Blahota István

SQLite alapok

oktatási segédanyag, mely a

Társadalmi Megújulás Operatív Program
Határon átnyúló együttműködés a szakképzés és a felnőttképzés területén c. pályázati felhívás keretében megvalósított

Mobil alkalmazásfejlesztés az informatikai tudás innovatív alkalmazásával című, ***TÁMOP-2.2.4-11/1-2012-0055*** kódszámú projekt keretében valósult meg.

2013.

A projektek az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósulnak meg.

Tartalomjegyzék

1. A jegyzetről

Ez a kiadvány a „Mobil platformok adatbázis-kezelése” című kurzushoz készült, elméleti áttekintésként. Tartozik hozzá egy kidolgozott feladatokból álló problémagyűjtemény is.

A jegyzet koncepciója, hogy minimális informatikai ismereteket feltételezve is bevezetést kapjunk az adatbázis-kezelésbe, kihangsúlyozva azon területeket, melyek fő témánk, az SQLite adatbázis-kezelő használatához szükségesek, illetve megismerkedjünk az SQLite jellegzetességeivel, alapvető használatával.

A jegyzetet lektorálta Faragó János fejlesztésvezető, rendszermérnök, az Andrews IT Engineering Kft. munkatársa.

Az oktatási segédanyag a *„Mobil alkalmazásfejlesztés az informatikai tudás innovatív alkalmazásával”* című, *TÁMOP-2.2.4-11/1-2012-0055* kódszámú projekt keretében valósult meg.
MAGYARORSZÁG MEGÚJUL
A projektek az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósulnak meg.

2. Bevezetés

Amennyiben **adatbázis**okon valamilyen rendszer alapján tárolt adatokat értünk, úgy az adatbázisok története egyidős az emberiség (legalábbis írott) kultúrájával. Bár legtöbb esetben akkor használunk adatbázist, ha adataink nagy száma azt megköveteli, nagy számú adat halmaza még nem adatbázis – a lényeg a strukturáltságon van. Az emberek hamar rájöttek arra, hogy ha sok adatot kell elraktározniuk, – és amennyiben ezeket a későbbiekben hatékonyan akarják felhasználni – elkerülhetetlen, hogy ne csak „ömlesztve", kategorizálás nélkül tárolják azokat, hanem bizonyos rendszerező elvek alapján. Ezek az elvek eleinte nyilván ad-hoc jellegűek voltak, a későbbiekben viszont hamar körvonalazódni kezdtek azon alapvető eljárások, melyek mentén az adatbázisok elmélete kialakult.

Az igazi áttörést a digitális számítógépek kifejlesztése hozta meg, bár a számítógép közvetlen elődjének tartott mechanikus és lyukkártya rendszerű gépek már a XIX. század végétől egyre fontos szerepet játszottak a – többek között népszámlálási adatok feldolgozása során keletkező – nagy mennyiségű adat hatékony feldolgozásában. A mára kizárólagossá vált Neumann-elvű digitális számítógépek gazdaságos felhasználását eleinte szinte kivétel nélkül az ilyen típusú feladatok megoldása jelentette.

A kezdetekben az adatokat célgépekkel dolgozták fel, de az első digitális számítógépek is még mechanikus, lyukkártyás, lyukszalagos adattárolást használtak. Az adatfeldolgozás sebessége sokat javult, amikor általánossá váltak az elektronikus háttértárak. Szintén nagy jelentősége volt a programozási nyelvek fejlődésének. Az első univerzálisnak szánt programok csak a magas szintű programozási nyelvek elterjedésével születhettek meg. Ráadásul az adatbázisok gyors elterjedése és széleskörűvé válása egyre elavultabbá tette az adatbázis programok fejlesztésének egyedi módját. Létrejöttek az adatbázis kezelő rendszerek és az úgynevezett negyedik generációs nyelvek. Ezek számos olyan eszközt nyújtanak (például az interaktív adatbevitel vagy a menük létrehozása terén), melyek előállítása a korábbi programozási nyelvekben csak igen hosszadalmas programozói munkával volt lehetséges. Az eszközök szabványossá válása egyrészt segítette a programozói munkát,

másrészt támogatta az egységes (de legalábbis hasonló elveket megvalósító) felhasználói felület kialakulásának folyamatát.

Az informatikában mára szinte minden adattárolást adatbázisokkal valósítunk meg. Adatainkat kategorizálva tároljuk, de tárolnunk kell az adatok közti kapcsolatokat is. Megfelelő esetben a tárolás alapelveit még az adatfeltöltés megkezdése előtt véglegesítjük. Amennyiben szükséges, a tárolt adatokat módosíthatjuk, törölhetjük. A tárolt adatokból új ismeretekhez, információhoz juthatunk. Ezek az **adatbázis-kezelés** alapvető elemei.

Maguk az adatok bárhonnan származhatnak, akár természeti, gazdasági, vagy társadalmi folyamatok vizsgálatából. Természetesen sem az adatbázisok elmélete, sem gyakorlata szempontjából nincs jelentősége az adatok eredetének.

3. Az adatbázis-kezelés alapjai

Az adatbázisok és az adatbázis-kezelés elmélete mára külön tudománnyá nőtte ki magát. A különböző céloknak megfelelően szerteágazó elmélet alakult ki, a gyakorlati megvalósítások száma ennek megfelelően nem kevés. Az alapelvek mára kiforrottak, az átlagos feladatok megoldása során használt eszközöket sok esetben viszonylag szűk körből választják. Ennek a munkának a keretében nem fogunk túlságosan elkalandozni, célunk az SQLite eszköz és a használatához szükséges elméleti megalapozás bemutatása. Az alapok megismeréséhez egy egyszerű példa kidolgozásán keresztül jutunk majd el.

Sorrendben az első feladat az adatbázis megtervezése. A tervezés során akkor van a legegyszerűbb dolgunk, ha az adataink azonos struktúrájúak. Példaként tegyük fel, hogy az otthonunkban található könyveket szeretnénk nyilvántartani. Mivel egy átlagos lakásban legfeljebb pár száz könyv található, úgy gondolhatjuk, hogy ez nem lehet nagy feladat. A probléma kissé árnyaltabb ennél. Az adatbázis megtervezésének összetettsége nem függ szorosan az adatbázis méretétől. Természetesen van az az elemszám, ami alatt nem érdemes túlbonyolítani az adatbázis szerkezetét. Valójában a struktúra összetettségét az adatfeldolgozással kapcsolatos igények határozzák meg. De hogy egy adatbázisban néhány ezer, vagy néhány tízezer könyv adatait tároljuk, az a struktúra szempontjából mindegy. Persze a túl nagyra nőtt adatbázis számos problémát felvethet. A nagyon nagy adatbázisok tervezése és kezelése komoly és sokrétű szakértelmet és tapasztalatot kíván, de ilyenekkel nem fogunk foglalkozni. Az SQLite jellemzően nem ilyen célokra készült.

Kezdjük egy egyszerű adattárolási móddal. Vegyük számba, hogy milyen azonosítói vannak egy könyvnek. A legfontosabb a könyv címe. De szinte biztosan számon akarjuk tartani a könyv szerzőjét, szerzőit is. Erre már csak azért is szükség van, mert lehetnek azonos című könyveink is. Nekem például több „Matematika” című könyvem is van. Néhány könyvnek alcíme is van, ez is segítheti a beazonosításukat. Például Dr. Gerőcs László „Matematika” című könyvének az alcíme „30 feladatsor a matematikából felvételizőknek”, míg Obádovics József Gyula azonos című könyvének „Középiskolai, technikumi tanulók, egyetemi hallhatók

és technikusok számára gyakorlati alkalmazásokkal”. Gyakori eset azonban, hogy egy könyvnek nincs alcíme. Ez azért érdekes, mert akkor lenne a legegyszerűbb dolgunk, ha minden könyvről pontosan ugyanazokat az adatokat tartanánk számon. Ez azonban sok esetben nincs így. Nyilvánvalóan előre meg kell terveznünk, hogy mely adatok megadását várjuk el kötelezően, melyekét nem. Természetesen legalább egy kötelező adatnak kell lennie. Jelen esetben a cím elég logikus választás. Már a szerző sem adható meg mindig, gondoljunk például a Bibliára.

Táblázatunkban szükségünk lenne még egy olyan oszlopra is, ami alapján egyértelműen be tudjuk azonosítani a sorokat (láttuk ugyanis, hogy a cím nem alkalmas erre). Ezt az oszlopot **elsődleges kulcs**nak nevezzük. Mivel ilyen tulajdonságú oszlopot jelen állapotában a táblázatunk még nem tartalmaz, vegyünk fel egy „id” (azonosító) nevűt. Értékeinek természetes számokat vettem fel.

Vegyünk fel még egy oszlopot, melyben a könyv típusát tároljuk. A típusként valójában nem a tényleges típus elnevezését (például hogy szakkönyv, tudományos-fantasztikus, szépirodalom, ismeretterjesztő, stb.) tüntetjük, fel, hanem csak egy egyértelmű azonosítót. Ha ugyanis mindenhová beírnánk ezen típus megjelölő kifejezések valamelyikét, az adatismétlődés (**redundancia**), vagyis helypazarlás lenne, ami mindenképpen elkerülendő. (Az adatbázis optimalizálására használjuk az úgynevezett normál formákat, lásd például http://en.wikipedia.org/wiki/Database_normalization vagy http://support.microsoft.com/kb/283878/hu.)

Ez így már egy egyszerű, de használható adatbázis szerkezet. (Valójában már ez sem a legkisebb értelmes rendszer, hiszen sem az alcím, sem a típus tárolása nem lenne életbevágó.) A kategóriákat (a táblázat oszlopait) mezőknek, az „id”, „szerzok”, a „cim” és „alcim” kifejezéseket mezőneveknek, a konkrét adatokat, vagyis az egyes könyveket a róluk tárolt információkkal (a táblázat sorait) rekordoknak nevezzük.

A rekordok nevében nem szerepeltettünk ékezetes karaktereket. Egyes adatbázis-kezelők megengedik ezt, de a magam részéről ekkor sem

javaslom használatukat. Fölösleges bonyodalmaktól kímélhetjük meg magunkat, ha csak az angol ABC betűit használjuk egyéb karakterek (például szóköz, ékezetes betűk) nélkül.

