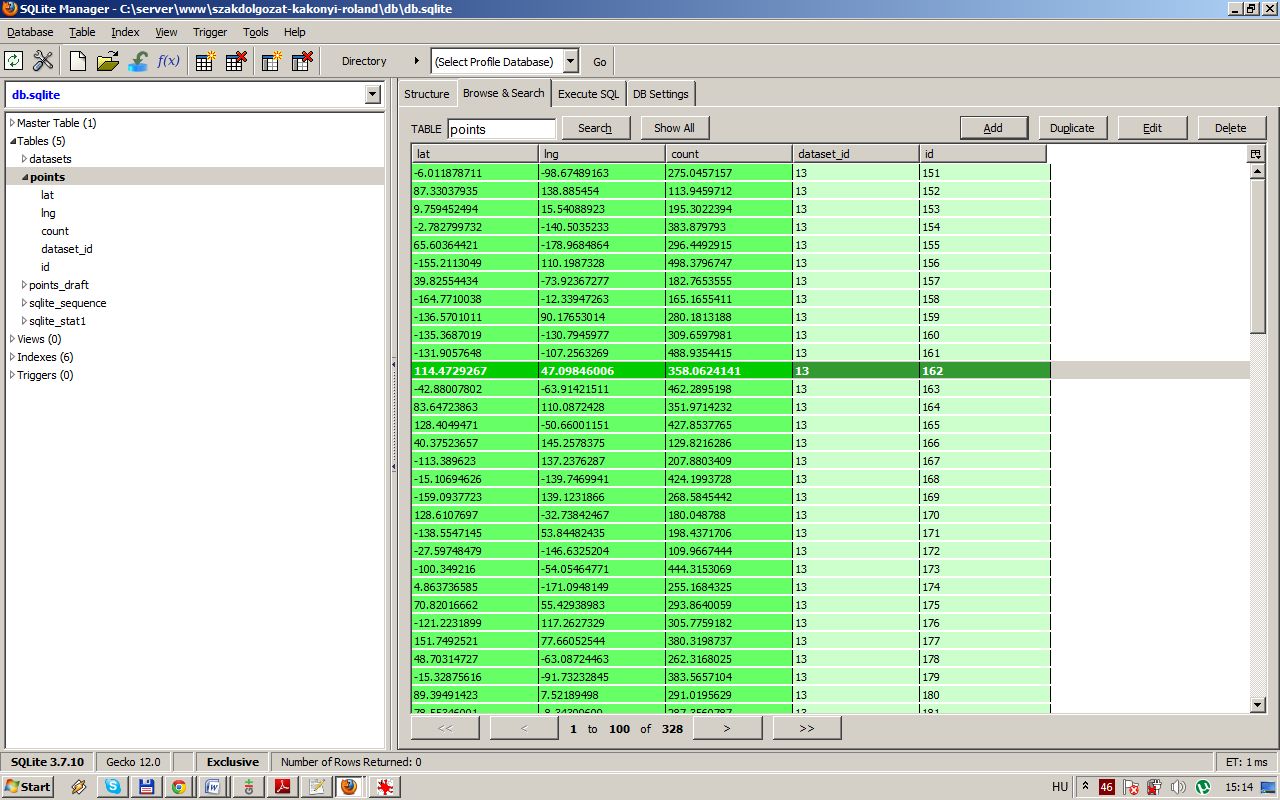
A legfontosabb felületei a következőek:



