Elektronski fakultet Niš

Seminarski rad iz predmeta

Sistemi za upravljanje bazama podataka

Tema:

Interna struktura i organizacija skladišta podataka u **SQLite** bazi podataka

Student:

Milica Mladenović, 1061

Mentor:

Aleksandar Stanimirović

Sadržaj

Uvod

SQLite sistem za upravljanje bazama podataka je biblioteka koja je kompaktna, samostalna, nezavisna od drugih komponenti, bez potrebe za konfiguracijom ili posebnom serverskom komponentom. Kod SQLite-a je javno dostupan, tj. besplatan za privatnu i komercijalnu upotrebu, ali je isto tako dostupna i profesionalna podrška.

Kompaktnost se ogleda u činjenici da sa svim uključenim opcijama njegova veličina ne prelazi 500KB, a ukoliko se nepotrebne opcije uklone manji je i od 300KB. U najmanjoj konfiguraciji je popularan na uređajima koji nemaju puno memorije kao što su mobilni telefoni, MPI plejeri i slični uređaji. Ipak, postoji kompromis između količine memorije i brzine, ali su performanse obično prilično dobre i na uređajima sa malo memorije.

Samostalnost proizilazi iz činjenice da zahteva minimalnu podršku spoljašnjih biblioteka i samog operativnog sistema. Za razliku od većine drugih SQL baza podataka, SQLite nema odvojeni proces na nekom serveru, tj. podaci se čitaju i pišu direktno s diska iz jedne jedine datoteke u kojoj je zapisana celokupna struktura baze podataka, kao i svi podaci koji se nalaze unutar nje. Od sistema koji takođe ne zahtevaju serversku komponentu, SQLite je jedini koji dopušta da više aplikacija pristupa istoj bazi podataka istovremeno.

SQLite sitem je jedan od najrasprostranjenijih sistema za upravljanje bazom podataka, koji se koristi od malih projekata do mnogo većih i značajnijih, a neke od kompanija koje ga koriste su *Adobe*, *Airbus*, *Apple*, *Facebook*, *Google* itd.

Slika 1. - SQLite logo

Osobine SQLite sistema za upravljanje bazama podataka

U nastavku su date neke od glavnih i najznačajnijih osobina SQLite sistema:

- Transakcije su atomične, konzistentne, izolovane i trajne čak i nakon pada sistema ili nestanka struje.
- *Zero-configuration* - SQLite ne zahteva instalaciju niti bilo kakav postupak podešavanja pre korišćenja. Ne postoji proces na serveru koji treba pokrenuti, zaustaviti ili konfigurisati. SQLite jednostavno radi.
- Potpuno opremljena SQL implementacija s naprednim mogućnostima.
- Kompletna baza podataka je smeštena u jednoj datoteci na disku na više platformi.
- Podržava baze podataka veličine nekoliko terabajta i stingove i blobove veličine nekoliko gigabajta.
- Jednostavan i lak za upotrebu.
- Brz; u nekim slučajevima SQLite je čak brži i od direktnog ulazno-izlaznog faj sistema.
- *Cross-platform*: podržani sistemi su Android, iOS, Mac, Solaris, VxWorks i Windows. Lako se prenosi na druge sisteme.
- Samostalan, nema spoljašnjih zavisnosti.
- Dolazi sa samostalnim *CLI* (command-line interface) klijentom koji se može koristiti za administraciju SQLite baze podataka.

In-Memory baze podataka

SQLite baza podataka se obično čuva u jednoj običnoj datoteci na disku. Međutim, u određenim okolnostima, baza podataka se može čuvati i u memoriji.

Najčešći način primoravanja SQLite baze podataka da postoji isključivo u memoriji je otvaranje baze pomoću posebnog naziva datoteke ":memory:". Drugim rečima, umesto prosleđivanja stvarnog imena datoteke sa diska, jednoj od funkcija sqlite3_open(), sqlite3_open16() ili sqlite3_open_v2(), koje otvaraju datoteku SQLite baze na način na koji je specificirano argumentom, prosleđuje se string ":memory:".