Magát a táblázatot adattáblának, vagy gyakran röviden csak táblának fogjuk hívni. Jelen esetben (most még) az adatbázisunk ebből az egy táblából áll. Ez a tábla öt mezőt tartalmaz. A tábla neve legyen „konyvek”.

id	szerzok	cim	alcim	tipus
1	Dr. Gerőcs László	Matematika	30 feladatsor a matematikából felvételizőknek	1
2	Obádovics József Gyula	Matematika	Középiskolai, technikumi tanulók, egyetemi hallgatók és technikusok számára gyakorlati alkalmazásokkal	1
3	-	Biblia	-	3

1. táblázat: Egy egyszerű adatbázis-séma, néhány adattal

Amennyiben ennyi mező tárolásával megelégszünk, elkezdhetjük feltölteni az adatbázist adatokkal. Gyakorlatilag már hozzá is láttunk kitölteni egy táblázatot. Eddig három rekordot vettünk fel (lásd az 1. táblázatot).

Láttuk, hogy helytakarékossági okok miatt nem érdemes a típusok teljes nevét mindig kiírni. Viszont valahogy tudnunk kellene, hogy melyik kódszám melyik típusnak felel meg. E célból érdemes egy újabb táblát felvennünk, amely esetleg csak annyit tartalmaz, hogy a típusnak, mind kódnak melyik típusnév felel meg (lásd a 2. táblázatot). A második táblánk neve legyen „tipusok”.

tipus	tipusnev
1	szakkönyv
2	szépirodalom
3	egyéb

2. táblázat: Típuskódok és -nevek táblája

A „tipusok” táblában is kell lennie elsődleges kulcsnak, erre a „tipus” mező most alkalmas. Két táblánk nyilván összetartozik, a kapcsolódó mező a két tábla „tipus” mezője. Úgy mondjuk, hogy a „konyvek” tábla „tipus” mezője **idegen kulcs** (hiszen egy másik tábla elsődleges kulcsának mása). Mivel egy könyvnek csak egy típusba való tartalmazást engedünk meg, de egy típus több könyvhöz is tartozhat, azt mondjuk, hogy az ilyen kapcsolat egy-a-többhez **típusú.** Jelen esetben ez némiképp mesterséges leegyszerűsítése a helyzetnek, hiszen (bár nem gyakran, de) előfordulhat, hogy egy könyv egyszerre több típusúnak is tekinthető. Ha mégsem maradunk az egyszerű modellnél, akkor szükségünk lesz a több-a-többhöz **típus** használatára. Ennek gyakorlati megvalósításához úgynevezett kapcsolótáblát kell használnunk (erre majd a gyakorlati részben látunk majd példát).

Ha nem is gyakran, de használhatunk **egy-az-egyhez típusú** kapcsolatokat is. Erre akkor lehet szükség, ha mesterségesen el akarunk szeparálni valamilyen adatot a többitől. Például ha külön táblában tüntetünk fel valamilyen bizalmas adatot, és ezt a táblát más biztonsági előírásoknak vetjük alá (például fokozottabban védjük). Az SQLite esetén ez nem életszerű, nem is jellemző az egy-az-egyhez típusú kapcsolat előfordulása.

Néhány rekord tárolására egy táblázatkezelő szoftver, de akár egy négyzetrácsos füzet is megfelelne. Az adatbázis-kezelés azonban minőségileg is más kategória. Az adattárolás alapvető elemein (új rekord felvétele, módosítása, törlése) kívül ugyanis még számos egyéb funkciót

használatára nyílik lehetőségünk a korszerű adatbázis-kezelők segítségével. Felhasználóként komolyabb igényein is lehetnek ugyanis. Szükségünk lehet például egy (formázott) listára az adatainkról. Másrészt általában előnyben részesítjük azokat a listákat, melyek elemeinek sorrendje valamiféle szabályszerűséget követ. Vagyis rendezések kellhetnek, esetleg több szempont alapján is. A leggyakoribb azonban a különféle szűrések, az úgy nevezett **lekérdezés**ek igénye. Bár ezek egyszerűbb kivitelezésére a táblázatkezelő szoftvereket is felkészítették, hosszú távon, igényeink növekedésével nem kerülhetjük el valamilyen adatbázis szoftver használatát. Konkrét példánk esetén például lekérdezéssel kaphatjuk meg az összes szakkönyvünk listáját.

Az adatbázis-kezelés elterjedésének kezdetén még az volt az általános helyzet, hogy a komplett adatbázis-kezelést ugyanazon személyek programozták le. Ma már nem ez a gyakorlat. A konkrét megoldások során kész adatbázis-kezelő magprogramokat, úgynevezett **adatbázis motor**okat használnak. Az adatbázis motor feladata, hogy kapcsolatot teremtsen a fizikai adatbázis és az azt kezelő alkalmazások között, végrehajtva az írási, olvasási, valamint a keresési műveleteket.

Képesnek kell lennie az adatbázishoz érkező alapvető feladatok ellátására, például táblák, mezők, létrehozására, esetlegesen a tevékenységek naplózásra, a hibák észlelésére is. Az alkalmazáskészítő programozó az adatbázis motor funkcióit felhasználva írja meg a kívánt, a felhasználó számára testre szabott szoftvert. (Megjegyezzük, hogy a terminológia ez esetben nem egységes. Néha az „adatbázis motor” elnevezést az az egész „adatbázis-kezelő”, más esetben viszont kizárólag az adatbázis és a felhasználói réteg közti interfész elnevezésére használják.)

Az adatbázis motorok lehetnek drága, rendkívüli teherbírású, gazdagon szofisztikát kereskedelmi szoftverek (ilyen például az Oracle), egyszerűbbek, kevésbé finoman beállíthatóak (mint például a szintén kereskedelmi MSSQL, vagy az ingyenes és szabad MySQL és PostgreSQL), vagy az előzőekhez képest nagyon egyszerűek is. Ez utóbbiak közé tartozik az SQLite.

1. ábra: Az SQLite logója

Az adatbázisokat leggyakrabban többen használják, akár egy időben is. Jellemző felhasználási mód, amikor a központi adatbázist a számítógépes hálózaton keresztül számos kliens éri el. Az adatbázis motor ilyenkor külön számítógépen fut, de ez nem szükségszerű, egyszerűbb esetekben a kiszolgáló (szerver) és a felhasználó (kliens) program akár ugyanazon a gépen is futhat. Az SQLite eleve ilyen, egygépes rendszerekre készült. Sőt az SQLite egy úgynevezett beágyazott (embedded) relációs adatbázis kezelő, azaz az őt használó szoftverhez linkelve lehet használni, tehát nem egy különálló adatbázis szerver. Ez azért lehet így, mert az SQLite önálló, kisméretű (mérete körülbelül fél MB), C forrású programkönyvtárként (úgynevezett library-ként) megvalósított adatbázis-kezelő rendszer, illetve adatbázismotor.

Az adatbázisok kétféle szinten valósulnak meg; logikai és fizikai szinten. A fizikai szint az adatok tárolásának fizikai megvalósulása, tehát hogy a bitek milyen szerkezetekben helyezkednek el a háttértároló egységeken. A SQLite esetében az adatok fizikai tárolása, a komplett adatbázis elhelyezése egyetlen fájlban valósul meg. Ez nem általános az adatbázis-kezelésben, sőt kifejezetten ritka. Ez a tulajdonság – az adatbázis-kezelő mérete és beágyazhatósága mellett – egyben az SQLite könnyű kezelhetőségének kulcsa. Az adatbázisok logikai szintje az általunk megtervezett struktúrára utal, szerkezeti és működési leírásuk alakját szokás adatmodellnek nevezni. Az adatbázisok logikai szintje, más néven a logikai adatbázis nem foglalkozik azzal, hogy az adatok valójában hogyan tárolódnak a háttértáron, hogyan írjuk, olvassuk azokat, illetve hogyan töltjük be őket a memóriába.

Az adatmodellek eltérései alapján beszélhetünk relációs, objektumorientált, hálós, deduktív, illetve objektumrelációs, deduktív

relációs, deduktív objektumorientált adatbázisokról. Egyes adatbázisokban előfordulhat, hogy ugyanazt a fizikai adatbázist több, különböző logikai adatbázison keresztül érhetjük el, tehát ugyanazokat az adatokat akár relációs, vagy objektumorientált technológiával is kezelhetjük. Mivel az SQLite kizárólag a(z egyébként legelterjedtebb) relációs technológiára van felkészítve, jegyzetünkben másfajta adatbázis-kezelési módszerrel nem is foglalkozunk. Az eddig említett példa-adatbázisunkat is a relációs adatbázismodell bevezetésével összhangban terveztük. (Az adatbázis típusoknak más csoportosítása is létezik. Lásd például http://en.wikipedia.org/wiki/Data_model.)

4. A relációs adatbázismodell

Relációs adatbázisnak a relációs adatmodell elve alapján létrehozott adatok összességét nevezzük. Mint ahogy minden más adatmodell, ez is definiálja az általa használt jellemző adatszerkezeteket, valamint a rajta értelmezett műveleteket.

A **relációs adatmodell** egy olyan adatmodell, amelynek legfontosabb eleme a relációnak, mint matematikai objektumnak a fogalma. A relációs adatmodell a logikai adatbázis kereteit határozza meg.

A reláció matematikai értelemben két vagy több halmaz Descartes szorzatának részhalmaza. Nyilván akkor van valódi értelme a fogalomnak, ha az alaphalmazok és a megadott részhalmaz egyike sem üres. Lényegileg arról van szó, hogy az elemek közti reláció (vagyis más néven kapcsolat) írja le az adatszerkezetet. Az 1. táblázatban szereplő adatok esetében a „cim” halmazban szereplő „Matematika” elem a „szerzok” halmazban lévő „Dr. Gerőcs László” és „Obádovics József Gyula” elemmel is relációban van. Az összes kapcsolat megadása megmutatja az adatbázis felépítését.

A relációs adatbázis-kezelők esetén az adatbázis tervezési fázisában a mezőknek kötelezően meg kell adnunk egy **adattípus**t. A leggyakoribb adattípusok a következőek:

- NUMBER. Szám típus, lehet egész vagy valós.
- BOOLEAN. Logikai típus. Értéke 0 vagy 1 (hamis vagy igaz, esetleg igen vagy nem).
- CHAR. Rögzített hosszúságú karakter típus.
- VARCHAR. Változó hosszúságú karakter típus.
- DATE. Dátum típus.
- TIME. Idő típus.

- CLOB. Nagyméretű karakter (long varchar – character large object) típus.
- BLOB. Nagyméretű bináris (long binary – binary large object) típus.