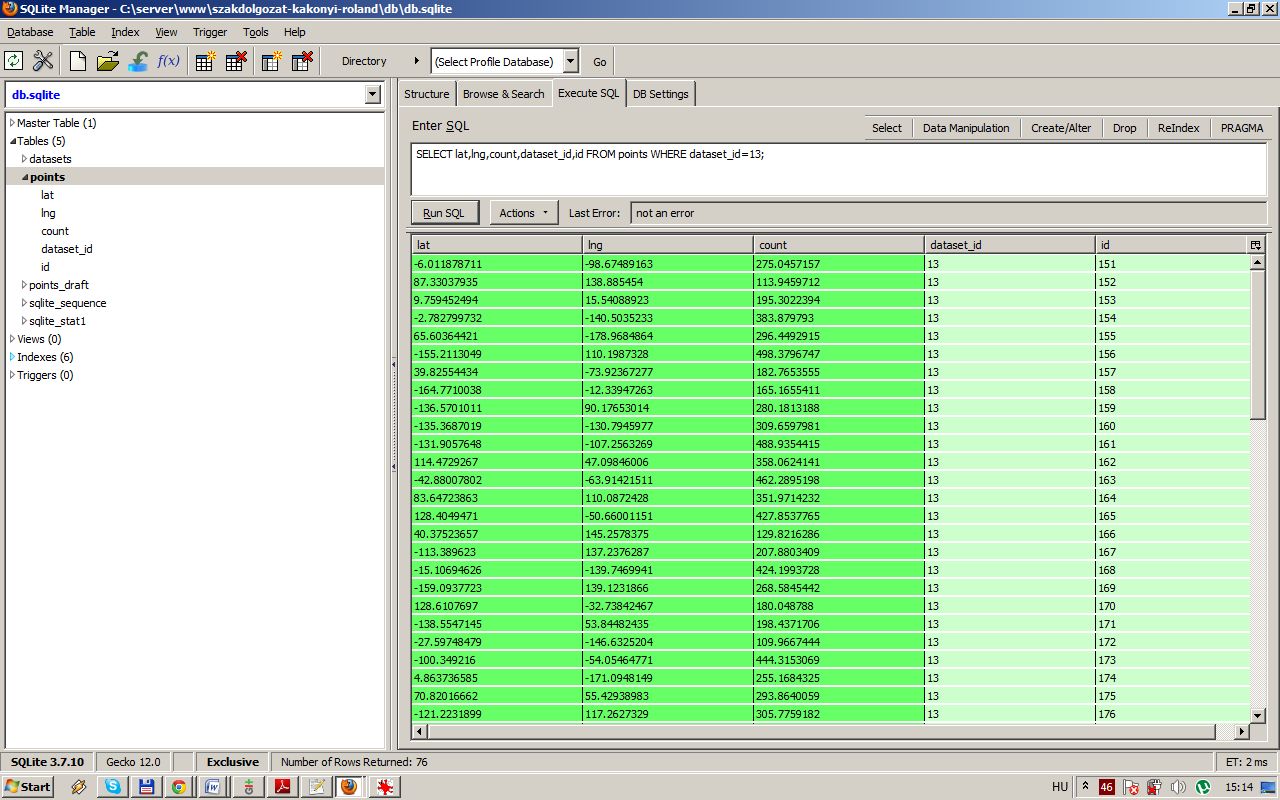
1. ábra Tábla létrehozása



1. ábra Rekord hozzáadása egy táblához



1. ábra Tábla adatainak böngészése



1. ábra Lekérdezés futtatása és annak eredménye

## Verziókövető rendszerek

### Verziókövető rendszerek általánosan

A verziókövető rendszerek [15] arra szolgálnak, hogy a programkódunk minden változatát elmentsék és segítsék a korábbi verziókra való visszatérést és a csoportmunkát. Láthatjuk ki melyik fájlt módosította, mikor, miért, a változtatások pedig könnyen összevethetőek egymással. A szakdolgozatom során egyedül csináltam az alkalmazást, mégis jó hasznát vettem a verziókövetésnek, mert így két gépen is tudtam haladni a fejlesztéssel, attól függően, hogy épp hol voltam, valamint utólag is nyomon tudom követni, mikor mit csináltam.

Csoportmunka során pedig elengedhetetlen a verziókövetés, mert egy remek segítség abban, hogy ne kelljen egymásra várni, hanem egyszerre lehessen dolgozni különböző részeken, sőt, akár ugyan azon a fájlon is.

A verziókövetéshez szükséges egy szerver, ami a verziókezelésbe vont fájlokat tárolja. Első alkalommal feltöltünk egy induló változatot és utána minden egyes újbóli feltöltésnél a rendszer azt fogja eltárolni, hogy melyik fájlok változtak és miben. Ezzel a módszerrel biztosítja azt, hogy lépésenként láthassuk, hogy hogyan változtak a fájlok és bármelyik verzióra vissza tudjunk térni.

A verziókövető rendszerek általában kliens-szerver alapúak: a fejlesztők a szerveren tárolt ún. *repository*-ból jutnak hozzá a fájlokhoz és a változtatásaikat is először a *repository*-ba küldik be (ez a *commit* művelet), a többi fejlesztő már a *repostory*-ból jut hozzá a változtatásokhoz.

### Verziókövető rendszerek típusai

Két nagy csoportra bonthatjuk a verziókövetőket: elosztott és központosított.

A központosított verziókövetők a régebbiek, ma már kevésbé népszerűek, itt ugyanis minden változás csak egy helyen, a szerveren van tárolva. Ilyen például a *Subversion*. [16]

Az elosztott verziókövető azt jelenti, hogy bár van egy központosított szerver, minden felhasználónál megtalálható a központi szerveren található *repository* tulajdonképpeni teljes archívuma (biztonsági mentése). Egy üzemzavar vagy meghibásodás esetén ezek bármelyike visszatölthető a szerverre, hogy lecserélje a központi szerveren levő példányt vagy pótolja hiányát.

Az elosztott verziókövető rendszer további jellemzője, hogy nem igényel állandó, megbízható internet-kapcsolatot a központi szerverrel, és emiatt sokkal kevesebbszer fordul a szerverhez, mint nem elosztott társai. A legtöbb művelet a helyi *repository*-ra hat, így jóval kisebb hálózati forgalmat generál és az átlagos válaszideje is kisebb. A két legnépszerűbb és legelterjedtebb elosztott verziókövető a *Git[8]* és a *Mercurial[17]*.

### Verziókövető rendszerek alapfogalmai

Az alábbiakban a legfontosabb alapfogalmakat és műveleteket ismertetem: [15]

* **Repository** (*tároló*): egy távoli szerver, ezen találhatóak a fájlok
* **Revision** (*változat*): a szerveren található fájlok egy adott változata
* **Commit** (*elkönyvelés*): a helyi változtatások elkönyvelése a továbbításhoz
* **HEAD revision** (*fő változat*): a szerveren található legújabb állapot
* **BASE revision** (*alapváltozat*): a munkakönyvtárban található legfrissebb állapot
* **Differences** (*eltérésmutatás*): két különböző állapot összevetésének naplója
* **Merge** (*összefűzés*): két különböző változat egybefűzése (előfordul, hogy nem sikerül tökéletesen, ilyenkor összetűzés (*conflict*) alakul ki)
* **Conflict** (*összetűzés*): tökéletlen összefűzés (*merge*) során kialakuló helyzet
* **Resolve** (*feloldás*): az összetűzés (*conflict*) megoldása (ilyenkor a munkatárs könyvelés előtt átnézi a konfliktusba került módosításokat és végrehajt egy sikeres összefűzést)
* **Checkout**: a távoli szerver egy adott állapotának lemásolása a gépeden található munkakönyvtárba
* **Working copy** (*munkakönyvtár*): a fájlokat tartalmazó mappa a számítógépeken, ennek tartalma kerül könyvelésre a szerverre
* **Trunk** (*törzs*): a fő fejlesztési ág
* **Branch** (*ág*): a fő ággal és egyéb ágakkal párhuzamosan fejlesztett ágak
* **Tag** (*megjelölt változat*): egy lezárt fejlesztési ág vagy kiadás mappája

### A verziókövető rendszer kiválasztása

A fenti ismertetésből már látszik, hogy mindenképp érdemes elosztott rendszert használni, még akkor is, ha csak egyedül fejlesztünk valamit. Én személy szerint használtam már mind a három fentebb említett verziókövetőt, és a szubjektív véleményem alapján a *Git* használata mellett döntöttem. A *Subversion* azért esett ki a választásból, mert nem elosztott és hálózati kapcsolat nélkül nem használható. A *Git* és a *Mercurial* között az döntött, hogy *Git*-et áthatóbban ismertem meg, valamint gyorsabbnak és gyorsabban tanulhatóbbnak mutatkozott, valamint ennek a kezelése számomra szimpatikusabb és egyszerűbbnek tűnik, mint a *Mercurial* kezelése.

