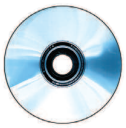
SQL adatbázis beágyazása az SQLite segítségével

Ha élvezni szeretnéd az SQL kényelmét, de nem akarsz egy nagyméretű adatbázis-kiszolgáló telepítésével szenvedni, ágyazd be az SQLite-ot a programodba, bármilyen nyelven íródjon is.



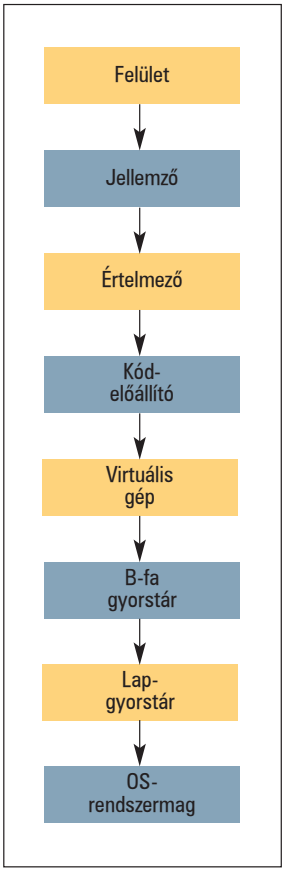
Az SQLite nagy teljesítményű, beágyazott relációs adatbázis-kezelő rendszer, ami egy viszonylag kisméretű C-könyvtárban kapott helyet. Fejlesztője ***D. Richard Hipp***. Támogatása az SQL92 szabvány túlnyomó részére, több tábla és index létrehozatalára, tranzakciókra, nézetekre, ravaszokra és rengeteg különböző ügyfélfelületre és illesztőprogramra terjed ki. A könyvtár önálló egységet képez, és mindössze 25 000 sornyi ANSI C-ben készült kódból áll. Bármely célra szabadon felhasználható; gyors, hatékony és méretezhető, számos különböző rendszeren és géptípuson fut, az ARM/Linux párosítástól kezdve egészen a SPARC/Solarisig. Adatbázis-formátuma bájtszinten alkalmas a különböző bájtsorrendet használó gépek közötti együttműködésre, adatbázisainak mérete pedig akár 2 TB is lehet.

Hipp agyából akkor pattant ki az SQLite ötlete, amikor a General Dynamicsnél dolgozva csapata az amerikai haditengerészet DDG osztályú rombolói számára fejlesztett fedélzeti programot. A program HP-UX alatt futott, és Informix adatbázist használt. Miután elkezdték a munkát, hamarosan felismerték, hogy az Informixszel nagyon nehéz dolgozni. Ha egyszer sikerült üzembe helyezni, akkor elég szépen futott, ám még egy nagy tapasztalattal rendelkező szakembernek is egy teljes napja ráment egy-egy telepítésre vagy frissítésre. Abban az időben a csapat a fejlesztésekhez Linuxot és PostgreSQL-t használt. A PostgreSQL lényegesen kevesebb felügyeletet kívánt, ám ők olyan önálló programot akartak készíteni, ami bárhol futtatható, függetlenül az adott rendszerre telepített egyéb programoktól. 2000 januárjában Hipp és egy munkatársa megvitatták egy egyszerű, felügyeletet és támogatást nem igénylő, beágyazott SQL-adatbázismotor készítésének az ötletét, aminek alapja GDBM lenne. Később Hipp egymaga látott neki a munkának, és hamarosan elkészült az SQLite 1.0.

A General Dynamics a PostgreSQL helyett azonnal az SQLite-ot kezdte alkalmazni. Az SQLite segítségével önálló alkalmazásokat tudtak készíteni, amiket gyorsan és könnyen telepíthettek a bemutatókra és vásárokra vitt hordható (wearable computer) és hordozható számítógépekre. Az Informix továbbra is fontos szerepet játszik a fedélzeti rendszerekben – igaz, Hippet nemrég felkérte a tengerészet, hogy segítsen az SQLite HP 9000-re való lefordításában. Lehet, hogy hamarosan változások lesznek?