Na primer:

```
rc = sqlite3_open(":memory:", &db);
```

Kada se ovo uradi, nijedna datoteka sa diska se ne otvara. Umesto toga, nova baza podataka je u potpunosti kreirana u memoriji. Baza podataka prestaje da postoji čim se zatvori konekcija sa njom. Svaka ":memory:" baza se razlikuje od svake druge. Prema tome, otvaranjem dve konekcije baze podataka, sa imenom datoteke ":memory:", kreiraće se dve nezavisne baze u memoriji.

Specijalno ime datoteke ":memory:" se može korisiti svuda gde je ime datoteke baze podataka dozvoljeno. Na primer, može se koristiti kao ime datoteke u naredbi ATTACH:

```
ATTACH DATABASE ':memory:' AS aux1;
```

Treba imati na umu da za upotrebu specijalnog ":memory:" imena i stvaranje baze podataka u memoriji, ne mora biti dodatnog teksta u nazivu datoteke. Ovaj naziv takođe funkcioniše kada se koriste URI nazivi datoteka. Na primer:

```
rc = sqlite3_open("file::memory:", &db);
```

ili,

```
ATTACH DATABASE 'file::memory:' AS aux1;
```

Tipovi podataka u SQLite sistemu

Većina SQL sistema baza podataka koristi statičko, strogo "tipovanje" koje omogućava da se tip vrednosti podatka određuje prema njenom kontejneru, odnosno određenoj koloni u koju je ta vrednost smeštena. SQLite koristi opštiji sistem dinamičkih tipova. Naime, u SQLite-u je tip podatka povezan sa samom vrednošću podatka, a ne sa njenim kontejnerom. Dinamički tipovi u SQLite-u su kompatibilni sa većinom uobičajenih statičkih tipova ili drugim sistemima baza podataka u smislu da se SQL izrazi nad statičkim tipovima podataka ponašaju isto u SQLite sistemu koji radi sa dinamičkim tipovima. To omogućava SQLite sistemu da radi stvari koje nisu moguće u tradicionalnim, bazama podataka koje rade sa statičkim tipovima podataka.

SQLite ima nekoliko tipova podataka, dok se svi ostali moraju svesti ili pretvoriti u neki od ovih tipova:

- **NULL -** predstavlja praznu vrednost, odnosno da nema podataka,
- **INTEGER -** predstavlja označen ceo broj i moze biti veličine 1, 2, 3, 4, 6 ili 8 bajtova u zavisnosti od veličine podatka koji treba da se unese u polje ovog tipa,
- **REAL -** predstavlja označen broj sa pokretnim zarezom i veličine je 8 bajtova,
- **TEXT -** predstavlja znakovni niz (*engl*. string) i može biti kodiran sa UTF-8, UTF-16BE ili UTF-16LE formatom znakova,
- **BLOB -** predstavlja podatak bilo koje vrste i smešta se isti kao što je bio pre upisa u bazu ili identičan u binarnom zapisu.

Svi ostali tipovi se svode na neki od ovih tipova. Na primer, SQLite ne podržava Bulov tip podataka koji je predstavljen sa tačno i netačno, pa se umesto tog tipa može koristiti integer tip koji sa vrednošću 1 označava tačno, a sa vrednošću 0 netačno. Sličan primer je i sa datumom. Ako je datum predstavljen u formatu "gggg:mm:dd-ss:mm:ss" može se predstaviti kao text, odnosno niz znakova. Vreme takođe može da se predstavi kao integer tip u milisekundama.

Kako bi se povećala kompatibilnost SQLite sistema sa ostalim sistemima za upravljanje bazama podataka, ovaj sistem podržava koncept pod nazivom *type affinity*, odnosno afinitet tipa. Afinitet tipa kolone je preporučeni tip za podatke skladištene u toj koloni. Ideja kod ovog koncepta je da je tip preporučen, nije neophodan. Sve kolone i dalje mogu da skladište bilo koji tip podatka, a samo neke, u odnosu na izbor, će preferirati jedan tip za skladištenje podataka i taj tip se naziva njen afinitet.