Az adatbázis-kezelők konkrét megvalósításainál eltéréseket tapasztalhatunk a típusokkal kapcsolatban, illetve a fenti adattípusok további altípusokat tartalmazhatnak (pl. egész számoknál 8, 16, 32 és 64 bites tárolás). Megemlítjük, hogy ezek az eltérések kisebb-nagyobb problémákat okozhatnak az adatbázis esetleges egyik rendszerről másik rendszerre való konvertálása során.

Példánkban a „szerzok”, a „cim” és az „alcim” nevű mezők típusát is valamilyen szöveges (karakter) típusúnak, míg az „id”-t egész típusúnak célszerű beállítani.

Előrebocsátjuk, hogy az SQLite sem pontosan a fent felsorolt típusokat kezeli.

Minden mezőnek megadható egy alap (default) érték, például egy szám, vagy valamilyen szöveg, mely a kitöltetlenségre utal. Vagyis amennyiben a táblázat egy sorában nem töltenénk ki a megfelelő oszlopot, úgy az adatbázis-kezelő ezt az alapértéket illeszti be a rekord tárolt adatai közé.

A táblákat és az egyes mezőket **megjegyzés**ekkel is elláthatjuk. Ez azért hasznos, mert ha ésszerűen járunk el, nem lesz szükség külön dokumentáció vezetésére.

5. Az adatbázis-kezeléssel kapcsolatos egyéb fogalmak

Gyakran van szükség arra, hogy az adatbázisunkból (lekérdezéssel) kinyert adatokat valamilyen (nagyság szerinti, vagy ABC) sorrendben kapjuk meg. E célból használjuk az **index**eket. Az index egy táblához kapcsolódó táblázat, mely azt tartalmazza, hogy az eredeti tábla rekordjai egy vagy több mező alapján sorba rendezve hogyan következnek egymás után. Segítségével lehetőségünk nyílik a táblánkban való gyors keresésre. Index használata esetén nem duplázzuk meg más sorrendben az egész táblánkat, hanem csak mutatókat, úgynevezett „indexeket” tárolunk. Előnye, hogy jelentősen növeli az adatok kinyerésének sebességét, viszont hátránya lehet az adatbázis megnövekedett mérete, illetve sok index használata esetén az adatfelvitel lassulása. Ha az adatbázison végzett műveletek többsége adatfelvitel és módosítás, célszerű az indexek számát alacsonyan tartani (ez maximum 4-5 indexet jelent). Minél több indexet használunk, az adatbázismotor annál több időt fordít az indexek frissítésére.

Ha viszont a lekérdezések vannak túlsúlyban, akkor célszerű minél több indexet használni. Ekkor legfeljebb a rendelkezésre álló tárterületet miatt kell az indexek számát mérlegelnünk, hiszen növelik a helyhasználatot.

Ha arra van szükségünk, hogy egy indexelt mező minden adata különböző, egyedi legyen, **egyedi index**et használunk.

Többtáblás adatbázis-kezelés esetén felmerül a **hivatkozási integritás** kérdése. A hivatkozási integritás olyan szabályrendszer, amely a kapcsolatban lévő táblákban található rekordok közötti relációk érvényességére felügyel. Például fontos, hogy az egyik tábla idegen kulcsába nem lehessen olyan értéket beírni, amely a másik (kapcsolódó) tábla elsődleges kulcsában nem található meg. Vagy ne lehessen egy táblából olyan rekordot törölni, amelyhez egy kapcsolódó táblában rekordok tartoznak (pl. a „tipusok” táblából ne lehessen kitörölni olyan rekordot, amire a „konyvek” tábla valamely rekordja hivatkozik)..

Az ilyen típusú szabályokat **kényszer**eknek (constraints) nevezzük. A legtöbb adatbázis-kezelő többé-kevésbé automatikusan tudja lekezelni az ilyen helyzeteket. Az SQLite ellenben nem tudja önállóan kezelni a hivatkozási integritást, nincs például idegen kulcs kényszer. Ennek a problémának a kezeléséről a programozónak magának kell gondoskodnia.

Ide kapcsolódik, hogy az SQLite nem ismeri a JOIN funkciót, tehát a táblák összekapcsolását a célból, hogy egyszerre több logikai kapcsolatban lévő táblából tudjunk kinyerni adatokat. (Valójában részlegesen ismeri. Konkrétan a LEFT OUTER JOIN-t használhatjuk, de a RIGHT OUTER JOIN-t vagy a FULL OUTER JOIN-t nem. Ez eltérés a az SQL-92 standardtól.)

Az adatbázis-kezelés fogalmi köréhez tartozik még a **trigger** (jelentése „ravasz” (a fegyver elsütőbillentyűje), bár a magyar elnevezés nem igazán honosodott meg) is. A trigger egy adatbázis-eseményre adott válasz. Három részből áll: eseményből, feltételből és egy utasításból. Ha az esemény bekövetkezik, az adatbázis-kezelő megvizsgálja a feltétel teljesülését. Ha teljesül, végrehajtja az utasítást (lefuttat egy előzetesen meghatározott programot). Tehát a trigger nem az adatbázishoz tartozik, hanem az adatbázis-kezelőhöz.

A triggerek az adatok feltöltésekor vagy törlésekor végrehajtódó parancsok előtt, helyett vagy után hívathatók meg. Céljuk sokféle lehet, például az üres mezők kitöltése, hivatkozási integritás biztosítása, különféle hibák bekövetkezésének megakadályozása.

6.Az SQLite főbb jellemzői

Az SQLite egy kis méretű, beágyazható relációs adatbázis-kezelő rendszer, melyet C forrású programkönyvtárként valósítottak meg, és amely nyelve legjobban az SQL-92 szabvány hasonlít. Részlegesen megvalósítja a triggereket és a legtöbb összetett lekérdezést.

A szoftver első verzióit D. Richard Hipp tervezte és programozta le. Jelenleg egy nemzetközi csapat fejleszti és nyújt rá szakmai támogatást. Az SQLite forráskódja nyílt, közkincs (public domain). Ez azt jelenti, hogy a program forrása nyilvános, elérhető az SQLite honlapjáról – a http://www.sqlite.org címről – (akárcsak a különböző rendszerekre lefordított binárisok), bárki szabadon megismerheti, felhasználhatja korlátozások nélkül bármire, legyen az magán- vagy üzleti cél. A program legfrissebb verziója jelenleg a 3.7.15.2. A 3. fő verziószámú változat megjelenése komoly változásokat hozott, jegyzetünk csak ezzel a sorozattal foglalkozik.

Az SQLite használata során nem különálló folyamatként működik, hanem a gazdaprogram részeként, a hozzálinkelt programkönyvtár révén. Ez nem a megszokott gyakorlat, hiszen legtöbb esetben az adatbázis-kezelők kliens-szerver alapon működnek. A program részeként működve a kommunikáció függvényhívásokon keresztül valósul meg, melyek gyorsabbak, mint a folyamatok közötti kapcsolat. Az SQLite tervezésénél fontos szempont volt, hogy könnyen hozzáilleszthető legyen minél többféle nyelven írt programhoz, és lehetőleg ne függjön más függvénykönyvtáraktól. Így az SQLite-hoz létezik csatlakozó felület BASIC, C, C++, Common Lisp, Java, C#, Visual Basic, .NET, Delphi, Curl, Lua, Tcl, R, PHP, Perl, Ruby, Objective-C, Python, newLisp, Haskell, OCaml, Smalltalk és Scheme nyelvekhez, sőt használata megoldható Javascript és VBScript-ből meghívva is. (Forrás: http://hu.wikipedia.org/wiki/SQLite)

Az SQLite által kezelt egész adatbázis (mindenestül, tehát a definíciók, táblák, indexek és az adatok összessége) egyetlen fájlban tárolódik. Ez a fájl a programot futtató számítógépen található. Gépen természetesen nem

csak személyi számítógépet, hanem okostelefont is érthetünk. Maga a fájl keresztplatform, vagyis a bármilyen – akár 32, akár 64 bites – ismert rendszerre (Windows, Linux, Mac-OS X, különféle BSD-k, Solaris, Symbian, Android) egyszerű fájlmásolással átvitt adatbázis ott korlátozás nélkül használható. Az SQLite a szöveg típusú adatokat belsőleg alapértelmezés szerint UTF-8-ban tárolja, de tudja kezelni az API UTF-16 kódolású szöveget is.

Az SQLite tervezésénél fontos szempont volt az egyszerűség, ezért nem meglepő, hogy nem olyan kifinomultan vezérelhető, mint ahogy azt bonyolultabb szerkezetű és működésű adatbázis-kezelők esetén megszokhattuk. Az SQLite esetében például a zárolási technika úgy valósul meg, hogy adatok írásának kezdetekor a komplett adatbázis állománya zárolás (locking) alá kerül. Ez azt jelenti, hogy amíg egy folyamat módosít egy adatot az adatbázisban, addig az adatbázist sem olvasni, sem írni nem lehet. (Valójában ez nem ennyire egyszerű. A részletekért lásd http://www.sqlite.org/lockingv3.html.) Ilyenkor az írási művelet meghiúsul, hibaüzenetet kapunk. Viszont olvasási műveletek esetén nincs ilyen korlátozás, az adatbázist egyszerre több folyamat is olvashatja párhuzamosan, minden probléma nélkül. Ez egy kellemes tulajdonsága az SQLite-nak: jelenleg nem ismert más olyan nem kliens-szerver architektúrát megvalósító adatbázis-kezelő, mely egy időben több adatbázis-elérést engedélyez.

Az egyszerűség az alapfilozófia fontos eleme. Az SQLite nem igényel semmilyen trükkös beállítási procedúrát, nincs szerverfolyamat, amit el kellene indítani, megállítani, vagy konfigurálni.

A legtöbb relációs adatbázisnak része egyfajta jogosultsági rendszer, melyben nyilvántartják az adathozzáféréshez jogosult felhasználókat, azok jogait és amely a felhasználók minden tevékenységét ellenőrzi. A jogosultsági rendszer különböző összetettségű lehet, nevezetesen lehet hierarchikus vagy egyszintű. Hierarchikus jogosultsági rendszer esetén csoportokat és alcsoportokat alakíthatunk ki, egyszintű jogosultsági rendszer esetén csak szerepekről beszélünk. A csoportoknak és a szerepeknek részletesen szabályozhatók a jogaik: a hozzáférés engedélyezése vagy tiltása az adatbázis objektumaihoz. Már egyetlen objektum elérésénél is nagyon sokféle jogosultság képzelhető el, mint

például az olvasás, a futtatás, a módosítás, a törlés, vagy a szerkezet megváltoztatása.