A nagyobb módosítások a 2.0-s változattal érkeztek. Az első változat az adatok tárolásához GDBM-et használt, ami rendezetlen kulcsokat, másszóval kivonatolást alkalmaz. Emiatt azonban korlátozott volt az SQLite szolgáltatásainak a köre. Emellett a GDBM GPL szerződéssel jelenik meg, és ez néhányakat visszatart a kipróbálásától. 2001 januárjában Hipp egy saját B-fa alapú háttéren kezdett dolgozni, amivel a GDBM-et akarta kiváltani. Az új B-fa alrendszer kulcsok szerint rendezve tárolja a rekordokat, így hatékonyabb működést, logaritmikus időben végrehajtott minimum- és maximumkeresést, egyenlőtlenségi megszorításokkal végrehajtott indexelt lekérdezéseket tesz lehetővé, illetve a tranzakciókat is támogatja. A végeredmény egy sokkal nagyobb tudású adatbázisrendszer lett. A 2.0-s változat 2001 szeptemberében jelent meg public domain szerződéssel.

Az SQLite felemelkedése is a 2.0-s változat megjelenésével indult meg. Ekkor már egyre több, különféle területeken dolgozó fejlesztő ragadott billentyűzetet, és írta le, hogyan építette be kereskedelmi vagy éppen szabad termékekbe. Néhányan műszaki támogatást vagy egyedi módosításokat is kértek Hipptől. Hipp elmondása szerint az egyik széles körben elterjedt windowsos program újabb kiadásai is magukban foglalják az SQLite egy módosított változatát. Az SQLite nemcsak a fejlesztést eredetileg ösztönző haditengerészetnél, de az amerikai hadsereg egyéb részlegeinél és a Duke Energynél is megtalálta helyét. Másfél évvel ezelőtti nyilvános megjelenése óta az SQLite szolgáltatásainak és felhasználóinak köre rohamosan bővül. Az SQLite Wikibe bepillantva számos további alkalmazást találunk, ezeknek a fejlesztői felfedezték és a saját programjukba építették be az SQLite-ot. Saját Apache modullal is rendelkezik (`mod_auth_sqlite`), aminek létezését már önmagában is a siker jelének tekinthetjük. ***Gerhard Häring*** és jómagam a PySQLite – Python kiterjesztés SQLite-hoz – készítőiként meglepődve számláltunk össze több mint 3000 letöltést, kevesebb mint egy év alatt. Az SQLite jelenleg a legmagasabb pontszámot elért adatbázismotor a ➲ http://freshmeat.net oldalon.



Az SQLite felépítése

Felépítés

Az SQLite tetszetős, moduláris felépítésű. Nyolc elsődleges alrendszerre osztható (lásd az *ábrán*), ezek közül néhány meglehetősen érdekes megközelítést tükröz az adatbázis-kezelés területét illetően.

Az ábra tetején az értelmező és a jelkezelő látható. Az SQLite saját, magas fokon hatékonnyá tett és a Lemon névre keresztelt

1. lista Az explain kimenete egy egyszerű lekérdezésre

```
SQLite version 2.7.1
Enter ".help" for instructions
sqlite> .explain
sqlite> explain select lastname,firstname
        from person;
addr   opcode         p1     p2    p3
----   -------------  ----   ----  ---------
0      ColumnCount    2      0
1      ColumnName     0      0     lastname
2      ColumnName     1      0     firstname
3      Open           0      3     person
4      VerifyCookie   276    0
5      Rewind         0      10
6      Column         0      4
7      Column         0      3
8      Callback       2      0
9      Next           0      6
10     Close          0      0
```

értelmezőt tartalmazza, ami gyors, hatékony kódot készít, és újszerű felépítésénél fogva kifejezetten ellenáll a memóriaszivárgásoknak. Alul a Knuth alapú B-fa megvalósítás található, ami egy hangolható lapgyorsítótár felett fut, a lehető legkisebb számú lemezhozzáférés mellett. A lapgyorsítótár egy operációs rendszertől függő elvonatkoztatási rétegen fut, ennek köszönhetően a könyvtár magasabb fokon hordozható.