Afinitet kolone se određuje deklarisanim tipom kolone, prema sledećim pravilima i prikazanom redosledu:

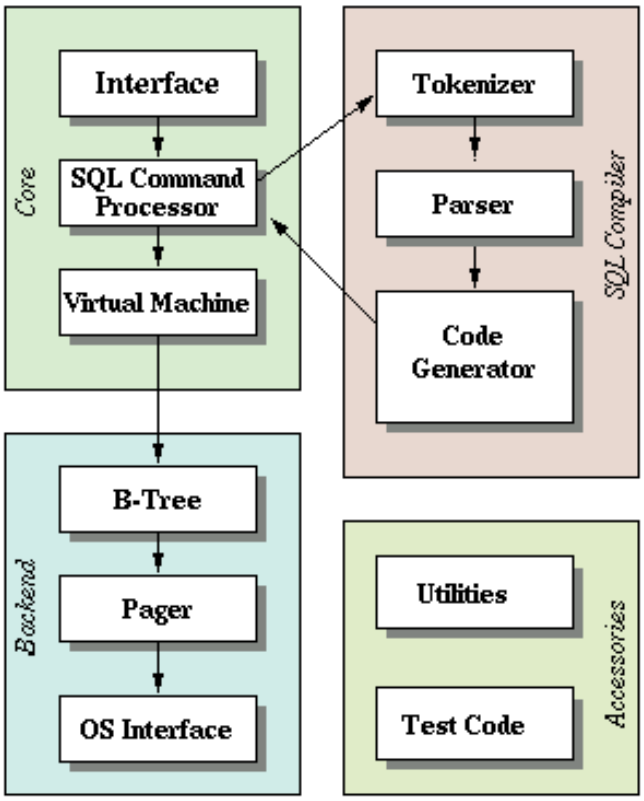
1. Ako deklarisani tip kolone sadrži string "INT" tada mu je dodeljen INTEGER afinitet.
2. Ako deklarisani tip kolone sadrži bilo koji od stringova "CHAR", "CLOB" ili "TEXT" onda ta kolona ima TEXT afinitet.
3. Ako deklarisani tip kolone sadrži string "BLOB" ili ako nije naveden nijedan tip, tada mu je dodeljen BLOB afinitet.
4. Ako deklarisani tip kolone sadrži neki od stringova "REAL", "FLOA" ili "DOUB", onda ta kolona ima REAL afinitet.
5. U svim ostalim slučajevima, afinitet je NUMERIC.

Redosled ovih pravila za određivanje afiniteta je veoma važan. Kolona kojoj je deklarisani tip "CHARINT" odgovara pravilima 1 i 2, ali prvo pravilo ima prioritet i u ovom slučaju afinitet kolone bi bio INTEGER.

Arhitektura SQLite sistema za upravljanje baze podataka

Arhitektura SQLite sistema je podeljena u dve različite sekcije: jezgro i bekend. Jezgro sadrži interfejs, tokenizer, parser, generator koda i virtuelnu mašinu koja kreira nalog za izvršenje transakcija baze podataka. Bekend deo sadrži B-stablo, pager (straničnik) i OS interfejs za pristup fajl sistemu. Tokenizer, parser i generator koda se jednim imenom nazivaju kompajler koji generiše skup instrukcija na mašinskom jeziku koji se pokreće na virtuelnoj mašini.

Na slici je dat blok dijagram koji prikazuje osnovne komponente SQLite sistema i način na koji su one povezane. U nastavku teksta sledi kratak pregled svake od komponenti.



Slika 2. - Arhitektura SQLite sistema

Interfejs

Veliki deo interfejsa (C jezika) se nalazi u izvornim datotekama main.c i legacy.c. Elementi SQLite interfejsa se mogu grupisati u tri kategorije:

1. Lista objekata - lista svih apstraktnih objekata i tipova podataka koji se koriste u SQLite bazi podataka. Ukupno ima nekoliko desetina objekata, ali dva najvažnija objekta su objekat za povezivanje baze podataka (sqlite3) i objekat pripremljene naredbe (sqlite3_stmt)
2. Lista konstanti - lista numeričkih konstanti koje se koriste u SQLite bazi podataka i predstavljaju se pomoću #defines u header fajlu sqlite3.h.
3. Lista funkcija - lista svih funkcija i metoda koje rade sa objektima i koriste ili vraćaju konstante. Postoji mnogo funkcija ali većina aplikacija koristi samo nekoliko njih.