Az SQLite azonban nem rendelkezik semmilyen belső jogosultsági rendszerrel. Ez nem jelenti azt, hogy nem lehet szoftveresen, az adatbázis program kezelőfelületének programozásával, illetve az operációs rendszer szintjén, megfelelő fájlrendszer-támogatással valamiféle védelmet biztosítani adatainknak, de ne legyenek illúzióink, a gyakorlatban előforduló esetek többségében, minimális hozzáértéssel bárki bármilyen adatot kinyerhet az adatbázis tárolására szolgáló fájlból (amennyiben az illető olvashatja a fájlt). Mivel az SQLite felhasználói adatbázissal sem rendelkezik (maga az adatbázis nem különbözteti meg a felhasználókat), ha mégis szükségünk lenne ilyesmire, azt valamilyen módon le kell programoznunk. Ez azonban nem jellemző, az SQLite tipikus felhasználása nem igényli az autentikációt. Mindenesetre webes adatbázis-kezelés esetén célszerű az adatbázis fájlokat a `public_html` vagy `www` mappán kívülre elhelyezni, vagyis hogy azt weboldalról ne lehessen direktben elérni.

Tehát az SQLite nem igényel adminisztrátort, aki létrehoz egy új adatbázist és beszabályozza a felhasználók jogait, hiszen az adatbázist bárki létrehozhatja, jogokat állítani pedig nem is lehet. Az SQLite még rendszerhiba, összeomlás esetén sem igényel különleges, adminisztrátor-szintű beavatkozást (az adatbázis integritása teljes mértékben a fájlrendszerre van bízva).

Az SQLite típuskezelése nagy mértékben különbözik a többi adatbázis-kezelőnél tapasztaltaktól. Az SQLite ugyanis nem kifejezetten típusérzékeny. Általában nem foglalkozik azzal, hogy milyen típust adtunk meg melyik mezőnél, például egy egész típusú oszlopba nyugodtan írhatunk szöveget is, bár valójában ekkor a SQLite a szöveget egésszé konvertálja, vagyis az adatokat ez esetben gyengén típusos módon kezeli. Tulajdonképpen egy adattípus nem egy tábla mezőihez, hanem az egyedi értékekhez van hozzárendelve, vagyis dinamikus típuskezelést használ. A rendezések során azonban szerepet kap a megadott mezőtípus, ugyanis a megadottnak megfelelően számként, vagy szövegként kezelve lesznek sorba rendezve az elemek.

Az SQLite által használt adattípusok a következőek:

- NULL. Az értéke mindig NULL.
- INTEGER. Egész típus. Az értéke előjeles egész szám 1, 2, 3, 4, 6, vagy 8 bájton tárolva (az értéktől függ).
- REAL. Valós típus. Az értéke valós szám 8 bájton tárolva (lebegőpontos ábrázolással),
- TEXT. Szöveg típus. Az értéke szöveg (string), az adatbázis kódolásával tárolva (UTF-8, UTF-16BE vagy UTF-16LE).
- BLOB. Nagyméretű bináris típus. Az értéke blob (bináris adatok gyűjteménye), pontosan olyan módon tárolva, ahogy az a bemenetre érkezett.

A típuskezelés ezen sajátossága előnyként jelentkezik a dinamikus típuskezelésű script-nyelvekben való alkalmazás esetén, azonban más aspektusból szemlélve komoly hátrányt jelent. Az SQLite-tal kapcsolatban megfogalmazott legnagyobb ellenérv az, hogy képtelen a tipikus adatbázisokban található szigorúan típusos oszlopok kielégítő kezelésére. Ez megnehezíti, sőt bizonyos esetekben akár lehetetlenné is teheti idegen adatbázisok értelmes importálását. Valamilyen (nem minden igényt kielégítő) konvertálási eljárás létezik, leírása megtalálható az SQLite honlapján, tárgyalása azonban meghaladja jegyzetünk kereteit.

További furcsaság még, hogy bár a szövegesnek megjelölt oszlopok esetén megadhatjuk a méretet, ezt az SQLite (más adatbázis-kezelőkkel ellentétben) figyelmen kívül hagyja, leginkább csak magunknak dokumentálhatjuk vele, hogy milyen adatokat is szeretnénk itt tárolni.

7. Az SQLite felhasználási területei

Egyre több népszerű szoftver használja az SQLite-ot. Lássunk erre néhány példát:

- A Mozilla Firefox böngésző a konfigurációs adatokat, könyvjelzőket, sütiket tárolja SQLite adatbázisban.
- MacIntosh számítógépek (minden egyes példányukban több SQLite is fut).
- PHP-t futtató, adatbázist kezelő honlapok egy része (az arány nehezen becsülhető).
- A Skype kommunikációs szoftver.
- Újabb típusú Symbian alapú okostelefonok.
- iPhone és Android telefonok.
- A Solaris 10 (szerver) operációs rendszer, mely már a boot folyamathoz is SQLite-ot használ.
- A McAfee anti-vírus szoftver.

Ezen szoftverek felhasználóinak száma összesen akár a milliárdot is meghaladhatja.

Bár az SQLite nyilvánvalóan nem alkalmas arra, hogy például egy nagyvállalati központi adattár-kiszolgálót építsünk belőle, de saját webhelyeink blogainak, fórumainak, kisebb online boltjaink meghajtására kiválóan alkalmas lehet, nem is beszélve okostelefon-alkalmazásaink helyi adattárolási igényeinek kielégítéséről.

Olyan esetekben érdemes megfontolni az SQLite alkalmazását, amikor hasznos lehet, hogy

- egyszerűen beüzemeltethető és kezelhető,
- külön szolgáltatástól, szervertől független,
- adatainkat a helyi rendszeren tudjuk tárolni,
- kis erőforrásigénnyel rendelkezik, de képes a megfelelő szolgáltatások biztosítására,
- egyszerűen hordozható.

Vegyük észre, hogy egy hordozható eszközhöz írt szoftvernek az esetek többségében pontosan ilyen adatbázis-kezelőre van szüksége.

8. Az SQL

Mi az SQL?

Az 1970-es években már komoly igény mutatkozott egy egységes, relációs adatmodell lekérdező nyelv kidolgozására. Az alapötlet az IBM-től származott, de más gyártók (pl. az Oracle) is érdekeltek voltak kifejlesztésében. Végül is iparági összefogással, 1986 körül jött létre az első, úgynevezett SQL változat, az SQL-86, melyet szabványként az ANSI (Amerikai Nemzeti Szabványügyi Intézet – American National Standards Institute) és az ISO (Nemzetközi Szabványügyi Szervezet – International Organization for Standardization) is bejegyzett.

Az SQL jelsorozat egy betűszó, a Structured Query Language (magyarul: strukturált lekérdezőnyelv) kifejezés rövidítése. Az SQL a relációs adatbázis-kezelők lekérdezési nyelve, tehát egy kifejezetten szakterület-specifikus programozási nyelv. Bár eredetileg egy nyelvről volt szó, az idők során számos változata, „nyelvjárása” alakult ki, melyek között nagy eltérések lehetnek.

Az SQL fejlesztése természetesen nem állt meg, Számos kiadás látott napvilágot, ezek közül a fontosabbak az alábbiak:

- SQL-86
- SQL-89
- SQL-92
- SQL-99 (más néven: SQL3)
- SQL:2006
- SQL:2008

Az SQLite (néhány kis eltéréssel) az SQL-92 szabványt valósítja meg.

Az SQL nyelv fontosabb tulajdonságai

- Jellegét tekintve részben procedurális (más néven imperatív, vagyis használata értékadó utasítások megfelelő sorrendben történő kiadásával történik), de inkább deklaratív (programozó leírja a problémát, amit a rendszer valósít meg). Nem algoritmikus (hiszen nem tartalmaz vezérlő utasításokat).
- Halmaz-orientált, vagyis a mintához illeszthető összes bejegyzést visszakeresi és kezeli.
- Nem azt kell megadni, hogy hogyan kapjuk meg a kívánt adatokat, hanem azt, mely adatokat szeretnénk megkapni.
- Az SQL parancsok végrehajtásakor egy belső algoritmus dönti el, hogy milyen konkrét elemi lépésekkel születik meg a várt eredmény.
- Az utasítások gyakorlatilag egyszerűen értelmezhető angol mondatok, melyeket pontosvessző választ el egymástól.
- Nevétől (Query Language) eltérően nem csak lekérdezések végrehajtására alkalmas, hanem használható adatok karbantartására, az adatok rendelkezésre állásának ellenőrzésére és számos egyéb adatbázis-kezelő funkció végrehajtására is.
- Az adatbázis-kezelő rendszer integrált része, nem pedig önálló szoftver.
- Felhasználható interaktívan (SQL konzolból), vagy fejlesztőeszközökbe beépítve.
- Elágazást, ciklust, rekurziót az SQL nyelv eredetileg nem tartalmazott. Az újabb SQL változatokban ezek is megjelentek, bár nagy SQL nyelvjárásbeli különbségek jöttek létre konkrét megvalósulásaik során. Az SQLite eredendően ilyen utasításokat nem tartalmaz, de ha nagy szükségünk lenne rá, trükkökkel, kerülő utakon (rekurzív triggerekkel vagy extra táblák felvételével) célt érhetünk. Mindez azonban nem tekinthető tipikus felhasználásnak, meg is haladja jegyzetünk kereteit.

Összességében elmondható, hogy az SQL egy, a relációs adatbázisban tárolt adatok kezelésére alkalmas, hatékony eszköz.

Az SQL nyelv elemei – az SQLite tükrében

Néhány szót a szintaktikáról. Az SQL nyelvhez tartozó utasításneveket és egyéb kulcsszavakat nyomtatott nagybetűvel, a tábla-, mező-, index, stb. neveket kisbetűkkel (esetleg nagybetűvel kezdve) írjuk, bár ez inkább csak a parancsok olvasását megkönnyítő konvenció, mert az SQLite nem különbözteti meg a kis- és nagybetűket. Azért sem érdemes eltérni ettől, mert egyes esetekben az adatbázis-kezelő viszont megkülönbözteti a kis- és nagybetűket (néha ez az adatbázis-szerver telepítésétől is függhet), és ebből (az adatbázis hordozásakor) kellemetlen bonyodalmak támadhatnak. Hasonló okok miatt kerüljük az általunk definiált objektumok nevében az ékezetes és egyéb, nem szokványos karakterek használatát, még ha a rendszer meg is engedné. SQLite esetén ez a használt kezelőfelülettől függ. Az utasítások bemutatásánál a jobb láthatóság kedvéért sortöréseket is alkalmazunk, ez sem szükséges okvetlenül.