A könyvtár középpontjában a virtuális adatbázismotor (VDBE) található. A VDBE végzi az összes adatkezeléssel kapcsolatos feladatot, minden az ügyfelek és a tárrendszer között áramló adat rajta halad keresztül. Tulajdonképpen ez az SQLite szíve. A VDBE szerepe az SQL-utasítások értelmezése után kezdődik. A kódkészítő az értelmezési fa alapján egy apró programot állít elő, ami a VDBE virtuális gép nyelvében található utasítások sorozata. A VDBE egyenként végrehajtja az utasításokat, végeredményként az SQL-utasításban megadott parancsot teljesíti. A VDBE gépi nyelve 128 utasításból áll, ezek mindegyike az adatbázis-kezeléssel kapcsolatos. Külön parancsok szolgálnak a táblák megnyitására, az indexekben való keresésre, a rekordok tárolására és törlésére, illetve a számos egyéb adatbázis-kezelő műveletre. A VDBE minden utasítása egy műveleti kódból és legfeljebb három műveleti értékből áll. Egyes utasítások – jellegüktől függően – akár mindhárom műveleti értéket használják, míg mások egyiket sem. Az `open` utasítás például, ami egy tábla egy mutatóját nyitja meg, mindhárom műveleti értéket használja. Az első műveleti érték (P1) a mutató későbbi kezelésére szolgáló azonosítót tartalmazza. A második (P2) a tábla gyökér (vagyis első) lapjának helyére mutat, a harmadik pedig a tábla neve. A `rollback` utasítás – ellenpéldaként – egyetlen műveleti értéket sem igényel, a VDBE-nek csak annyit kell tudnia, hogy végrehajtsa-e a visszavonást vagy sem.

Ha érdekel, hogy az adott SQL-utasításhoz milyen VDBE-program tartozik, az SQLite parancssorának `explain` parancsával megtekintheted. Az *1. lista* erre mutat példát.

Az `explain` nem csupán akkor hasznos, ha betekintést szeretnénk nyerni a VDBE működésébe, de olyan gyakorlati teendőknél is, mint például a lekérdezések hatékonyságának a javítása. A VDBE valóban érdemes arra, hogy önmagában is kiterjedt vizsgálatok tárgyát képezze. Szerencsére ehhez minden feltétel adott: a motorhoz remek leírás tartozik, működésének elméletéről és programnyelvének műveleteiről részletesen olvashatunk az SQLite weboldalán.

A fizikai tárolás szempontjai miatt minden adatbázis egyetlen fájlban található. Minden olyan objektum tehát, ami szerepel az adott adatbázisban (nézetek, ravaszok, indexek, táblák, sémák stb.), egyetlen, az adott SQLite-adatbázist meghatározó állományban foglal helyet. Az adatbázisfájlok egységesen formázott lapokból állnak össze. A lapméret meghatározása az adatbázis létrehozásakor történik, értéke 512 bájt és 4 Gb között lehet. Alapértelmezett esetben az SQLite 1 Kb méretű lapokat hoz létre, így lehet a legjobb általános teljesítményt elérni.

A tranzakciókat egy második állomány, a napló segítségével valósítja meg, ami ténylegesen csak akkor létezik, ha egy vagy több működő kapcsolat áll fenn az adatbázissal. Minden adatbázishoz egyetlen naplófájl tartozik, ez azokat az eredeti – módosítás előtti – lapokat tartalmazza, amik tartalma a tranzakció feldolgozása közben módosult. Amikor jóváhagyod a tranzakciót, a naplólapokra többé nincs szükség, így törlésre kerülnek. A műveletek visszavonásakor a motor a naplófájlból állítja vissza a lapokat az adatbázisfájlba. A naplófájl használata teszi lehetővé azt, hogy az adatbázis akár egy összeomlást is túlélhessen, és egységes állapotba lehessen visszaállítani. Összeomlás után az első ügyfélnek az adatbázishoz történő csatlakozása egy ravasszal elindítja az előző tranzakció visszavonását. Kicsit pontosabban: amikor az ügyfél csatlakozik, az SQLite megpróbál egy új naplófájlt létrehozni, és ekkor észleli, hogy már létezik egy ilyen. Ha erre kerül sor, akkor a motor feltételezi, hogy összeomlás történt, és a régi naplófájl tartalmát visszamásolja az adatbázisba, gyakorlatilag az eredeti, összeomlás előtti állapotába állítva vissza azt. Az ügyfél csak ezt követően kapja meg a lehetőséget a munka megkezdésére.

Egyszerű API, sok nyelv

Az SQLite egy rendkívül könnyen használható API-val rendelkezik, ami mindössze három függvény révén alkalmas az SQL-utasítások végrehajtására és az adatok kinyerésére. Bővíthető, a programozó C-visszahívók formájában egyedi függvényeket és összesítéseket adhat meg. A C API a parancsfájl-készítő felületek alapja is egyben, ezek egyikét – a Tcl-felületet – a terjesztésben is megtaláljuk. A nyílt forrású programok közössége számos további ügyfélfelületet, csatolót és illesztőprogramot is kifejlesztett, ezek révén az SQLite más nyelvekkel és könyvtárakkal is használható.