Da bi se izbeglo preklapanje u imenima, svi eksterni simboli u SQLite biblioteci počinju prefiksom sqlite3. Ti simboli koji su namenjeni za eksternu upotrebu (drugim rečima, koji formiraju API za SQLite) dodaju donju crticu, odnosno počinju sa sqlite3_.

Tokenizer

Kada string, koji sadrži SQL izraze treba biti obrađen, prvo se šalje tokenizeru koji razbija SQL tekst u tokene i šalje ih jedan po jedan parseru. Tokenizer se ručno kodira u datoteci *tokenize.c*. Treba imati na umu da u ovom dizajnu tokenizer zove parser. Neko ko je upoznat sa Yacc i Bison generatorima je možda navikao na obrnut slučaj - da parser zove tokenizer. Ipak, situacija kada tokenizer zove parser je bolja jer je taj način brži i sigurniji.

Parser

Parser dodeljuje značenje tokenima na osnovu njihovog konteksta. Parser za SQLite sistem je generisan pomoću parser generatora pod nazivom *Lemon*. On čita gramatiku ulaznog jezika i generiše C kod da bi implementirao parser za taj jezik. Lemon nema svoje source skladište, već se sastoji od nekoliko datoteka u SQLite source stablu:

- lemon.html - originalna detaljna Lemon dokumentacija o upotrebi sa uputstvima za programere,

- lemon.c - izvorni kod uslužnog programa koji čita gramatički fajl i generiše odgovarajući kod za raščlanjivanje,
- lempar.c - tempejt za generisani parser C koda. Pomoćni *lemon* program čita ovaj obrazac i ubacuje dodatni kod da bi stvorio parser.

Lemon generiše LALR(1) parser (*engl*. Look-Ahead LR). Njegovo delovanje je slično poznatijim generatorima parsera Yacc i Bison, ali on dodaje bitna poboljšanja:

- gramatička sintaksa je manje sklona greškama zbog toga što se koriste simbolička imena za semantičke vrednosti
- U Lemon-u tokenizer zove parser. Yacc, na primer, deluje obrnuto, odnosnu parser zove tokenizer. Lemon pristup je bezbedan i siguran, dok Yacc koristi globalne promenljive i zbog toga nije siguran.
- Lemon ima koncept ne-terminalnog destruktora koji se može koristiti za ponovno prikupljanje memorije ili drugih resursa nakon sintaksičke greške ili nekog drugog prekinutog raščlanjivanja. Drugim rečima, ne dolazi do curenja memorije ako se nađu greške u sintaksi.

Gramatička datoteka koja pokreće Lemon i koja definiše SQL jezik koji SQLite razume nalazi se u *parse.y*. S obzirom na to da je Lemon program koji se obično ne nalazi na razvojnim mašinama, njegov celokupni izvorni kod (zapravo samo jedna C datoteka) je uključen u SQLite distribuciju u podmeniju *tool*.

Generator koda

Nakon što parser sastavi tokene u parsovano stablo, pokreće se generator koda za analizu tog stabla i generisanje bajt koda koji izvršava rad SQL izraza. Objekat pripremljene naredbe predstavlja kontejner za bajt kod.

```
typedef struct sqlite3_stmt sqlite3_stmt;
```

Instanca ovog objekta predstavlja jedinstveni SQL izraz koji je sastavljen u binarni oblik i spreman za evaluaciju. Svaki SQL izraz se mora konvertovati u pripremljeni pre nego što se pokrene.

U generatoru koda postoji mnogo datoteka, uključujući: *attach.c, auth.c, build.c, delete.c, expr.c, insert.c, pragma.c, select.c, trigger.c, update.c, vacuum.c, where.c, wherecode.c,* i *whereexpr.c.* Na primer, *expr.c* obrađuje generisanje koda za izraze, *where.*c* obrađuje generisanje koda za WHERE klauzule na naredbama SELECT,

UPDATE i DELETE. Datoteke: *attach.c, delete.c, insert.c, select.c, trigger.c update.c* i *vacuum.c* obrađuju generisanje koda za SQL izraze sa istim imenima. Svi ostali SQL izrazi su kodirani iz *build.c* datoteke.

