SQLite

Table des matières

SQLite est une bibliothèque écrite en C qui propose un moteur de base de données relationnelle accessible par le langage SQL et implémente en grande partie les propriétés ACID.

Son code source est dans le domaine public

1. Introduction

Contrairement aux serveurs de bases de données traditionnels, comme MySQL ou PostgreSQL, sa particularité est de ne pas reproduire le schéma habituel client-serveur mais d'être directement intégrée aux programmes. L'intégralité de la base de données (déclarations, tables, index et données) est stockée dans un fichier indépendant de la plateforme.

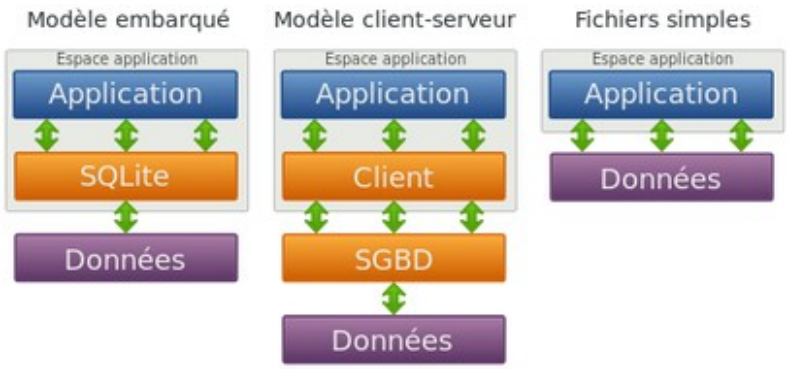
SQLite est le moteur de base de données le plus distribué au monde, grâce à son utilisation dans de nombreux logiciels grand public comme Firefox, Skype, Google Gears, dans certains produits d'Apple, d'Adobe et de McAfee et dans les bibliothèques standards de nombreux langages comme PHP ou Python. De par son extrême légèreté (moins de 300 Kio), il est également très populaire sur les systèmes embarqués, notamment sur la plupart des smartphones modernes : l'iPhone ainsi que les systèmes d'exploitation mobiles Symbian et Android l'utilisent comme base de données embarquée.

2. Caractéristiques

2.1. Base de données embarquée

La majorité des systèmes de gestion de base de données sont construits selon le paradigme client-serveur : une bibliothèque logicielle cliente est utilisée dans une ou plusieurs applications alors que le moteur de base de données fonctionne sur une machine différente.

SQLite, au contraire, est directement intégrée dans l'application qui utilise sa bibliothèque logicielle, avec son moteur de base de données. L'accès à une base de données SQLite se fait par l'ouverture du fichier correspondant à celle-ci : chaque base de données est enregistrée dans un fichier qui lui est propre, avec ses déclarations, ses tables et ses index mais aussi ses données.



La suppression de l'intermédiaire entre l'application et les données permet de réduire légèrement la latence d'accès aux données comparée aux systèmes utilisant le paradigme client-serveur.

Cependant, cette architecture pose plusieurs problèmes :

- lorsqu'un nombre important de clients accèdent à une même base, si un des clients commence la lecture d'une partie, la totalité de la base est bloquée en écriture pour les autres clients, et inversement. Ces derniers sont alors mis en attente durant ce laps de temps, ce qui peut être contre-performant ;
- il est très compliqué de répartir la charge sur plusieurs machines : l'utilisation d'un système de fichiers sur le réseau engendre des latences et un trafic considérable, du fait que la base doit être rechargée chez le client à chaque ouverture.
- avec des bases de grande envergure, il est impossible de diviser la base en plusieurs parties ou fichiers dans le but de distribuer la charge sur plusieurs machines.

Il est conseillé d'utiliser SQLite là où les données ne sont pas centralisées et où l'expansion de la taille de la base ne risque pas de devenir critique. Si la base de données a pour but de centraliser une grande masse de données et de les fournir à un grand nombre de clients, il est préférable d'utiliser des SGBD basés sur le paradigme client-serveur. SQLite a pour objectif de remplacer les fichiers texte et non les serveurs de base de données traditionnels.