Az SQLite nyelv által használt SQL kulcsszavak az 2. táblázatban találhatóak. Haladó SQL felhasználóknak is nagyon fontos tanulmányozni az SQLite „nyelvjárását”, mert számos standard SQL kulcsszó hatástalan, vagy bizonyos esetekben nem várt eredményű lehet.

Az alábbiakban kategóriákba rendezve vesszük végig a főbb, az SQLite használata esetén érvényes SQL utasításokat. Az utasítások teljes körű tárgyalása nem volt cél, csupán az alapvető használat során előforduló eseteket ismertetjük. További információkért lásd az SQLite dokumentációs listáját az http://www.sqlite.org/docs.html címen.

Az SQL nyelv elemeit hat fő csoportba sorolhatjuk. Ezek az adatdefiníciós (más néven adatleíró) (Data Definition Language, DDL), adatkezelési (más néven adatkarbantartó) (Data Manipulation Language, DML), lekérdező (QUERY Language)) és adatvezérlő (Data Control

Language, DCL), tranzakció-kezelő (Transaction Control (TCL)) és egyéb nyelvi elemek.

ABORT	DATABASE	INDEXED	RAISE
ACTION	DEFAULT	INITIALLY	REFERENCES
ADD	DEFERRABLE	INNER	REGEXP
AFTER	DEFERRED	INSERT	REINDEX
ALL	DELETE	INSTEAD	RELEASE
ALTER	DESC	INTERSECT	RENAME
ANALYZE	DETACH	INTO	REPLACE
AND	DISTINCT	IS	RESTRICT
AS	DROP	ISNULL	RIGHT
ASC	EACH	JOIN	ROLLBACK
ATTACH	ELSE	KEY	ROW
AUTOINCREMENT	END	LEFT	SAVEPOINT
BEFORE	ESCAPE	LIKE	SELECT
BEGIN	EXCEPT	LIMIT	SET
BETWEEN	EXCLUSIVE	MATCH	TABLE
BY	EXISTS	NATURAL	TEMP
CASCADE	EXPLAIN	NO	TEMPORARY
CASE	FAIL	NOT	THEN
CAST	FOR	NOTNULL	TO
CHECK	FOREIGN	NULL	TRANSACTION
COLLATE	FROM	OF	TRIGGER
COLUMN	FULL	OFFSET	UNION
COMMIT	GLOB	ON	UNIQUE
CONFLICT	GROUP	OR	UPDATE
CONSTRAINT	HAVING	ORDER	USING
CREATE	IF	OUTER	VACUUM
CROSS	IGNORE	PLAN	VALUES
CURRENT_DATE	IMMEDIATE	PRAGMA	VIEW
CURRENT_TIME	IN	PRIMARY	VIRTUAL
CURRENT_TIMESTAMP	INDEX	QUERY	WHEN
			WHERE

2. táblázat: Az SQLite által használt SQL kulcsszavak

Adatdefiníciós nyelv

Az adatdefiníciós nyelv az adatbázis szerkezetének, sémájának meghatározására szolgál. Egy adatbázishoz nyilvánvalóan csak egyetlen séma tartozhat, amely pontos leírást ad az adatszerkezetről, a tárolási struktúráról, és a konkrét adatelemek között fennálló logikai kapcsolatokról. Itt találjuk az adatbázisok és a táblák létrehozási, módosítási és törlési parancsait.

Utasítás neve:	**CREATE TABLE**
Leírás:	Létrehoz egy táblát.
Használata:	CREATE TABLE tabla_nev (mezo_nev1 adat_tipus,...,mezo_nevn adat_tipus);
Példa:	CREATE TABLE konyvek (id INTEGER PRIMARY KEY AUTOINCREMENT, szerzok TEXT, cim TEXT NOT NULL, alcim TEXT);
Magyarázat:	Létrehozza a „konyvek” nevű táblát, benne az „id” nevű egész, illetve a „szerzok”, „cim”, „alcim” nevű szöveg típusú mezőkkel. Az „id” mező elsődleges kulcs, értékei automatikusan fognak növekedni, a „cim” mezőnek pedig mindenképpen értéket kell kapnia, ami ha más nem, egy üres string lesz.

Utasítás neve:	**ALTER TABLE**
Leírás:	Módosít egy táblát.
Használata:	ALTER TABLE tabla_nev RENAME TO tabla2_nev; vagy ALTER TABLE tabla_nev ADD COLUMN uj_mezo_nev adattipus;

Példák:	ALTER TABLE konyvek RENAME TO konyv_tabla; vagy ALTER TABLE konyvek ADD COLUMN ido TEXT;
Magyarázat:	Átnevezzük a „konyvek” nevű táblát „konyv_tablara”, másik példánkban ugyanezt kibővítjük az „ido” nevű mezővel.

Utasítás neve:	**DROP TABLE**
Leírás:	Töröl egy táblát.
Használata:	DROP TABLE tabla_nev;
Példa:	DROP TABLE konyvek;
Magyarázat:	Törli a „konyvek” nevű táblát.

Utasítás neve:	**CREATE INDEX**
Leírás:	Létrehoz egy indexet.
Használata:	CREATE INDEX index_nev ON tabla_nev (mezo_nev1 ACS/DESC,...,mezo_nevn ACS/DESC);
Példa:	CREATE INDEX cimlista ON konyvek (cim ASC, alcim DESC);
Magyarázat:	Létrehozza a „cimlista” nevű indexet a „konyvek” nevű tábla „cim” mezője szerint, azon belül pedig az „alcim” nevű mezőre, „cim” szerint növekvő, „alcim” szerint csökkenő sorrendben.

Utasítás neve:	**DROP INDEX**
Leírás:	Töröl egy indexet.

Használata:	DROP INDEX index_nev
Példa:	DROP INDEX cimlista
Magyarázat:	Törli a „cimlista” nevű indexet.

Utasítás neve:	**CREATE TRIGGER**
Leírás:	Létrehoz egy triggert.

Használata:	CREATE TRIGGER trigger_nev
	BEFORE/AFTER/INSTEAD OF
	DELETE/INSERT/UPDATE
	ON tabla_nev
	BEGIN
	utasitas1;...utasitasn;
	END;

Példa:	CREATE TRIGGER konyv_beszuras_utani_ido
	AFTER
	INSERT
	ON konyvek
	BEGIN
	UPDATE konyvek SET ido=CURRENT_TIMESTAMP
	WHERE rowid = new.rowid;
	END;

Magyarázat:	Létrehozza a „konyv_beszuras_utani_ido” nevű triggert, amelynek parancs része lefut, miután a táblába beszúrtunk egy adatot. Ilyenkor az aktuálisan szerkesztett rekord „ido” mezőjébe beírja az aktuális rendszeridőt. Lásd még: UPDATE.

Utasítás neve:	**DROP TRIGGER**
Leírás:	Töröl egy triggert.
Használata:	DROP TRIGGER trigger_nev;
Példa:	DROP TRIGGER konyv_beszuras_utani_ido;
Magyarázat:	Törli a „konyv_beszuras_utani_ido” nevű triggert.

Utasítás neve:	**CREATE VIEW**
Leírás:	Létrehoz egy nézetet.
Használata:	CREATE VIEW nezet_nev AS SELECT kif;
Példa:	CREATE VIEW cimek AS SELECT cim FROM konyvek;
Magyarázat:	Létrehozza a „cimek” nevű nézetet, amely a SELECT parancs eredményét tartalmazza, konkrét példánkban a „konyvek” tábla „cim” mezőjének értékeit. Lásd még: SELECT.

Utasítás neve:	**DROP VIEW**
Leírás:	Töröl egy nézetet.
Használata:	DROP VIEW nezet_nev;
Példa:	DROP VIEW cimek;
Magyarázat:	Törli a „cimek” nevű nézetet.

Megjegyzések:

1. Ha egy már létező táblát, indexet, triggert vagy nézetet akarunk létrehozni, vagy nem létező táblát, indexet, triggert vagy nézetet akarunk törölni, módosítani, hibaüzenetet kapunk. Például

   „Error: table tabla_nev already exists”.

   Ha el akarjuk ezt kerülni, használjuk az IF NOT EXISTS (ha nem létezik), vagy az IF EXISTS (ha létezik) kulcsszavakat.

   Példák:

   - CREATE TABLE IF NOT EXISTS kis_konyvek(
     id INTEGER,
     szerzok TEXT,
     cim TEXT);

- DROP INDEX IF EXISTS cimlista;

2. Ha egyedi értékű mezőt vagy egyedi indexet szeretnénk létrehozni, használjuk a UNIQUE kulcsszót.

   Példák:

   - CREATE TABLE kis_konyvek(
     id INTEGER,
     szerzok TEXT UNIQUE,
     cim TEXT);

   - CREATE UNIQUE INDEX cimlista
     ON konyvek (
     cim ASC,
     szerzok DESC);

3. Ideiglenes tábla létrehozására használjuk a CREATE TEMPORARY TABLE, vagy CREATE TEMP TABLE parancsot (lásd még: CREATE TABLE). Az ilyenkor létrejövő tábla csak az aktuális adatbázis-kapcsolatban látszik, ha kilépünk az adatbázisból, automatikusan törlődik.
4. Fontos tudnunk, hogy az SQLite nem tartalmazza a CREATE DATEBASE utasítást. Az adatbázis létrehozásáról a 10. fejezetben olvashatunk.
5. Az ALTER TABLE a tábla nevének átnevezésén és új mező beszúrásán kívül mást nem tud. Tehát az SQLite nem képes mező átnevezésére vagy törlésére.
6. Szintén SQLite sajátosság, hogy egy AUTOINCREMENT mező létrehozásához valójában elegendő csupán elsődleges kulcsnak és INTEGER típusúra állítani a kívánt mezőt, ezek után a mező értéke automatikusan növekszik minden újonnan felvett rekordban.

Adatkezelési nyelv

Ha elkészültünk az adatbázis sémájának kialakításával, jöhet az adatkezelés. Feltölthetünk, megváltoztathatunk, törölhetünk rekordokat, vagy azok egyes részeit.

Utasítás neve:	**INSERT**
Leírás:	Beszúr egy új rekordot a táblába.
Használata:	INSERT INTO tabla_nev (mezo_nev1,...,mezo_nevn) VALUES (kif1,...,kifn);
Példa:	INSERT INTO konyvek (szerzok,cim,tipus) VALUES (”Szép Jenő”,”Analízis”,1);
Magyarázat:	Beszúr egy új rekordot a „konyvek” táblába. Megadtuk, hogy az értékek mely mezőkhöz tartoznak, majd ugyanilyen sorrendben felsoroltuk az értékeket. A szöveges adatokat idézőjelbe kell tenni.