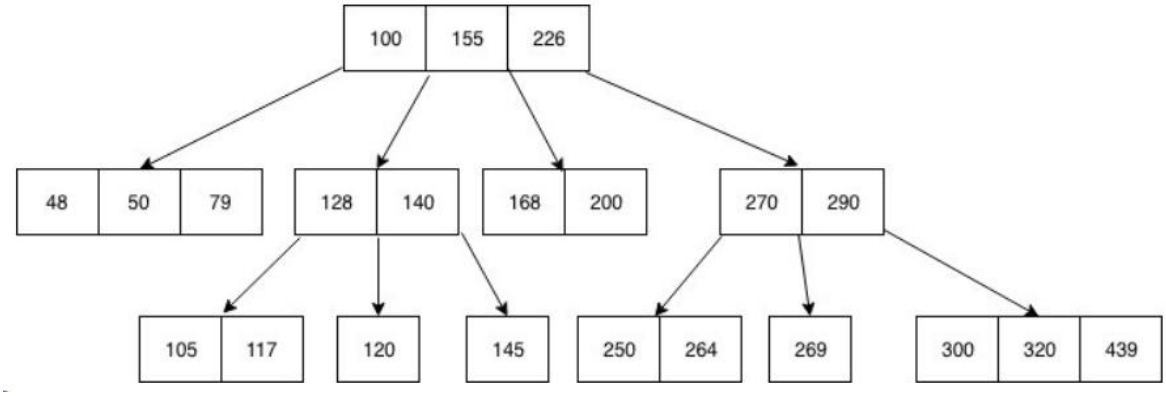
Generator koda i posebno logika u datotekama *where.*c* i *select.c* se ponekad nazivaju query planner. Za bilo koji određeni SQL izraz može postojati milion različitih algoritama za implementaciju tog izraza, zavisno od složenosti samog izraza i osnovne šeme baze podataka. Query planner je taj koji teži da odabere najbolji algoritam od svih mogućih izbora koji daje odgovor sa minimalnim troškovima.

Virtuelna mašina

Virtuelna mašina pokreće skup operacija, generisan od strane generatora koda. Sve SQL komande, uključujući *insert, delete, update, select,* se konvertuju u skup instrukcija na mašinskom jeziku, a zatim se pokreću na virtuelnoj mašini. Sama virtualna mašina se u potpunosti nalazi u jednoj izvornoj datoteci - *vdbeInt.h*. SQLite implementira SQL funkcije koristeći Callback pozive u rutinu C jezika. Čak su i ugrađene SQL funkcije implementirane na ovaj način.

B-Tree

B-Tree je struktura podataka koja se koristi za čuvanje podataka u obliku stabla na osnovu njihove vrednosti. B-Tree čuva podatke tako što svaki čvor sadrži ključeve u rastućem redosledu. Svaki od ključeva ima dva pokazivača na dva čvora potomka. Ključevi čvora potomka na levoj strani su manji od trenutnog ključa, dok su ključevi čvora potomka na desnojj strani veći. Ako jedan čvor ima n ključeva, onda može imati maksimalno n+1 čvorova potomaka. U nastavku je dat primer reprezentacije B-stabla.



Slika 3. - Reprezentacija B-stabla

Baze podataka koriste B-Tree strukturu za skladištenje indeksa kako bi poboljšale svoje performanse. Kao što je već rečeno, svaki čvor B-stabla sadrži listu ključeva koji se koriste za indeksiranje. Indeksi B-stabla se mogu kreirati za svaku kolonu u tabeli. Svako B-stablo ima korensku stranicu, koja predstavlja polaznu osnovu za bilo koju vrstu pretrage podataka.

SQLite skladišti šemu baze podataka u tabeli koja je poznata pod nazivom master_table i koja se uvek čuva na prvoj stranici baze podataka.