### Git-hez használatos eszközök bemutatása

A Git használatához a Windows-ra elérhető támogató eszközöket tartalmazó programcsomagot telepítettem.[8]

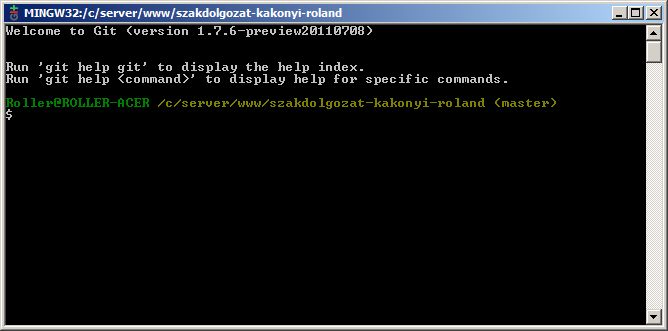
### Git bash

A Git bash-t a 6. ábra mutatja, parancssori eszköz, használata a Linux alapú rendszerekből ismert *BASH*[18] parancsértelmező működése szerint történik.

A Git bash-t használhatjuk a Git Gui vagy a Gitk grafikus felületek elindításához a "git gui" vagy a "gitk" parancsok begépelésével és futtatásával, vagy parancssorból használhatjuk a már említett grafikus felülettel rendelkező segédprogramokban is elérhető funkciókat. A parancsokat a „git” parancsszó után begépelve érhetjük el. A Git bash-ben használható parancsokat a „git help” parancs futtatásával listázhatjuk, ezek rövid ismertetése az alábbi táblázatban található:

|  |  |
| --- | --- |
| add | Fájl hozzáadása a verziókövetéshez |
| branch | A fejlesztési ágak listázását, létrehozását és törlését valósítja meg |
| checkout | Fejlesztési ágak, branchek közötti váltásra szolgál |
| clone | Lemásol egy repository-t egy új könyvtárba |
| commit | Változások elkönyvelése a verziókövezéshez |
| diff | Megmutatja a küönbségeket adott commitok vagy branchek között |
| fetch | Letölti csak a változások listáját a távoli repositoryból |
| grep | Kilistázza a megadott mintára illeszkedő sorokat |
| init | Új repositoryt hoz létre vagy újrainicializál egyet |
| log | Megjeleníti a commit-ok történetét |
| merge | Összevon egy vagy több fejlesztői ágat |
| mv | Átmozgat vagy átnevez egy mappát vagy fájlt |
| pull | Letölt és összefűz egy fejlesztői ágat a helyi változatával |
| push | Elküldi az elkönyvelt változásokat a távoli tárolónak |
| reset | A jelenlegirl visszaáll egy korábbi verzióra |
| rm | Eltávolít egy fájlt a fájlrendszerből és a verziókövetésből |
| status | Megmutatja a jelenlegi állapot változott, de nem commit-olt fájljainak listáját |
| tag | Pillanatnyi állapotról mentést, pillanatképet, úgynevezett címkét készít |