Megjegyzések:

1. Ha valamelyik mezőnk AUTOINCREMENT típusú, vagy kap alapértelmezett értéket, esetleg nem szükséges megadni, nem kell a rekord hozzá tartozó mezőnevét és értékét felsorolnunk a zárójelek között.
2. Lehetőségünk van csupán az alapértelmezett értékek beszúrására:
   - INSERT INTO tabla_nev DEFAULT VALUES;
3. A beszúrandó érték(ek)nek egy lekérdezés eredményét is megadhatjuk. Például:
   - INSERT INTO tabla_nev (mezo_nev1,...,mezo_nevn)
     SELECT kif;

Lásd még: SELECT. Ha kiválasztás eredménye (amely értékei természetesen más táblából is származhatnak) több rekord, akkor az INSERT mindet beilleszti a táblába.

Utasítás neve:	**UPDATE**
Leírás:	Rekord módosítása.
Használata:	UPDATE tabla_nev SET mezo_nev1=”kif1”,...,mezo_nevn=”kifn” WHERE kif;

Példa:	UPDATE konyvek SET szerzok=”Gerőcs László” WHERE szerzok=”Dr. Gerőcs László”;
Magyarázat:	”Gerőcs László”-ra módosítja a „konyvek” tábla „szerzok” mezőjének értéket minden olyan rekord esetén, ahol a „szerzok” mező értéke ”Dr. Gerőcs László”.

Megjegyzés:

1. Ha a módosítást minden rekordra el akarjuk végezni, hagyjuk el a WHERE kulcsszót és ne írjunk utána semmilyen kifejezést. Például (a „konyvek” táblában minden alcímet „-” jelre cserél):
    - UPDATE konyvek SET alcim=”-”

Utasítás neve:	**DELETE**
Leírás:	Rekord törlése.
Használata:	DELETE FROM tabla_nev WHERE kif;
Példa:	DELETE FROM konyvek WHERE szerzok="Gerőcs László";
Magyarázat:	Törli a „konyvek” tábla minden olyan rekordját, ahol a „szerzok” mező értéke „Gerőcs László”.

Megjegyzés:

1. Ha a tábla minden rekordját törölni akarjuk, hagyjuk el a WHERE kulcsszót és ne írjunk utána semmilyen kifejezést. Például:
    - DELETE FROM konyvek;

Adatlekérdező nyelv

Ez a kategória egyetlen utasítást tartalmaz, azonban talán ez a legfontosabb, de mindenképpen a leggyakrabban használt. Magának az SQL-nek az elnevezésében is benne van a lekérdezés (Q=Query) szó, és

akik keveset hallottak az SQL-ről, azok is ismerni szokták az azt végrehajtó SELECT utasítást. Az igazság az, hogy sokak számára az SQL egyet jelent a SELECT használatával.

Utasítás neve:	**SELECT**
Leírás:	Rekord(ok) adatainak lekérdezése (kiválasztása).
Legalapvetőbb használata:	SELECT mezo_nev1,...,mezo_nevn FROM tabla_nev WHERE kif;
Példa:	SELECT cim FROM konyvek WHERE szerzok="Gerőcs László"; (Lásd még a lenti példákat is!)
Magyarázat:	Megadja a „konyvek” tábla minden olyan rekordjának címét, ahol a „szerzok” mező értéke „Gerőcs László”.

Végrehajtása során az adatbázishoz tartozó adathalmazból, vagy annak egy részéből előállítunk egy másik adathalmazt, konkrétan egy táblát. Tehetjük ezt szűréssel, csoportosítással, rendezéssel, de akár matematikai műveleteket is végezhetünk adatainkon. Ennek érdekében a SELECT utasítás több alparancsot is tartalmazhat. Használata során megadjuk a céltábla elkészítésének szabályát, szabályrendszerét. Az eredmény elérésére használt algoritmust az adatbázis-kezelő hozza létre.

A SELECT használata tehát igen változatos módon történhet. Tanulmányozzuk át alaposan az alábbi példákat.

1. A DISTINCT használatával kiköszöbölhetjük a keletkezett listában lévő duplikációkat. A következő parancs csak a különböző címeket írja ki a „konyvek” táblából.
    - SELECT DISTINCT cim FROM konyvek;

2. A „konyvek” tábla teljes tartalmának felsorolása a szerzők nevei szerinti ABC sorrendben (fordított ABC sorrendben: DESC):
    - SELECT * FROM konyvek
      ORDER BY szerzok ASC;

3. Megadja a „konyvek” tábla minden olyan rekordjának címét és alcímét, ahol a „szerzok” mező értéke „Gerőcs László” és ahol az alcím mező értéke nem üres:
    - SELECT cim, alcim FROM konyvek
      WHERE szerzok="Gerőcs László" AND alcim<>"";

4. Kiválasztja a „konyvek” tábla minden olyan rekordjának címét, ahol a „szerzok” mező értéke „Dr”-rel kezdődik, vagy az alcímben nem szerepel benne az „atem” karaktersorozat:
    - SELECT cim FROM konyvek;
      WHERE szerzo LIKE "Dr%" OR alcim NOT LIKE "%atem%";

5. Mindegyik szerzőtől megad egy könyvcímet:
    - SELECT cim FROM konyvek
      GROUP BY szerzok;

6. A HAVING a WHERE-hez hasonlóan szűr, de nem a táblában, hanem a csoportosítás utáni eredmények között. Például nem mindegyik szerzőtől ad meg egy könyvcímet, hanem csak azoktól, akiknek egynél több könyvük (a csoportban egynél több tag) van:
    - SELECT cim FROM konyvek
      GROUP BY szerzok HAVING COUNT(*)>1;

7. Az egyes lekérdező utasítások akár többszörösen is egymásba ágyazhatóak (subquery). Az alábbi példában a belső SELECT a „konyvek” tábla „cim” mezőjéből létrehoz egy új táblát. A mező ottani neve „cim2” lesz. A külső SELECT kinyeri ennek a tartalmát:
    - SELECT cim2 FROM (SELECT cim AS cim2 FROM konyvek);

   Ez csak egyszerű példa, nem is túl hasznos, hiszen nyilván pontosan ugyanazt eredményezi, mint az alábbi kód:
    - SELECT cim FROM konyvek;

   A következő példa jóval hasznosabb: kiválasztjuk azokat a típusokat, melyekhez nem tartozik egyetlen egy könyv sem.
    - SELECT * FROM tipusok WHERE tipus NOT IN (SELECT tipus FROM konyvek WHERE tipus IS NOT NULL)

8. Végrehajtási sorrendjük a következő: FROM, WHERE, GROUP BY, HAVING, SELECT, ORDER BY.

9. A SELECT-tel számos függvény és műveleti jel használható. Ezek használatáról a gyakorlati részben olvashatunk.
    - Matematikai függvények: ABS, AVG, COUNT, SUM, MIN, MAX, RANDOM
    - Matematikai műveleti jelek: +, –, *, /
    - Szövegkezelő függvények: INSTR, LENGTH, LOWER, REPLACE, UPPER
    - Idő és dátumkezelő függvények: TIME, DATE, DATETIME, STRFTIME
    - Egyéb függvények: SQLITE_VERSION, TOTAL_CHANGES

Adatvezérlő nyelv

Ide tartoznak az SQL nyelv adatvédelmi utasításai, ezekből azonban az SQLite egyet sem tartalmaz.

Tranzakció-kezelő nyelv

Tranzakciónak logikailag összetartozó SQL utasítások együttesét nevezzük. Komoly problémát okozhatna, ha a tranzakció egyes elemeit sikerülne végrehajtani, de nem mindet. Ilyenkor be kell avatkoznunk a folyamatba, hiszen ez az adatbázis **inkonzisztens**sé (ellentmondásossá) válását okozhatja. A beavatkozás a legutóbbi konzisztens állapot visszaállítását jelenti. Tehát vagy végre kell hajtani az egész tranzakciót, vagy egyáltalán nem, egyik részét sem. Szemléletes példa rá a banki tranzakció. Tegyük fel, hogy pénzt utalunk a számlánkról egy másik számlára. Nyilván nem megengedhető, hogy levonják a pénzt a számlánkról, de nem adják hozzá a másik számlához.

Utasítás neve:	**SAVEPOINT**
Leírás:	Az adatállomány aktuális állapotának mentése a célból, hogy probléma esetén visszatérhessünk ebbe az állapotba.
Használata:	SAVEPOINT mentespont_neve;
Példa:	SAVEPOINT mentes;
Magyarázat:	Az adatállomány aktuális állapotának mentése a „mentes” nevű mentéspontba.

Utasítás neve:	**ROLLBACK**
Leírás:	Az adatállomány állapotának visszaállítása mentésből.
Használata:	ROLLBACK mentespont_neve;
Példa:	ROLLBACK mentes;
Magyarázat:	Az adatállomány állapotának visszaállítása a „mentes” nevű mentéspontból.

Utasítás neve:	**BEGIN TRANSACTION és END TRANSACTION**
Leírás:	Tranzakció kezdete és vége.
Használata:	BEGIN TRANSACTION; utasitas1,...,utasitasn END TRANSACTION;
Példa:	BEGIN TRANSACTION; INSERT INTO konyvek (szerzok,cim,tipus) VALUES (”Móricz Ferenc”,”Numerikus analízis I.”,4); INSERT INTO konyvek (szerzok,cim,tipus) VALUES (”Móricz Ferenc”,”Numerikus analízis II.”,5) END TRANSACTION;
Magyarázat:	BEGIN és END TRANSACTION-ök közötti utasítások lesznek a tranzakció részei.

Megjegyzések:

1. A BEGIN TRANSACTION helyett írhatunk egyszerűen BEGIN-t, END TRANSACTION helyett pedig END-et, COMMIT-ot, vagy COMMIT TRANSACTION-t, hatásuk ugyanaz.
2. Tranzakció végződhet ROLLBACK-kel is. Ekkor az állapot visszaáll a tranzakció előttire. Példa:
   - BEGIN
     DELETE FROM konyvek
     ROLLBACK

Utasítás neve:	**ON CONFLICT vagy OR ROLLBACK/ABORT/FAIL/IGNORE/ REPLACE**
Leírás:	Utasítja a rendszert, hogy probléma esetén mit, vagy mit ne tegyen. Az ON CONFLICT nem önálló utasítás, a CREATE TABLE, CREATE INDEX vagy a BEGIN TRANSACTION parancsok kiegészítője lehet. COPY,

	INSERT és UPDATE parancsokkal is használható, ekkor az ON CONFLICT helyett OR használatos.
Használata (példa):	CREATE TABLE teszt (id INTEGER PRIMARY KEY ON CONFLICT IGNORE);
Magyarázat:	Létrehozza a „teszt” nevű táblát az „id” nevű mezővel, mely egész típusú elsődleges kulcs. Nem küld hibaüzenetet, ha két azonos rekordot kísérlünk meg felvinni, de nem is hajtja végre a nem megengedett lépést.