2.2. Portabilité

La bibliothèque est entièrement écrite en C-ANSI, la version normalisée du langage de programmation C, et n'utilise aucune autre bibliothèque externe que la bibliothèque standard du langage. Ceci rend SQLite compilable sans modification majeure sur toutes les architectures informatiques mettant à disposition un compilateur C respectant la norme ANSI.

Les fichiers de base de données de SQLite sont entièrement indépendants du système d'exploitation et de l'architecture sur laquelle ils sont utilisés. Le même fichier de base de données peut être utilisé sur deux architectures ayant un fonctionnement radicalement différent, SQLite fournissant une couche d'abstraction transparente pour le développeur.

2.3. Types de données

SQLite utilise un typage dynamique : lors de la création d'une nouvelle table dans la base de données, c'est un type recommandé ou d'affinité, de la donnée à stocker dans la colonne qui est renseigné et non un type qui définit la façon dont celle-ci sera représentée en mémoire. Lorsque des données seront entrées dans la base, SQLite tentera de convertir les nouvelles données vers le type recommandé mais ne le fera pas si cela s'avère impossible.

Il existe plusieurs types d'affinité dans SQLite :

- TEXT : enregistre la donnée comme une chaîne de caractères, sans limite de taille. Si un nombre est entré dans une colonne de ce type, il sera automatiquement converti en une chaîne de caractères ;
- NUMERIC : tente d'enregistrer la donnée comme un entier ou comme un réel, mais si cela s'avère impossible, la donnée sera enregistrée comme une chaîne de caractères ;
- INTEGER : enregistre la donnée comme un entier si celle-ci peut être encodée sans perte, mais peut utiliser les types REAL ou TEXT si ça ne peut être fait ;
- REAL : enregistre la donnée comme un réel, même s'il s'agit d'un entier. Si la valeur est trop grande, la donnée sera convertie en chaîne de caractères ;
- NONE : la donnée est enregistrée telle quelle, sans conversion.

Ainsi, chaque type d'affinité peut accepter n'importe quel type de donnée, la seule exception est le type particulier INTEGER PRIMARY KEY, lorsqu'il est appliqué sur une seule colonne, car il ne s'agit pas d'un type habituel mais d'un alias à la colonne interne au moteur ROWID qui correspond à l'adresse de l'enregistrement, unique à travers la table.

2.4. Contraintes

SQLite gère les contraintes sur une ou plusieurs colonnes. Les contraintes NOT NULL, CHECK, DEFAULT et COLLATE sont déclarées sur la colonne alors que les contraintes PRIMARY KEY, UNIQUE, CHECK et FOREIGN KEY peuvent être déclarées sur une ou plusieurs colonnes.

- La contrainte UNIQUE crée automatiquement un index sur la ou les colonnes sur lesquelles

elle est appliquée.

- PRIMARY KEY : La contrainte de clé primaire va créer une contrainte UNIQUE sur la ou les colonnes concernées.
- ROWID et AUTOINCREMENT : Chaque ligne d'une table est identifiée par un entier signé de 64 bits appelé ROWID. Lorsqu'une table est déclarée avec une et une seule colonne INTEGER PRIMARY KEY, cette colonne devient un alias du ROWID. L'utilisation d'un alias à l'identifiant ROWID permet d'augmenter la vitesse des recherches, celles-ci pouvant être jusqu'à deux fois plus rapides qu'avec une clé primaire normale associée à son index d'unicité.
- Il est possible d'utiliser le mot clé AUTOINCREMENT. Ce dernier modifie légèrement l'algorithme : une fois la limite d'un entier atteinte (2^{63} -1), il ne sera plus possible d'insérer un nouvel enregistrement.
- FOREIGN KEY : toute colonne référencée par une clé étrangère doit être déclarée comme UNIQUE (PRIMARY KEY crée une clé d'unicité).