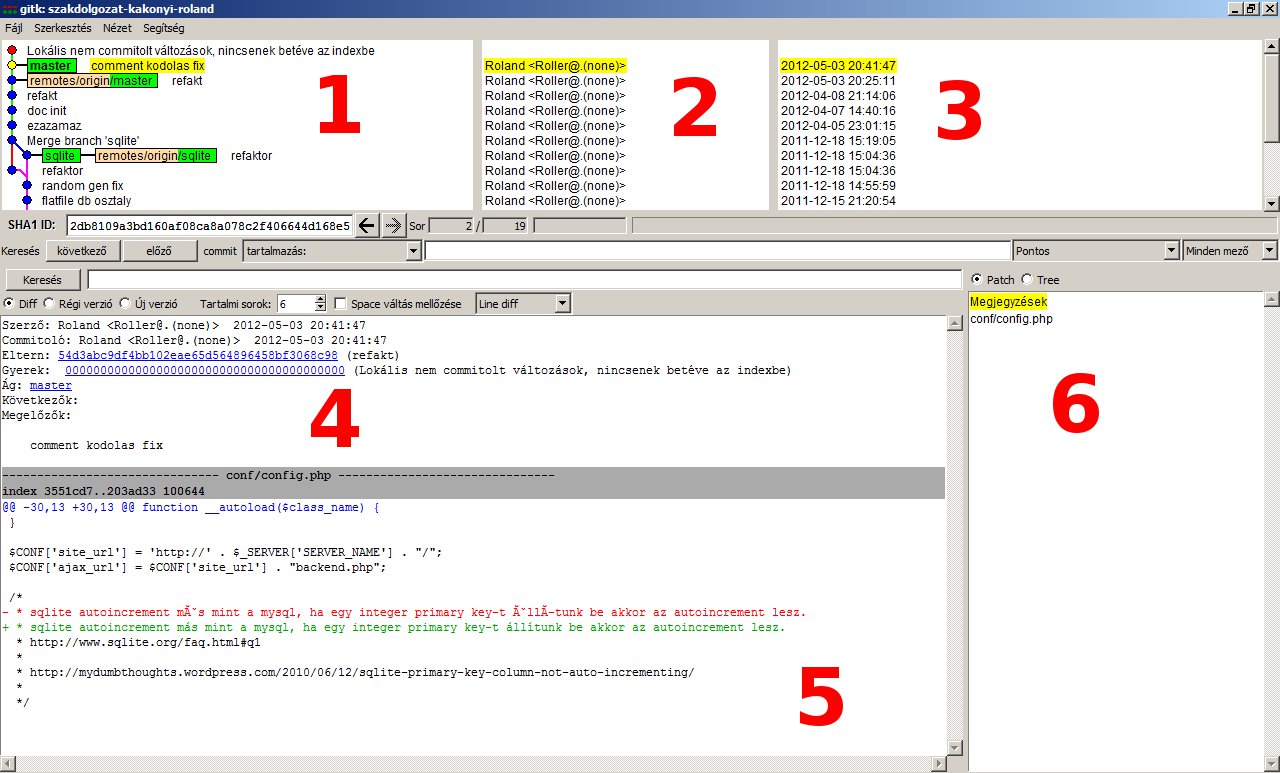
1. táblázat Git bash parancsok



1. ábra Git bash működés közben

### Gitk

A fejlesztési történet vizualizálására szolgál. A verziókövetés során eltárolt változások között böngészhetünk, kereshetünk vagy adott esetben visszatérhetünk a kívánt verzióra.

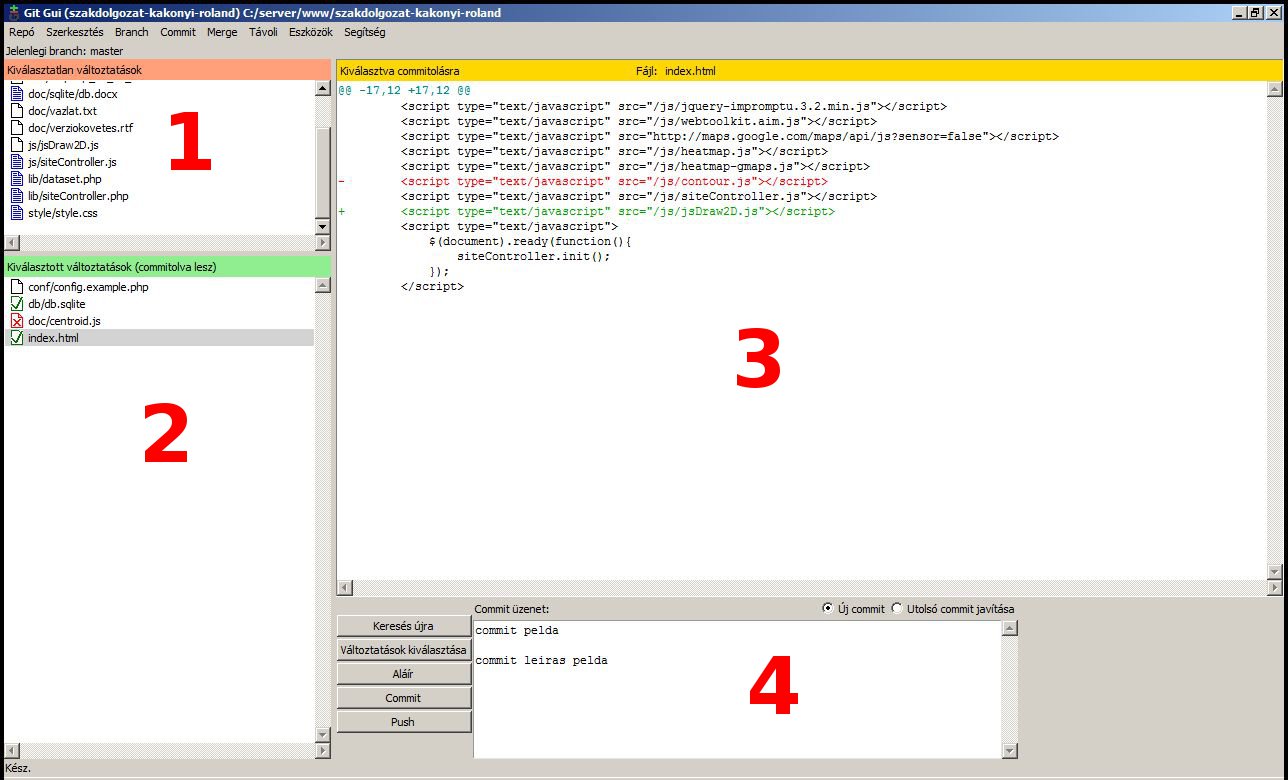


1. ábra Gitk működés közben

A Gitk működését a 16. ábra szemlélteti. A felület részei a következők:

1. Verzió fa: elkönyvelt változások listája fejlesztési áganként
2. Az adott változást beküldő felhasználó neve
3. Az adott változás beküldési ideje
4. A változás részletes adatai, fejlesztési ág, szülő commit, gyerek commit
5. Az adott változásban érintett fájlok tartalma
6. Az adott változásban érintett fájlok listája

### Git Gui



1. ábra Git Gui működés közben

A Git Gui működését a 17. ábra szemlélteti. A felület részei a következők:

1. A legutóbbi commit óta változott fájlok listája
2. A jelenlegi commit-ba beválasztott fájlok listája
3. Az 1. vagy a 2. felületen kiválasztott fájlban történt változások
4. A jelenlegi commit-hoz tartozó üzenet

### Verziókövető használata és előnyei a szakdolgozatom fejlesztése során

Ahhoz, hogy használni tudjuk a verziókövetőt, szükségünk van egy központi szerverre, ahol létrehozhatjuk a repository-t (tárolót). Három népszerű, ingyenes tároló megoldást ismerek, ezek közül választottam ki a fejlesztés során használandót. A *GitHUB* [19] és a *Bitbucket* [20] nagy népszerűségnek örvend, én mégis a *Google code* [21] mellett döntöttem egyszerű, gyors és biztonságos használata miatt. A *Google code* támogatja a Git-et és a Mercurial-t, tehát ebből a szempontból is megfelelő számomra.