Pager (straničnik)

Stranice su najmanja jedinica transakcije baze podataka u fajl sistemu. Kada baza treba da pročita neke podatke iz datoteke, ona to zahteva kao stranicu. Stranice su numerisane i počinju od 1. Prva stranica se naziva korenska stranica. Njihova veličina je konstantna. Podrazumevana veličina stranice je 4096 bajtova, ali može biti i bilo koje veličine stepena dvojke između 512 i 65536 bajtova.

B-Tree modul zahteva informacije sa diska na tim stranicama. Keš stranice je odgovoran za čitanje, pisanje i keširanje ovih stranica. Upravljački program B-stabla zahteva određene stranice iz keša stranice i obaveštava ga kada želi da modifikuje stranice ili da komituje ili povrati promene. Keš stranice obrađuje sve "neuredne" detalje kako bi se osiguralo da se zahtevima upravlja brzo, sigurno i efikasno.

Osnovna implementacija keša stranice se nalazi u *pager.c* datoteci. Interfejs između podsistema keš stranice i ostatka SQLite baze podataka je definisan zaglavljem datoteke *pager.h*.

```c
/*
Open pager with the given file name
*/
int sqlitepager_open(Pager **ppPager,const char *zFilename,int nPage,int nEx);
```

```c
/*
Get page specified by the page number
*/
int sqlitepager_get(Pager *pPager, Pgno pgno, void **ppPage);
```

```c
/*
Start to write data into a page specified in pData
*/
int sqlitepager_write(void *pData);
```

```c
/*
Commit page changes into the file
*/
int sqlitepager_commit(Pager*);
```

```c
/*
Close the connection to the file
*/
int sqlitepager_close(Pager *pPager);
```

OS Interfejs

Da bi se osigurala prenosivost između operativnih sistema, SQLite sistem koristi apstraktni objekat pod nazivom VFS (*engl*. Virtual File System). Kako se pristup datotekama na Unix-u i Windows-u razlikuje, VFS pruža API za pristup datotekama bez obzira na vrstu operativnog sistema na kome se pokreće. Ovaj API uključuje funkcije za otvaranje, pisanje, čitanje i zatvaranje datoteke.

Korisni dodaci

Alokacija memorije, funkcije za poređenje stringova, prenosive funkcije za konvertovanje teksta u broj i drugi uslužni programi, nalaze se u datoteci *util.c*. Tablice simbola koje koristi parser se održavaju pomoću heš tabela smeštenih u datoteci *hash.h*. SQLite ima svoju privatnu implementaciju funkcije printf() (sa nekim ekstenzijama) u datoteci *printf.c* i svoj sopstveni pseudo-slučajni generator brojeva u *random.c*.

Arhitektura - rezime

SQLite ima jednostavnu arhitekturu i vrlo čitljiv kod. Pager (straničar) pruža apstraktni sloj za čitanje i upis podataka u fajl sistem u vidu blokova fiksne veličine. B-Tree struktura pruža efikasan način za skladištenje podataka u memoriju kako bi se brže pristupalo podacima. Kada SQL naredbe stignu u SQLite sistem, on ih konvertuje u SQLite mašinski kod koji se pokreće na virtuelnoj mašini. I na kraju, rezultat se vraća korisniku preko API-ja.

Datoteka baze podataka

Kompletna SQLite baza podataka se obično nalazi u jednoj datoteci na disku koja se naziva glavna datoteka baze podataka.

Za vreme transakcije, SQLite skladišti dodatne informacije u drugu datoteku koja se naziva rollback journal ili, ukoliko se baza nalazi u WAL (*engl*. **W**rite-**A**head **L**ogging) režimu, u write-ahead log datatoteku.

Rollback journal je datoteka koja je povezana sa svakom datotekom u SQLite bazi podataka i koja sadrži informacije koje se koriste za vraćanje datoteke u njeno inicijalno stanje tokom transakcije. Ona se nalazi u istom direktorijumu kao i datoteka baze podataka i ima isto ime, uz dodatak stringa -journal. Može postojati samo jedan rollback journal povezan sa bazom i stoga u jednom trenutku može biti otvorena samo jedna transakcija za upis u bazu podataka. Ukoliko dođe do prekida transakcije zbog pada aplikacije, pada operativnog sistema,nestanka struje i slično, baza može ostati u nekonzistentnom stanju. Kada SQLite sledeći put pokuša da otvori datoteku baze podataka, biće detektovano prisustvo rollback journal-a koji će automatski vratiti bazu u stanje u kom je bila na početku nedovršene transakcije.