3. Installer SQLite

Sous M$-Windows, télécharger les binaires windows de la dernière version stable sur le site officiel de SQLite.

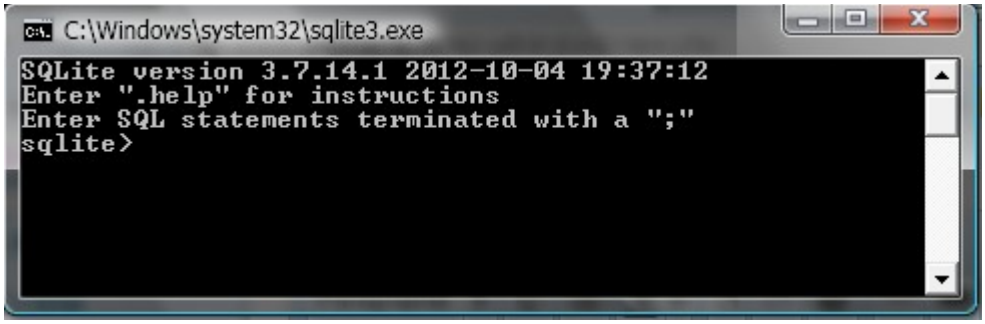
Puis, décompresser les 3 fichiers dans un répertoire temporaire.

Et copier le contenu des 3 répertoires décompressés dans le répertoire \windows\System32 (ou dans un autre, pourvu que celui-ci soit déclaré dans votre environnement comme chemin %path%).

3.1. Commandes en ligne

Une fois SQLite installé, lancez la commande sqilte3 en console ou dans la fenêtre rechercher du menu windows.

Un écran comme celui ci-contre s'affiche si tout est correctement installé.



Ce shell permet d'interagir avec SQLite, de créer des bases, des tables… le shell acceptera des commandes SQL ou des commandes SQLite (.help -> affiche une aide des commande SQLite / .exit quitte l'interpréteur).

Par exemple, pour créer une nouvelle base de données SQLite intitulée "test" avec une seule table nommée "ma_table", il suffit d'entrer :

```
$ sqlite3 test
SQLite version 3.8.5 2014-05-29 12:36:14
Enter ".help" for usage hints.
sqlite> create table ma_table(one varchar(10), two smallint);
sqlite> insert into ma_table values('bonjour !', 10);
```

```
sqlite> insert into ma_table values('salut', 20);
sqlite> select * from ma_table;
bonjour !|10
salut|20
sqlite>
```

Pour connaître toutes les commandes en ligne, connectez-vous sur le site officiel.

3.2. Interface graphique

Une autre façon d'interagir avec une base de donnée SQLite est d'installer le greffon SQLite Manager pour le navigateur FireFox. Il s'agit d'une interface graphique complète qui permet entre autre :

- la gestion de plusieurs base de données (Création, accès ou téléversement de base)
- la gestion des bases de données attachées
- la création, modification et suppression de tables et indexes.
- l'insertion, modification, suppression d'enregistrement dans ces tables
- la gestion des Vues, possibilités de créer une vue a partir d'une requête SELECT
- ...

4. Les API SQLite

Il existe de nombreuses fonctions pour SQLite. Cependant, seules 3 fonctions sont absolument nécessaires pour des opérations courantes et sont détaillées ci-dessous. Pour avoir la liste exhaustive de toutes les autres fonctions, nous vous invitons à vous rendre sur le site officiel.

- `sqlite3_open(const char *filename, sqlite3 **ppDb)`

  Cette fonction ouvre une connexion à un fichier de base de données SQLite et retourne un objet de connexion de base de données pour être utilisé par d'autres fonctions SQLite.

  Si l'argument filename est NULL, sqlite3_open () permettra de créer une base de données dans la RAM en mémoire qui ne dure que pendant la durée de la session.