A C API használatához mindössze három lépést kell végrehajtani. Az `sqlite_open()` hívással – a fájlnév és az elérési mód megadásával – lehet csatlakozni a kívánt adatbázishoz. Ezután egy visszahívó függvényre van szükség, amit az SQLite az adatbázisból lekért rekordok mindegyikére meghív. A záró lépés az `sqlite_exec()` meghívása, aminek egy karakterláncban a végrehajtandó SQL-utasítást, illetve a visszahívó függvényünkre hivatkozó mutatót adjuk át. A hibakezeléstől eltekintve végeztünk is. A 2. listában (50. CD Magazin/SQL könyvtár) erre találunk egy egyszerű példát.

Szembeötlő – és fontos – eltérés a többi adatbázis-ügyfélkönyvtárhoz képest a visszahívó függvény használata. Az egyéb ügyfél API-k használatakor meg kell várnunk az eredmény összeállítását, az SQLite esetében viszont az eredményt összeállító folyamat kellős közepébe csöppenünk, és végigkövethetjük az eseményeket. Így az adatok kinyerésében is sokkal aktívabb szerepet játszhatunk, és közvetlen hatást gyakorolhatunk a lekérdezési folyamatra. Az adatok összesítését már kinyerés közben elvégezhetjük, de akár meg is szakíthatjuk a rekordok lekérdezését. Vegyük észre, hogy az adatbázis beágyazott

3. lista Python-példa

```
import sqlite

conn = sqlite.connect(db="db", mode=077)
cursor = conn.cursor()

SQL = "select * from person order by
      ➥lastname"
cursor.execute(SQL)

row = cursor.fetchone()
while row != None:
    print "%14s, %15s" % (row[´firstname´],
                         ➥row[´lastname´])

    row = cursor.fetchone()
```

4. lista Perl-példa

```
use DBI;
my $dbh = DBI->connect
          ➥("dbi:SQLite:dbname=dbfile",
                        "","");

my $cursor;
my @rec;

my $SQL = "select * from person order by
           ➥lastname";

$cursor = $dbh->prepare($SQL);
$cursor->execute();

while(@rec = $cursor->fetchrow_array)
{
    print "$rec[0], $rec[1]\n";
}

$cursor->finish;
$dbh->disconnect;
```

jellegénél fogva alkalmazásunk legalább annyira kiszolgáló, mint ügyfél, és a visszahívó felület révén az SQLite maradéktalanul ki is használja ennek az adottságnak az előnyeit.
A normál C API mellett egy kiterjesztett API is rendelkezésünkre áll, amivel még könnyebb a rekordok lekérdezése. Az `sqlite_get_table()` használatakor visszahívó függvényre nincs szükség. Ez a függvény inkább a hagyományos ügyfélkönyvtárakra emlékeztethet, amiknél átadjuk az SQL-utasítást, majd megkapjuk az eredmény sorait. A kiterjesztett API saját függvények és összesítők hozzáadásával többek között az SQL kibővítését is lehetővé teszi – ezekről a lehetőségekről később még lesz szó.
Végül, ha valamilyen oknál fogva ODBC-felületre van szükséged, örömmel újságolhatom, hogy ***Christian Werner*** munkájának hála, erről sem kell lemondanod. Az ODBC illesztőprogram a ➲ http://www.ch-werner.de/sqliteodbc címről tölthető le.

5. lista Héjpélda

```
#Az SQL-utasítás létrehozása, elhelyezése
#a $sql változóban
read -d ´EOF´ sql<<EOF
select * from person order by lastname
EOF

# Lekérdezés
ret=$(sqlite $db "$sql" 2>&1)
if [ $? = "1" ]
then
    # Hiba történt. Hibajelzés és kilépés.
    printf "Error: $ret.\n"
    printf "SQL: $sql\n"
    exit 1
    else
        # Hiba nem történt, az eredmény
        # kiírása
        for row in $ret
        do
            IFS="|"

            # Változók értékadása
            # a rekordokból
            set $row

            # Hozzárendelések kényelmi
            # okokból
            firstname=$1
            lastname=$2
            dir=$3
            printf "%-14s %-15s\n"
                   ➥$firstname $lastname

            unset IFS
        done
fi
```