Regisztráció után létre tudunk hozni egy repository-t, majd a saját gépünkre le kell *clone*-oznunk (vagyis egy lokális másolatot csinálunk). Innentől kezdve a Git Shell, a Gitk és a Git Gui segítségével végezhetjük a verziókövetést.

Ezek segítségével több lokális ágat (branch-et) tarthatunk fenn, amelyek teljesen függetlenek egymástól. Ezen fejlesztési vonalak létrehozása, egyesítése és törlése csupán másodpercekbe kerül.

Ez azt jelenti, hogy olyan dolgokat tehetünk, mint például:

* Létrehozhatunk egy új ágat, hogy kipróbálhassunk egy új ötletet, néhány változtatás (commit) után visszaválthatunk oda, ahonnan nyitottuk az új ágat, összefésülhetjük az azóta fejlesztett dolgokkal, majd visszaváltunk a kísérletező helyre, és beolvasztjuk.
* Fenntarthatunk egy ágat, amiben mindig csak az éles rendszerbe kerülő dolgok vannak (ezt szokás *master* ágnak hívni), egy másikat, amibe a tesztelésre kerülő munkákat olvasztjuk be, és több kisebbet a mindennapi feladatokra.
* Minden újabb feladatra, amelyen dolgozunk, új ágat hozhatunk létre, hogy gond nélkül váltogathassunk közöttük. Ezek után mindegyiket törölhetjük, amikor az adott dolog beolvasztásra kerül a főágban.
* Létrehozhatunk egy új ágat kísérletezésre; ha esetleg nem jött be a kísérletezés, egyszerűen kitörölhetjük, lemondva az abban végzett munkáról. Ezt az egészet senki más nem látja (akkor sem, ha közben más ágakat feltöltöttünk).
* Amikor egy távoli tárolóba (repositroy-ba) feltöltjük a változásokat, akkor nem kell minden ágat továbbítani, elég csak azt, amin épp dolgoztunk.

A projekteken belül általában vannak olyan fájlok, amiket nem szükséges követni. Ilyenek például a lokális konfigurációs fájlok, amik minden fejlesztőnél mások (pl. adatbázis-kapcsolat, elérési utak, stb.), vagy az ideiglenes létrehozott fájlok. Egy *.gitignore* nevű fájlban beállíthatjuk, hogy ezek a fájlok ne legyenek követve, így mielőtt commit-olnánk, nem is látszódnak, hogy ezek változtak.

Az én szakdolgozatom esetében ilyen nem követett állomány a *config.php*, ami az adatbázis-elérés konfigurációja és az elérési utak beállításait tartalmazza. Helyette egy *config.example* fájl van követve, amiben mindig jelzem a változásokat.

A szakdolgozatomat tartalmazó tároló megtalálható a <http://code.google.com/p/szakdolgozat-kakonyi-roland/> címen.

# Az alkalmazás tervezése

## A tervezés és a kódolás során alkalmazott módszertanok

Az alkalmazás létrehozásának legfontosabb eleme a tervezés, itt a specifikációban megfogalmazott elvárásoknak megfelelő alkalmazás elméleti alapjait kell lefektetni. A gördülékeny fejlesztéshez és hatékony működéshez elengedhetetlenek a tervezés és a kódolás során alkalmazandó különböző programtervezési és programozási módszertanok ismerete.

### Az Objektum Orientált Programozás alapjai

Az objektum-orientált programozás[22] (röviden OOP) a természetes gondolkodást, cselekvést közelítő programozási mód, amely a programozási nyelvek tervezésének természetes fejlődése következtében alakult ki. Az így létrejött nyelv sokkal strukturáltabb, sokkal modulárisabb és absztraktabb, mint egy hagyományos nyelv. Egy OOP nyelvet három fontos dolog jellemez. Ezek a következők:

Az *egységbezárás* (*encapsulation*) azt takarja, hogy az adatstruktúrákat és az adott struktúrájú adatokat kezelő függvényeket kombináljuk; azokat egy egységként kezeljük, és elzárjuk őket a külvilág elől. Az így kapott egységeket *objektumoknak* nevezzük. Az objektumoknak megfelelő tárolási egységek típusát a PHP-ban *osztály*nak (*class*) nevezzük.

Az *öröklés* (*inheritance*) azt jelenti, hogy adott, meglévő osztályokból levezetett újabb osztályok öröklik a definiálásukhoz használt alaposztályok már létező adatstruktúráit és függvényeit. Ugyanakkor újabb tulajdonságokat is definiálhatnak, vagy régieket újraértelmezhetnek. Így egy osztályhierarchiához jutunk.

A *többrétűség* (*polymorphism*) alatt azt értjük, hogy egy adott tevékenység (metódus) azonosítója közös lehet egy adott osztályhierarchián belül, ugyanakkor a hierarchia minden egyes osztályában a tevékenységet végrehajtó függvény megvalósítása az adott osztályra nézve specifikus lehet. Az ún. *virtuális függvények* lehetővé teszik, hogy egy adott metódus konkrét végrehajtási módja csak a program futása során derüljön ki.

Ezek a tulajdonságok együtt azt eredményezik, hogy programkódjaink sokkal strukturáltabbá, könnyebben bővíthetővé, könnyebben karbantarthatóvá válnak, mintha hagyományos, nem OOP technikával írnánk őket.

### Magyar jelölés

A magyar jelölés[23] egy elnevezési megállapodás a számítógép-programozásban. Ebben a jelölésrendszerben egy objektum neve feltünteti a típusát és a tervezett használatát is.