Megjegyzés:

A példában szereplő parancsban az IGNORE helyett a ROLLBACK, ABORT, FAIL, REPLACE kulcsszavak is szerepelhetnek. Ezek használatáról és hatásáról részletesebben például itt: http://www.dbtalks.com/uploadfile/ca5be5/conflict-resolution-algorithms-in-sqlite/ olvashatunk.

Egyéb nyelvi elemek

Az idők során számos olyan utasítás vált az SQL nyelv részévé, melyek a kényelmesebb, hatékonyabb programozói munkát, illetve az erőforrások jobb kihasználhatóságát szolgálják. Az egyes nyelvjárások nagy változatosságot mutatnak ezen utasítások létében és használatában. Mi itt csak az SQLite-ban megvalósított eszközökkel foglalkozunk.

Az SQLite nem rendelkezik külön beállításokat tartalmazó állománnyal, telepítése után azonnal üzemkész. Természetesen sokszor lehet szükség finomításokra, beállítgatásokra. Az ilyen feladatok elvégzésére szolgál az SQLite PRAGMA parancsa. A PRAGMA parancsok a többi SQL kéréssel teljesen azonos módon hívhatók.

Utasítás neve:	**PRAGMA**

Leírás:	A paranccsal egyedi beállításokat érvényesíthetünk, de ezáltal begyűjthetők bizonyos információk is.
Használata (példa):	PRAGMA table_info (tabla_nev);
Magyarázat:	Az adott tábla szerkezetét kapjuk vissza, minden mezőre megkapva az oszlop nevét, típusát, hordozhat-e NULL értéket, valamit (ha meg lett adva) megkapjuk az alapértelmezett értékét.

Más rendszerekből is ismert, igen hasznos karbantartó utasítás a VACUUM.

Utasítás neve:	**VACUUM**
Leírás:	Újjáépíti az adatbázist, optimálisabb belső szerkezetben és méretben.
Használata:	VACUUM;
Magyarázat:	A gyakori beszúrások, törlések, módosítások defragmentálják (töredezetté teszik) az adatbázist. Ezért célszerű a VACUUM parancs rendszeres használata, mely töredezettség-mentesít, valamint törli az üres helyeket, csökkentve ezzel az adatbázis méretét, ami a jobb helykihasználáson kívül az adatbázis-műveletek sebességének gyorsítását is eredményezheti.

Megjegyzés:

1. Mivel az SQLite-ot több, mindennapi szoftverünk is igénybe veszi, érdemes fontolóra vennünk a VACUUM használatát alkalmazói szinten is. Például felgyorsíthatjuk vele a Firefox böngésző futását. Ehhez csak annyit kell tennünk, hogy nyitunk egy terminálablakot, belépünk az SQLite adatbázisokat tartalmazó könyvtárba, és kiadjuk a VACUUM parancsot minden adatbázisban. Mindez automatizálva a következő (az `xxxxxxxx` egy esetenként különböző, 8 karakterekből álló sorozat).

Linuxon:

```
cd ~/.mozilla/firefox/xxxxxxxx.default

for i in *.sqlite; do echo "VACUUM;" |
sqlite3 $i; done
```

(Mivel a `for` ciklusban is megadhatjuk az elérési utat, a `cd` parancs nem feltétlenül szükséges ide.)

Windowson:

```
cd C:\Documents and Settings\
{Felhasználónév}\Application Data\Mozilla\
Firefox\Profiles\xxxxxxxx.default

for %i in (*.sqlite) do @echo VACUUM; |
sqlite3 %i
```

9. Az SQLite telepítése

Általában elmondható, hogy az SQLite egyszerű szerkezete és felépítése miatt a telepítése nem igazán komoly feladat. Az SQLite honlapjáról letölthetjük a forráskódot, valamint előfordított binárisokat számos rendszerre.

Az SQLite telepítése Windows rendszerekre

Az SQLite telepítése általában egyet jelent a megfelelő binárist tartalmazó tömörített fájl (jegyzetünk írása idején ez az `sqlite-shell-win32-x86-3071502.zip`) letöltésével (a http://www.sqlite.org/download.html címről) és kicsomagolásával. (Az intenzív fejlesztés miatt a verziószám gyakran változik.)

Egyes esetekben (például Ruby on Rails használata esetén) szükség lehet a – szintén az SQLite honlapján található – `sqlite-dll-win32-x86-3071502.zip` fájl letöltésére és a benne lévő DLL valamely rendszerkönyvtárba (például a C:\WINDOWS\system könyvtárba) történő másolására.

Az SQLite telepítése Linux rendszerekre

A legtöbb ismert Linux disztribúció esetén a tárolók tartalmazzák az előre fordított SQLite csomagokat, így a telepítéshez használjuk az adott rendszer csomagkezelőjét. Az SQLite jelenleg a 3. sorozatnál tart, így telepítése Ubuntu (és egyéb Debian-típusú) rendszerre a következő:

```
> sudo apt-get install sqlite3
```

Amennyiben valamilyen oknál fogva az – elavult – 2. sorozatot akarjuk használni, akkor az sqlite nevű csomagot telepítsük.

Ahogy megszokhattuk, a tárolók nem mindig a legfrissebb verziókat tartalmazzák. Ha frissebbet szeretnénk, letölthetjük az előre fordított binárist az SQLite honlapjáról:

```
> wget http://www.sqlite.org/sqlite-shell-linux-x86-3071502.zip
```

A tömörített fájlban egyetlen futtatható állományt találunk. (Ez egy 32 bites, dinamikusan linkelt bináris. Ez azt jelenti, hogy 64 bites rendszeren még a 32 bites libek telepítése is szükséges működéséhez.)

Ha forrásból szeretnénk fordítani az aktuális változatot, azt is megtehetjük. Ekkor az előkonfigurált forrást kell letöltenünk:

```
> wget http://www.sqlite.org/sqlite-autoconf-3071502.tar.gz
```

Kicsomagolás után a konfigurálás, fordítás és telepítés lépései a szokásosak:

```
> tar xvfz sqlite-autoconf-3071502.tar.gz
> ./configure
> make
> sudo make install
```

A keletkezett bináris neve sqlite3.

Az SQLite telepítése Mac-OS X (x86) rendszerekre

Mivel az SQLite-ot minden modern Mac-OS X rendszer tartalmazza, semmilyen telepítésre nincs szükség.

Természetesen friss változatot Mac-OS X és Windows esetében is telepíthetünk forrásból (bár ott ennek kevésbé vannak hagyományai).

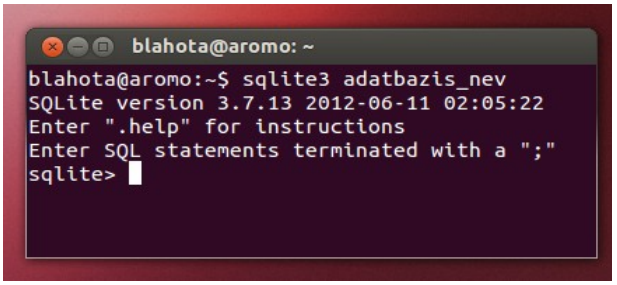
Az SQLite telepítése Androidra, iPadra, iPhonra

Az SQLite az iPad, iPhon és a standard Android része, nem szükséges telepíteni.

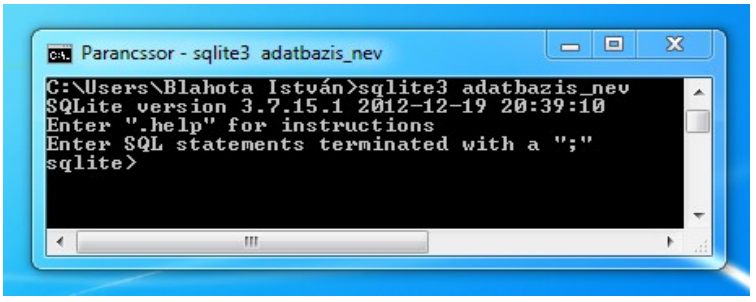
Előfordulhat azonban, hogy néhány Androidos telefonforgalmazó kihagyja azt a rendszerből, esetleg a telepítettnél frissebb SQLite-ot szeretnénk használni. Ilyenkor (rootolt telefon esetén) magunk is telepíthetjük, frissíthetjük. Léteznek erre leírások (melyek tulajdonképpen Linux parancsok végrehajtását írják le lépésről lépésre), de használhatjuk az egyszerűen használható „SQLite Installer for Root” nevű alkalmazást is, mely ingyenesen letölthető a Google Play-ből.

10. Az sqlite3 parancssoros eszköz

Az SQLite adatbázisok kezelése (az adatbázisok létrehozása, illetve SQL utasítások végrehajtása) – az SQLite tervezőinek szándéka szerint – az előzőekben telepített `sqlite3` parancssoros eszköz segítségével történhet, ami egyetlen végrehajtható állományból áll. Mivel ez a program is nyílt forráskódú, kódját tanulmányozva szükség esetén képesek lehetünk egyéb, SQLite állományokat kezelő szoftver létrehozására is.



2. ábra: Az sqlite3 parancssoros eszköz (Ubuntu) Linuxon...



3. ábra: ... és Windows 7-en

Használata egyszerű. Amikor parancssorból kiadjuk az

```
sqlite3 adatbazis_nev
```

utasítást, amennyiben az létezik megnyitja, ha nem létezik létrehozza az adatbazis_nev nevű adatbázist. Ez azért is különösen fontos, mert más SQL-ektől eltérően az SQLite „CREATE” SQL parancsa nem tud adatbázist létrehozni.

A bejelentkezés után látható alapvető információkat a 2. és 3. ábrán láthatjuk. Megjelenik a verziószám a kiadás pontos idejével, egy üzenet arról, hogy a „`.help`” parancs kiadására részletes súgót kapunk, valamint egy tájékoztatást arról, hogy egy kiadott SQL parancsot pontosvesszővel le kell zárni, és csak utána kell Entert ütnünk a végrehajtásához.