SQLite, takođe, podržava novi mehanizam kontrole transakcija koje se naziva Write-Ahead-Logging, ili skraćeno WAL. Kada se baza podataka nalazi u ovom režimu, sve konekcije sa tom bazom moraju takođe koristiti WAL režim. Određena baza podataka može koristiti ili rollback journal, ili WAL režim, ali ne obe stvari istovremeno. Wal se nalazi u istom direktorijumu kao i datoteka baze podataka i ima isto ime, uz dodatak stringa -wal. Wal fajl se sastoji od zaglavlja iza koga sledi nula ili više frejmova. Svaki frejm beleži pregledan sadržaj jedne stranice u datoteci baze podataka. Sve promene u bazi se beleže upisom frejmova u WAL. Transakcije se izvršavaju kada se upiše frejm koji sadrži signal za izvršavanje. Jedan WAL može i obično beleži više transakcija. Povremeno se sadržaj Wal fajla vraća nazad u

datotektu baze podataka u operaciji koja se zove kontrolna tačka (*engl*. checkpoint). Jedna Wal datoteka se može koristiti više puta.

Hot Journals

Ukoliko se aplikacija ili računar sruše pre nego što se transakcija završi, tada dnevnik povratka ili dnevnik pisanja unapred sadrži informacije potrebne za povratak glavne datoteke baze podataka u konzistentno stanje. U tom slučaju, ovi dnevnici se nazivaju hot journal odnosno "vrući dnevnici". Hot journals su samo faktor koji učestvuje u oporavku sistema od grešaka i zbog toga su veoma retki, ali ipak, oni su deo SQLite baze podataka i ne mogu se zanemariti.

Stranice

Glavna datoteka baze podataka se sastoji od jedne ili više stranica. Sve stranice u bazi su iste veličine koja predstavlja stepen dvojke u opsegu od 512 do 65536 bajtova. Veličinu stranica u datoteci baze određuje dvobajtni ceo broj koji se nalazi na rastojanju od 16 bajtova od početka datoteke. Stranice su numerisane počevši od 1. Maksimalni broj stranica je 2^{31}-1, a minimalni jedna stranica veličine 512 bajtova.

U uobičajenoj upotrebi, SQLite baze podataka se kreću u opsegu od nekoliko kilobajta do nekoliko gigabajta, mada u proizvodnji postoje i baze veličine nekoliko terabajta.

Pre nego što se bilo koja stranica koja sadrži neke informacije promeni, originalni, nepromenjeni sadržaj te stranice se upisuje u rollback journal. Kao što je već rečeno u prethodnom tekstu, ukoliko dođe do prekida transakcije, rollback journal se može iskoristi za vraćanje baze podataka u njeno originalno stanje.

The Lock-Byte stranica

Lock-Byte je jedna stranica u datoteci baze podataka koja sadrži bajtove u opsegu od 1073741824 do 1073742335. Datoteka baze čija je veličina manja ili jednaka od donje granice, uopšte ne sadrži ovu stranicu, a datoteka čija je veličina veća od gornje granice sadrži tačno jednu ovakvu stranicu.

Lock-Byte stranica je nastala iz potrebe da se podrži Win95 koji je bio glavni operativni sistem kada je dizajniran ovaj format datoteke i koji je podržavao samo

obavezno zaključavanje datoteke. Svi moderni operativni sistemi podržavaju sigurnosno zaključavanje datoteka, tako da ova stranica više nije potrebna, ali je zadržana radi kompatibilnosti sa ranijim verzijama.