A magyar jelölést a C programozási nyelvben való használatra tervezték, melyben alig van korlátozás a különböző adattípusokkal történő műveletekre, és így a nyelv a véletlen típusokkal kapcsolatos hibáknak erősen ki van téve. A magyar jelölés ezt próbálja orvosolni, mégpedig úgy, hogy a programozónak világos képet ad a változó típusáról.

A magyar jelölésben a változó neve egy vagy több kisbetűvel kezdődik, melyek emlékeztetnek a változó típusára vagy céljára; ezt követi a programozó által választott név. A megadott név nagy kezdőbetűs, hogy el legyen választva a típusjelölőktől.

Eredete: A magyar jelölést Charles Simonyi programozó találta fel, aki korábban a Xeroxnál, majd a Microsoftnál dolgozott, később saját céget alapított. A név Simonyi magyar származásából ered, és így nevezik más nyelveken is (például angolul Hungarian notation).

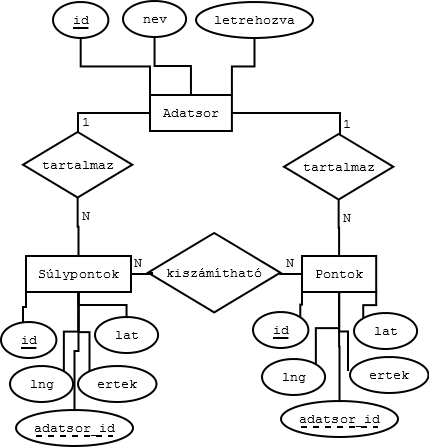
Néhány példa a magyar jelölés használatára:

1. táblázat Magyar jelölés példák

|  |  |
| --- | --- |
| **Változó név** | **Változó tartalmának leírása** |
| $iCount | egész számot tartalmazó változó |
| $oObject | objektumot tartalmazó változó |
| $sSzoveg | szöveges értéket tartalmazó változó |
| $aTomb | tömbösített értékeket tartalmazó változó |

### E-K diagram

Először az E-K, egyedkapcsolati diagram megtervezése a legfontosabb (10. ábra). Ennek segítségével már könnyebben átlátható az egész adatbázis, amire az alkalmazás ráépül.



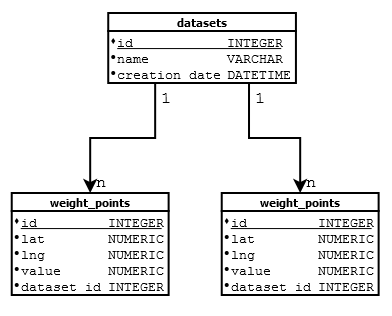
1. ábra E-K diagram

## 

## Az adatbázis szerkezete

Az adatbázis szerkezete lehetővé teszi több adatsor pontjainak és azok súlypontjainak tárolását és visszakeresését.

### Adattáblák fizikai terve



1. ábra Adattáblák fizikai terve

### Adattáblák bemutatása

1. táblázat Adatsor tábla szerkezete

|  |  |  |  |
| --- | --- | --- | --- |
| **Mezőnév** | id | name | creation\_date |
| **Típus** | integer | varchar | datetime |
| **Alapértelmezett érték** | nincs | nincs | datetime() |
| **Lehet null** | nem | nem | nem |
| **Index** | elsődleges kulcs | nincs | csökkenő |
| **Auto increment** | igen | nem | nem |

1. táblázat Pontok tábla szerkezete

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Mezőnév** | id | lat | lng | value | dataset\_id |
| **Típus** | integer | numeric | numeric | numeric | integer |
| **Alapértelmezett érték** | nincs | nincs | nincs | nincs | nincs |
| **Lehet null** | nem | nem | nem | nem | nem |
| **Index** | elsődleges kulcs | nincs | nincs | nincs | nincs |
| **Auto increment** | igen | nem | nem | nem | nem |

1. táblázat Súlypontok tábla szerkezete

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Mezőnév** | id | lat | lng | value | dataset\_id |
| **Típus** | integer | numeric | numeric | numeric | integer |
| **Alapértelmezett érték** | nincs | nincs | nincs | nincs | nincs |
| **Lehet null** | nem | nem | nem | nem | nem |
| **Index** | elsődleges kulcs | nincs | nincs | nincs | nincs |
| **Auto increment** | igen | nem | nem | nem | nem |

### Kapcsolatok az adatbázis táblái között

Az adatbázis táblái – egyedei közötti viszonyt jelölik a kapcsolatok, ezzel kiegészülve az E-K diagram, szemlélteti az adatbázis teljes struktúráját.

**Adatsorok-pontok kapcsolat:** Az „adatsorok” tábla minden egyes rekordja egy a többhöz kapcsolatban áll a „pontok” tábla bejegyzéseivel. Ez a kapcsolat biztosítja, hogy az egy adathalmazhoz tartozó pontokat be tudjuk azonosítani és csoportosítani. A kapcsolatot az „adatsorok” tábla „id” mezője és a „pontok” tábla „dataset\_id” mezőinek egyezése határozza meg.

**Adatsorok-súlypontok kapcsolat:** Az „adatsorok” tábla rekordjai egy a többhöz kapcsolatban áll a „súlypontok” tábla bejegyzéseivel is. Ez a kapcsolat biztosítja, hogy az egy adathalmazhoz tartozó pontokból kiszámított súlypontokat be tudjuk azonosítani és csoportosítani. A kapcsolatot az „adatsorok” tábla „id” mezője és a „súlypontok” tábla „dataset\_id” mezőinek egyezése határozza meg.