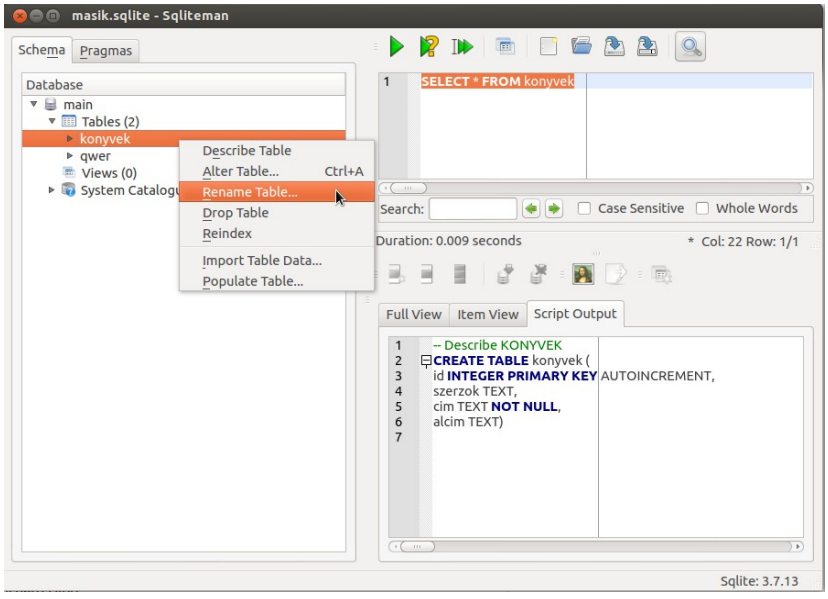
Az sqlite3 belső parancsai (tehát az SQL parancsok nem!) ponttal kezdődnek és az azt követő két karakterrel rövidíthetőek. Így például a „`.he`” parancsra is megkapjuk a súgót, a „`.ex`” is (az „`.exit`” helyett) kiléptet az sqlite3-ból, stb.

11. SQLite adminisztrációs programok

Több adminisztrációt segítő alkalmazás is létezik SQLitehoz. Vannak köztük, melyek többféle adatbázis-kezelővel is használhatóak, és olyanok is, melyet speciálisan SQLitehoz készítettek. Természetesen már az sqlite3 is tartalmaz minden szükséges eszközt SQLite rendszerünk kezeléséhez, de használhatunk könnyen kezelhető grafikus felületű alkalmazásokat is. Az alábbiakban ezek közül mutatunk be néhányat.

sqliteman

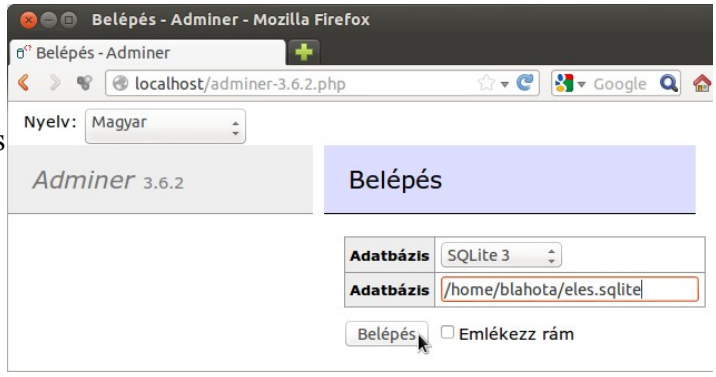
Az sqliteman egy fejlesztői és adminisztrációs célokra készített eszköz, kifejezetten az SQLite számára. Elérhető Windowsra, Linuxra (az ismertebb disztribúciók tárolóiban megtalálható), Mac-OS X-re, és a visszajelzések szerint néhány más, kevésbé ismert operációs rendszerre is. Nyílt forráskódú (GPL v2-es licencű), a forráskód és előre fordított binárisok, valamint dokumentációja elérhetőek honlapjáról: http://sqliteman.com. Ideális eszköz lehet SQL parancsokhoz, táblák, nézetek, triggerek, indexek létrehozásához, módosításához, törléséhez.



4. ábra: Az sqliteman használat közben

Adminer

Az Adminer szintén egy teljes értékű SQLite

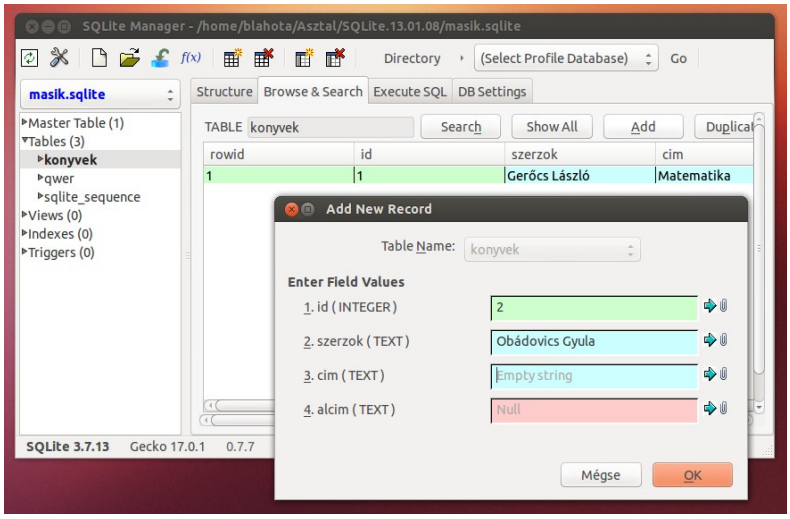


5. ábra: Belépés az Adminerbe

szerkesztő program, de több is annál. Egy igazi nagyágyú, a phpMyAdmin továbbfejlesztése. Míg a phpMyAdmin csak a MySQL kezeléshez készült, az Adminert – számos egyéb kiegészítésen felül – alkalmassá tették az SQLite, **PostgreSQL**, **MS SQL** és az **Oracle** menedzselésére is. Akárcsak a phpMyAdmin, az Adminer is PHP-ban íródott (egyébként egyetlen fájlról van szó mindössze), így futásához webkiszolgálóra van szükség. Támogatja a PHP 4.3.3+-t és a PHP 5-öt is. Elérhető Apache és GPL v2 licenc alatt. A program kezelőfelülete magyar nyelven is használható! Az Adminer, valamint használatát kibővítő további plug-in-ek letölthetőek honlapjukról, a http://www.adminer.org/ címről.

SQLite Manager

Akárcsak az előzőekben bemutatott programok, az SQLite Manager is egy ingyenes, teljes körű SQLite kezelői eszköz, viszont nem csak adminisztrációs célokat szolgálhat. Rekordokat is kezelhetünk ugyanis benne: felvihetjük, módosíthatjuk, törölhetjük azokat, ezért felhasználói alkalmazásra is kitűnően megfelel. Érdekessége, hogy nem önálló program, hanem Firefox kiegészítő. Telepítése igen egyszerű. Kattintsunk a Firefoxban az Eszközök → Kiegészítők (Ctrl+Shift+A) menüpontra, majd keressünk rá az SQLite Manager kifejezésre. A megjelenő alkalmazásnév melletti telepítés gomba kattintva a kiegészítő letöltődik és integrálódik a böngészőbe. A Firefox újraindítása után az Eszközök menüből indíthatjuk.



6. ábra: Adatfelvitel SQLite Managerrel

Minden olyan platformon használható, melyen a Firefox fut. További információkért látogassuk meg honlapját: http://www.sqlitemanager.org/

12. Irodalomjegyzék

- Nyílt forráskódú adatbázis-kezelők (CD melléklettel), Schönhofen Péter, Szak Kiadó, 2007
- SQL adatbázis beágyazása az SQLite segítségével, Michael Owens, Linuxvilág, 2003 augusztus, http://www.linuxvilag.hu/content/files/cikk/31/cikk_31_26_29.pdf
- Using SQLite, Small. Fast. Reliable. Choose Any Three., Jay A. Kreibich, O'Reilly Media kiadó, 2010
- The Definitive Guide to SQLite (2. kiadás), Mike Owens és Grant Allen, Apress kiadó, 2010
- SQLite 3 - Einstieg in die Datenbankwelt, Key Droessler, Lulu.com kiadó, 2010
- The SQL Guide to SQLite, Rick F. van der Lans, Lulu.com kiadó, 2009
- An Introduction to SQLite - 2nd Edition, Naoki Nishizawa Shoeisha kiadó, 2007
- Inside SQLite, Sibsankar Haldar, O'Reilly Media kiadó, 2007
- SQLite, Chris Newman, Sams kiadó, 2004

13. Hasznos linkek

- http://opendir.hu/webalkalmazas/sqlite/171-sqlite-emlekezteto-jegyzet
- http://www.codeproject.com/Articles/119293/Using-SQLite-Database-with-Android
- http://zetcode.com/databases/sqlitephptutorial
- http://www.agt.bme.hu/szakm/adatb/ora.htm
- http://www.sqlite.org/lang_keywords.html
- http://www.sqlite.org/lang.htmlhttp://www.w3schools.com/sql
- http://www.sqlite.org/omitted.html
- http://lcb.hu/linux/sql_alapok.html
- http://zetcode.com/databases/sqlitetutorial/tool
- http://infolab.stanford.edu/~widom/cs145/sqlite/SQLiteIntro.html
- http://www.thegeekstuff.com/2012/09/sqlite-command-examples
- http://souptonuts.sourceforge.net/readme_sqlite_tutorial.html
- http://www.bennadel.com/blog/1940-My-Safari-Browser-SQLite-Database-Hello-World-Example.htm
- http://stackoverflow.com/questions/7470393/helloworld-database-android-connection-with-sqlite
- http://www.vogella.com/articles/AndroidSQLite/article.html
- http://www.vogella.com/articles/Android/article.html
- http://www.chipkin.com/sqlite-vs-mysql-short-summary
- http://www.kobakbt.hu/jegyzet/SQLpelda/sql_0bev.html
- http://stackoverflow.com/questions/7370761/sqlite-loop-statements
- http://www.newobjects.com/pages/ndl/SQLite2%5CSQL-Pragma.htm

- http://hup.hu/node/75077
- http://en.wikipedia.org/wiki/Database_normalization
- http://support.microsoft.com/kb/283878/hu
- http://en.wikipedia.org/wiki/Data_model
- http://www.sqlite.org/lockingv3.html
- http://www.dbtalks.com/uploadfile/ca5be5/conflict-resolution-algorithms-in-sqlite/