Önműködő sorszámozás

Az SQLite a sorozatokat kifejezetten nem támogatja, ám kezeli az önműködően növelt kulcsokat, a MySQL `mysql_insert_id()` függvényével egyenértékű módon. Az elsődleges kulcsot önműködően úgy növelhetjük, hogy `INTEGER PRIMARY KEY` típussal adjuk meg. Az utoljára beillesztett rekord e mezőjének az értékét az `sqlite_last_insert_rowid()` hívással kaphatjuk meg.

BLOB-ok

Az SQLite oszlopaiban bináris adatokat is tárolhatunk, azzal a megkötéssel, hogy rögzítésük csak az első `NULL` karakterrel bezárólag történik meg. Ha bináris adatokat szeretnénk tárolni, először a kódolásukat kell elvégeznünk. Az egyik lehetőség az URL-stílusú kódolás, a másik a base64. Ha nincs semmilyen különleges igényünk, akkor az SQLite két ilyen célú függvényével – `sqlite_encode_binary()` és `sqlite_decode_binary()` – ezt is könnyedén letudhatjuk.

A programszálak biztonsága

Az SQLite pontosan annyira biztonságos, amennyire maga a beágyazóprogram is az. A kérdés többé-kevésbé az `sqlite_open()` által visszaadott SQLite-kapcsolatkezelőhöz

(connection handle) kötődik. Ez az, amit nem szabad megosztani a futtatási környezetek között, minden szálnak saját példányt kell biztosítani. Ha ragaszkodunk ahhoz, hogy a szálak osztozzanak rajta, egy `mutex` segítségével kell védeni. A kapcsolatkezelőket a Unix `fork()` hívások között sem szabad megosztani. Ezt – gondolom – nem kell különösebben magyarázni. Alapelv tehát, hogy minden szál vagy folyamat saját kapcsolatkezelővel rendelkezzen, és semmi gond nem lesz.
Az SQLite a futásidejű viselkedés szabályozására pragmákat használ. A pragmák olyan beállítások, amiket SQL jellegű írásmóddal adhatunk meg. Vannak olyan pragmák, amik – például a gyorstár méretének megadása vagy a szinkrn írások engedélyezése vagy letiltása révén – a teljesítmény finomhangolására szolgálnak. Mások hibakeresésre, az értelmező és a VDBE nyomkövetésére használhatók, illetve az ügyféloldali visszahívó függvénynek átadott adatok mennyisége is hasonló módon szabályozható. Bizonyos pragmák esetében a hatás tartóssága is szabályozható, így az adott beállítás vonatkozhat csak az adott munkamenetre, de lehet állandó jellegű is.
Ha az adott oszlop típusa `BLOB`, `CHAR`, `CLOB` vagy `TEXT`, akkor – és csak akkor – az SQLite lexigrafikusan rendezi azt. Egyéb típusoknál a rendezés numerikusan történik. Az SQLite korábban kizárólag az értékük alapján rendezte az oszlopokat. Ha „ránézésre" számot talált, akkor numerikus, egyébként pedig lexigrafikus rendezést végzett. A levelezési listán rengeteg vita volt ezzel a témával kapcsolatban, így alakultak ki a jelenlegi szabályok, amik a sémában megadott típus révén teszik lehetővé az összehasonlítás módjának kiválasztását.

Parancsfájlkészítő felületek

Mint korábban említettem, az SQLite-hoz számos ügyfélfelület létezik. Ízelítőként a 2. kódrészlet C-példájának Python-változata a *3.*, Perl-egyenértékese pedig a *4. listában* tekinthető meg. Kell ennél egyszerűbb? Az SQLite parancssorból is használható, így rendszerfelügyeleti célokra is alkalmas. A már ismert példa parancssori változatát mutatja be az *5. lista*.
Végül, mivel nem vagyok Java-, Tcl-, Ruby-, Delphi-, Lua-, Objective C-, PHP-, Visual Basic-, .NET-, Mono-, DBExpress-, wxWindows-, Euphoria- vagy REXX-programozó, mindenkinek javaslom, hogy az SQLite Wiki segítségével kutassa fel a hozzá hasonló érdeklődésűeket, és használja bátran az általa választott felületet (➲ http://cvs.hwaci.com:2080/sqlite/wiki?p=SqliteWrappers).