**Pontok-súlypontok kapcsolat:** Az „pontok” tábla rekordjai több a többhöz kapcsolatban állnak a „súlypontok” tábla bejegyzéseivel. Ez a kapcsolat biztosítja, hogy az egy adathalmazhoz tartozó pontokat és az azok feldolgozásából kiszámított súlypontokat össze tudjuk rendelni. A kapcsolatot az „pontok” tábla „dataset\_id” mezője és a „súlypontok” tábla „dataset\_id” mezőinek egyezése, tehát az azonos adatsorhoz való kapcsolat jellemzi.

## Kapcsolat az alkalmazás és az adatbázis között

Meg kell teremtenünk az alkalmazás és az adatbázis közötti kapcsolatot, erre a PHP, az egységes adatbázis kapcsolatok kezelésére létrehozott kiterjesztését, a PHP PDO kiterjesztés lehetőségeit használtam ki.

### Mi az a PDO?

A PHP Data Objects[24] egy PHP 5 kiterjesztés neve, mely C kód formájában oldja meg az adatbázis absztrakció problémáját, olyan szolgáltatásokat nyújtva, mint a tranzakciók kezelése, előkészített SQL[10] parancsok futtatása, paraméterezése, automatikus típus szerinti *escapelés* (az SQL Injection[25] támadások ellen) és így tovább. Ez az egységesített interfész lehetővé teszi például, hogy rendszerünket SQLite alapokról MySQL[26] alapokra vigyük át egyetlen sor átírásával.

A PDO nem más, mint egy objektum az adatbázis kapcsolatok, lekérdezések kényelmes, hatékony, átlátható kezelésére.

### Használata, legfontosabb metódusai

Használata egyszerű, gyorsan tanulható, a kódot újrahasznosíthatóvá, hordozhatóvá teszi [27].

Kapcsolat inicializálása:

$db = new PDO('mysql:host=localhost;dbname=database', 'user', 'password');

A kapcsolat létrehozásakor példányosított objektum metódusain keresztül érjük el az adatbázisunkat.

A legfontosabb metódusok:

Egyszerű lekérdezés:

A *query* metódussal egyszerűen futtathatunk lekérdezéseket.

$db->query(" SELECT \* FROM `table` ");

Tranzakciók:

A tranzakciók lényege, hogy egy blokkban vagy mindegyik SQL utasítás végrehajtódik, vagy pedig egyik sem. A legjobban ezt egy banki átutalásos példával lehet szemléltetni, ahol az egyik lekérdezés leveszi "A" ügyfél számlájáról a pénzt, a második pedig rárakja "B" számlájára. Ez esetben azt szeretnénk, hogy ha nem sikerül az első lépés, akkor a második se hajtódjon végre.

try {

$db = new PDO('mysql:host=localhost;dbname=database', 'user', 'password');

$db->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);

$db->beginTransaction();

$db->exec(" UPDATE `accounts` SET `account\_value`=`account\_value`-100 WHERE `account\_id`='1' ");

$db->exec(" UPDATE `accounts` SET `account\_value`=`account\_value`+100 WHERE `account\_id`='2' ");

$db->commit();

} catch (PDOException $e) {

$db->rollBack();

}

A példában használt metódusok:

setAttribute:

Az adatbázis kezelő objektum tulajdonságait tudjuk vele beállítani különböző, a PDO objektumban tárolt konstansokkal. Ez esetben az lett beállítva, hogy - az alapértelmezettől eltérően - kivételt dobjon hiba esetén.

beginTransaction: Jelzi, a tranzakció kezdetét.

commit: A tranzakció lefuttatása.

rollBack: A tranzakció visszavonása.

Előkészített lekérdezés:

Ennek előnye akkor mutatkozik, amikor egy lekérdezést többször kell futtatnunk különböző paraméterekkel. Ilyenkor a *prepare* metódussal előkészítjük a lekérdezést, a paraméterek helyére egy-egy tetszőleges nevet írva, majd az *execute* metódusnak megadjuk a paraméter név-érték párokat asszociatív tömbként.

$pre = $db->prepare(' INSERT INTO "attrs" ("id", "name") VALUES (:id, :name) ');

$pre->execute(array(':id' => '1', ':name' => '12'));

$pre->execute(array(':id' => '2', ':name' => '43'));

Kapcsolat lezárása:

$db = NULL;

### Miért választottam?

A választásomat a PDO objektum orientáltsága, ezáltal gyors, könnyed kezelése, valamint hasonló képességű konkurens megoldás hiánya indokolja. Működési feltétele csupán a PHP PDO kiterjesztésének és a kiválasztott adatbázis motort támogató PHP kiterjesztés, esetemben SQLite támogatás, a php\_sqlite kiterjesztés telepítése és engedélyezése.

# Az alkalmazás megoldásainak bemutatása

## Mikrokontrolleres adatgyűjtőből származó adathalmaz tárolása

Az adatok feltöltéséhez az adatsor nevét és a feltöltendő adatokat tartalmazó fájlt kell megadnunk. A betölteni kívánt adatokat az alkalmazás CSV[28] formátumból képes beolvasni és eltárolni. A CSV formátum jellegzetessége, hogy az összetartozó mezőket vesszővel (,), a sorokat sortöréssel választva tárolja. Ehhez igazodva dolgozza át az alkalmazás a beolvasott sorokat az adatbázisba való beszúráshoz. A feltöltött fájlt csak ideiglenesen tároljuk, a tartalma a későbbiekben az adatbázisból visszanyerhető.

Az adatsor nevének beszúrása után a pontok adatait (feltöltött fájl sorait) egyesével átalakítva szúrja be az adatbázisba az adatsor azonosítójával együtt - ügyelve a mezők szövegből lebegőpontos értékké való átalakításra. Beszúrás előtt ellenőrizzük az összes érték meglétét (szélesség, hosszúság, érték), ezek nélkül a pont adatai nem értelmezhetőek.