Freelist stranice

Datoteka baze podataka može sadržati jednu ili više stranica koje nisu u aktivnoj upotrebi. Neiskorišćene stranice se, na primer, mogu pojaviti kada se informacije izbrišu iz baze. One se čuvaju u freelist-u i ponove se koriste kada su potrebne dodatne stranice. Freelist stranice u listovima ne sadrže nikakve informacije i SQLite izbegava čitanje ili upis u ove stranice kako bi se redukovao broj ulaza/izlaza na disku. Broj ovakvih stranica se čuva u zaglavlju datoteke kao četvorobajtni ceo broj na rastojanju od 32 bajta od početka datoteke.

B-tree stranice

B-Tree algoritam omogućava skladištenje tipa ključ/podatak pomoću jedinstvenih i uređenih ključeva. SQLite koristi dva tipa B-Tree algoritma:

1. B*-Tree algoritam koji skladišti sve podatke u listovima stabla (poznat pod nazivom table b-tree)
2. B-Tree koja skladišti podatke i ključeve zajedno i u unutrašnjim, i u stranicama koje se nalaze u listovima

U SQLite implementaciji, originalni B-Tree algoritam skladišti samo ključeve, u potpunosti izostavljajući podatke i naziva se index b-tree.

B-tree stranica je ili unutrašnja stranica ili stranica koja se nalazi u listovima. U okviru unutrašnje B-tree stranice, svaki ključ i pokazivač na njegovog direktnog levog suseda se kombinuju u strukturu koja se naziva ćelija. Pokazivač na desni deo se drži odvojeno. B-tree stranica koja se nalazi u listovima nema pokazivače. Podaci su takođe u strukturi ćelije.

B-Tree stranica je ili tipa *table b-tree* ili *index b-tree*. Sve stranice unutar kompletnog B-stabla moraju biti istog tipa: ili tabela ili indeks. Postoji jedan *table b-tree* u datoteci baze podataka za svaku tabelu u šemi baze podataka. Isto tako postoji jedan *index b-tree* za svaki indeks u šemi. Svaki unos u *table b-tree* se sastoji od 64-bitnog celobrojnog ključa i do 2147483647 bajtova proizvoljnih podataka. Unutrašnji *table b-tree* sadrže samo ključeve i pokazivače na potomke. Svi podaci se nalaze u *table*

b-tree koji se nalaze u listovima stabla. Svaki unos u *index b-tree* se sastoji od proizvoljnog ključa dužine do 2147483647 bajtova i nema podataka.

Zaključak

SQLite je javno dostupni softverski paket koji predstavlja sistem za upravljanje relacionom bazom podataka (Relational database management system - RDBMS). U nazivu sistema reč Lite (*engl*. - lak, jednostavan) odnosi se na jednostavnost konfiguracije sistema i male zahteve za računarskim resursima, a nipošto na funkcionalnosti koje sistem nudi.

Većina RDBMS rešenja oslanja se na servere na kojima se izvršava veliki broj procesa zaduženih za upravljanje konekcijama, upisom i čitanjem iz fajlova, keširanjem podataka, optimizacijom i obradom upita. Za razliku od konvencionalnih rešenja, SQLite se ne oslanja na postojanje zasebnog servera. Kompletna funkcionalnost RDBMS ugrađuje se u jedan fajl koji se može integrisati u aplikaciju koja ga koristi. Eliminacijom servera eliminiše se i kompleksnost sistema. Komponente su jednostavne i nije potrebna velika podrška operativnog sistema. Jedini deljivi resurs između aplikacija je fajl baze podataka kome aplikacije pristupaju, a on se nalazi na disku. Premeštanje baze i kreiranje rezervne kopije postiže se jednostavnim kopiranjem fajla baze podataka.

SQLite je odličan izbor baze podataka za aplikacije kojima je potrebna prenosivost i koje ne zahtevaju proširenja u budućnosti. Primeri uključuju lokalne korisničke i mobilne aplikacije i igre. U slučajevima kada aplikacija treba direktno da čita i piše datoteke na disk, može biti korisno korišćenje ove baze za dodatnu funkcionalnost i jednostavnost korišćenja SQL-a.

Reference

1. https://www.sqlite.org/index.html, SQLite Home Page
2. https://dzone.com/articles/how-sqlite-database-works, How SQLite Database works
3. https://sr.wikipedia.org/wiki/SQLite, SQLite - Vikipedija