Az SQLite kiterjesztése

Az SQLite nagyszerű C-keretrendszerrel bír, amiben saját, SQL-ből meghívható függvények és összesítők készíthetők. Egyes burkolók – mint például a Python-burkoló – azt is lehetővé teszik, hogy ezeket a bővítmény nyelvén készítsük el. Egy SQL-utasítás – például `INSERT INTO orders purchase_date values CURRENT_TIME()` – jó szemléltető alanya egy visszahívó függvénynek (lásd a 6. listát, 50. CD Magazin/SQL könyvtár). Mindössze be kell jegyezni a függvényt, és máris lehet használni (lásd a 7. listát, 50. CD Magazin/SQL könyvtár).
Az SQLite összes beépített függvénye – mint az `avg()`, `min()`, `max()` és `sum()` – ezzel az API-val van megvalósítva, az egyetlen kivétel ez alól a `typeof()`. Felhasználói összesítők ugyanilyen könnyen adhatók hozzá. A `SELECT variance(age) from population` végrehajtása például nagyon közel áll a függvények készítéséhez. Ezt azonban már inkább a kedves olvasóra bízom, amolyan házi feladatként. Egy jó tanács: a ***func.c*** fájlban érdekes példákat lehet találni. Az SQLite a függvényekhez hasonlóan az összesítőket is az API segítségével valósítja meg.

Felügyelet

Felügyeleti célokra az SQLite esetében egy `sqlite` nevű segédprogram szolgál, ami a MySQL- és PostgreSQL-használók számára nem fog meglepetéseket okozni. Héj és parancssori módja egyaránt létezik. Héjból az adott adatbázis tábláinak nevét, sémáját és indexeit tekinthetjük meg, illetve SQL-utasításokat hajthatunk végre, akár a parancssorba gépelve, akár külső állományból véve őket. Az adatok és a VDBE kimenetének megjelenítésére is jól használható.
Az adatok betöltése és kimentése héjból és parancssorból egyaránt megoldható – igaz, az utóbbival könnyebb dolgunk lesz. Ha egy fájl érvényes DDL/DML utasításokat tartalmaz (a neve legyen ***dump.sql***), akkor a következő módon tölthető be az adatbázisba (ennek neve legyen db):

```
sqlite db < dump.sql
```

Ilyenkor létrejön egy db nevű adatbázis, mégha korábban nem is létezett. Ha az adatbázist ki szeretnénk íratni, a következő módon kell eljárnunk:

```
sqlite db .dump > dump.sql
```

Az SQLite kiváló teljesítményt nyújt. Széles körű alkalmazhatósága, könnyű használata, hordozhatósága, gyorsasága, méretezhetősége, kis mérete és letisztult kódja olyan könyvtárrá teszik, ami egyetlen programozható eszköztárából sem hiányozhat. Felhasználói szerződése mindenkinek a lehető legnagyobb szabadságot kínálja. Az SQLite Project mindig örömmel fogadja az új felhasználókat és fejlesztőket, ahogy az újabb ötleteket és vitatémákat is. Remélem, hogy mindenki ugyanolyan örömmel ismerkedik meg vele és kezdi használni, ahogy én is tettem.

A cikkhez tartozó listák megtalálhatóak az 50. CD Magazin/SQL könyvtárában.

Linux Journal 2003. június, 110. szám

Michael Owens (mike@mikesclutter.com)
Programozó, eredetileg kémikus. Egy ingatlankezelő cégnél dolgozik a texasi Dallasban, ahol Linux alatt fejleszti a cég belső alkalmazásait. A PySQLite életre hívója és társfejlesztője.

KAPCSOLÓDÓ CÍMEK

Az SQLite készítője, *D. Richard Hipp* magas szintű támogatást és tanácsadást nyújt az SQLite-hoz.
A ➲ http://www.hwaci.com/sw/sqlite/support.html címen további tájékoztatás található róla.
Az SQLite weboldala felbecsülhetetlen értékű forrás, itt találjuk az SQLite SQL C API-jának teljes leírását, illetve az alrendszerek, köztük a VDBE ismertetőjét.
➲ http://www.sqlite.org
SQLite levelezési lista
➲ http://groups.yahoo.com/subscribe/sqlite