### Adatsor beszúrása

try {

$oSth = Db::getInstance()->prepare("INSERT INTO datasets (name,creation\_date) VALUES (:name,:creation\_date)");

$oSth->bindParam(':name', $this->sName);

$oSth->execute();

$this->iId = Db::getInstance()->lastInsertId();

}

catch (PDOException $e) {

die();

}

### Adatsorhoz tartozó pontok beszúrása

if (($handle = fopen($sSource\_file, "r")) !== FALSE) {

try {

$sQuery = "

INSERT INTO points\_draft (lat,lng,count,dataset\_id)

VALUES (:lat,:lng,:count,:dataset\_id);";

$oSth = db::getInstance()->prepare($sQuery);

$oSth->bindParam(':dataset\_id', $this->iId);

$oSth->bindParam(':lat', $fLat);

$oSth->bindParam(':lng', $fLng);

$oSth->bindParam(':count', $fCount);

}

catch (PDOException $oE) {

die();

}

while (($aRow = fgetcsv($handle, $iMax\_line\_length, $sDelimiter, $sEnclosure)) !== FALSE) {

try {

$fLat = floatval($aRow[0]);

$fLng = floatval($aRow[1]);

$fCount = floatval($aRow[2]);

if (isset($fLat) && isset($fLng) && isset($fCount)) {

$oSth->execute();

}

}

catch (PDOException $oE) {

@fclose($handle);

die();

}

}

fclose($handle);

}

Ahogy azt a fenti két kódrészletből láthatjuk, az adatbázisba való beszúrás során kihasználtam a PHP PDO objektumainak képességeit, mint az előkészített lekérdezés és a hibakezelés kivételekkel.

## Mért koordinátákhoz tartozó sugárzási érték pontonkénti megjelenítése

## Kvázi ekvivalens pontokra görbe illesztés

## Az így megkapott görbék alapján súlypont (sugárzás forrása) meghatározása

# Működés bemutatása teszteseteken keresztül

## Adatok feltöltése fájlból

## Adatok betöltése adatbázisból

# Továbbfejlesztési lehetőségek

# Összefoglalás

# Mellékletek

# Irodalomjegyzék

[1] How Web Servers Work - Clients and Servers [Online]

<http://computer.howstuffworks.com/web-server4.htm> (látogatva: 2011. december 10.)

[2] PHP [Online] <http://php.net/> (látogatva: 2011. december 10.)

[3] APACHE HTTP SZERVER [Online] <http://httpd.apache.org/>

(látogatva: 2011. december 10.)

[4] HTML5 [Online] <http://html5.org/> (látogatva: 2011. december 10.)

[5] JQUERY [Online] <http://jquery.com/> (látogatva: 2011. december 10.)

[6] JQUERY IMPROMPTU [Online] <http://trentrichardson.com/Impromptu/>

(látogatva: 2011. december 10.)

[7] NETBEANS [Online] <http://netbeans.org/> (látogatva: 2011. december 10.)

[8] GIT [Online] http://git-scm.com/ (látogatva: 2011. december 10.)

[9] SQLITE [Online] <http://www.sqlite.org/> (látogatva: 2011. december 10.)

[10] ISO/IEC 9075:1992, Database Language SQL [Online]

<http://www.contrib.andrew.cmu.edu/~shadow/sql/sql1992.txt> (látogatva: 2011. december 10.)

[11] PHP FUNCTION FOPEN [Online] <http://php.net/manual/en/function.fopen.php> (látogatva: 2011. december 10.)

[12] Atomic Consistent Isolated Durable [Online]

<http://c2.com/cgi/wiki?AtomicConsistentIsolatedDurable> (látogatva: 2011. december 10.)

[13] sqlite-manager FIREFOX ADD-ON [Online]

http://code.google.com/p/sqlite-manager/ (látogatva: 2011. december 10.)

[14] sqlite-manager FIREFOX ADD-ON [Online]

<https://addons.mozilla.org/hu/firefox/addon/sqlite-manager/> (látogatva: 2011. december 10.)

[15] Verziókövetés [Online]

<http://logout.hu/cikk/verziokovetes/a_verziokovetes_lenyege.html> (látogatva: 2011. december 10.)

[16] VERSION CONTROL WITH SUBVERSION [Online] <http://svnbook.red-bean.com/> (látogatva: 2011. december 10.)

[17] MERCURIAL [Online] <http://mercurial.selenic.com/> (látogatva: 2011. december 10.)

[18] bash - GNU Bourne-Again Shell [Online] <http://linux.die.net/man/1/bash> (látogatva: 2011. december 10.)

[19] GITHUB [Online] <https://github.com/> (látogatva: 2011. december 10.)

[20] ATLASSIAN BITBUCKET [Online] <https://bitbucket.org/> (látogatva: 2011. december 10.)

[21] GOOGLE CODE [Online] <http://code.google.com/> (látogatva: 2011. december 10.)

[22] Angster, Erzsébet: Az objektumorientált tervezés és programozás alapjai (magánkiadás 1999; ISBN 9636508186)

[23] Hungarian Notation [Online]

<http://msdn.microsoft.com/en-us/library/aa260976(VS.60).aspx>

(látogatva: 2011. december 10.)

[24] PHP Data Objects [Online] <http://hu.php.net/pdo>

(látogatva: 2011. december 10.)

[25] Weboldalak biztonsága 1: SQL Injection [Online]

<http://pezia.hu/content/2009/03/08/weboldalak_biztons%C3%A1ga_1_sql_injection> (látogatva: 2011. december 10.)

[26] MYSQL [Online] <http://www.mysql.com/> (látogatva: 2011. december 10.)

[27] Introduction to PHP PDO [Online]

<http://www.phpro.org/tutorials/Introduction-to-PHP-PDO.html>

(látogatva: 2011. december 10.)

[28]

[29]

[30]

[31]

[32]

[33]

[34]

[35]