  Si filename est pas NULL, sqlite3_open () tente d'ouvrir le fichier de base de données en utilisant sa valeur. Si aucun fichier de ce nom existe, sqlite3_open () va ouvrir un nouveau fichier de base de données de ce nom.

- `sqlite3_exec(sqlite3*, const char *sql, sqlite_callback, void *data, char **errmsg)`

  Cette fonction fournit un moyen facile et rapide pour exécuter des commandes SQL fournies par l'argument SQL qui peut être constitué de plus d'une commande SQL.

  L'argument sqlite3 est un objet de base de données, sqlite_callback est une fonction de rappel qui prend data en 1er argument et errmsg sera le message d'erreur de retour en cas d'erreur rencontrée par la fonction.

- `sqlite3_close(sqlite3*)`

  Cette fonction ferme une connexion de base de données précédemment ouvert par un appel à sqlite3_open ( ) . Toutes les requêtes associées à la connexion devraient être finalisées avant la fermeture de la connexion.

Si des requêtes n'ont pas été finalisées, sqlite3_close ( ) retournera SQLITE_BUSY avec le message d'erreur « Impossible de fermer en raison de déclarations non finalisés ».

Remarque : SQLite appelle la fonction sqlite_callback pour chaque enregistrement traité dans chaque instruction SELECT exécutée dans l'argument SQL .

La déclaration de sqlite_callback est la suivante :

```
typedef int (*sqlite3_callback)(
      void*,    /* Data provided in the 4th argument of sqlite3_exec() */
      int,      /* The number of columns in row */
      char**,   /* An array of strings representing fields in the row */
      char**    /* An array of strings representing column names */
      );
```

3.1. Se connecter à une base de données

La première opération consiste à se connecter à une base de données existante, créer en ligne de commande ou par l'interface SQLite Manager.

```
#include <stdio.h>
#include <sqlite3.h>

int main(int argc, char* argv[])
{
      sqlite3 *db;

      int rc = sqlite3_open("test.db", &db);

      if ( rc ) {
         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
         return 1;
         }
      else
         fprintf(stderr, "Opened database successfully\n");

      sqlite3_close(db);

      return 0;
}
```

3.2. Créer une table

Une fois la base de données créée, il faut lui rajouter une ou plusieurs tables.

```
#include <stdio.h>
#include <stdlib.h>
#include <sqlite3.h>

static int callback(void *NotUsed, int argc, char **argv, char **azColName)
{
   int i;
   for (i=0; i < argc; i++)
      printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");

   printf("\n");
```

```
   return 0;
}

int main(int argc, char* argv[])
{
      sqlite3 *db;

      int rc = sqlite3_open("test.db", &db);

      if ( rc ) {
         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
         return 1;
         }
      else
         fprintf(stderr, "Opened database successfully\n");

      /* Create SQL statement */
      const char *sql = "CREATE TABLE COMPANY("  \
         "ID INT PRIMARY KEY     NOT NULL," \
         "NAME           TEXT    NOT NULL," \
         "AGE            INT     NOT NULL," \
         "ADDRESS        CHAR(50)," \
         "SALARY         REAL );";

      /* Execute SQL statement */
      char *errmsg = "";
      if ( sqlite3_exec(db, sql, callback, 0, &errmsg) != SQLITE_OK ) {
            fprintf(stderr, "SQL error: %s\n", errmsg);
            sqlite3_free(errmsg);
            }
      else
            fprintf(stdout, "Operation done successfully\n");

   sqlite3_close(db);

   return 0;
}
```

3.3. Insertion de valeurs

Pour insérer des valeurs dans une table existante, le code précédent reste valable. Il suffit juste de modifier la requête SQL comme indiqué ci-dessous :

```
      /* Create SQL statement */
      const char *sql = "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) "  \
         "VALUES (1, 'Paul', 32, 'California', 20000.00 ); " \
         "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) "  \
         "VALUES (2, 'Allen', 25, 'Texas', 15000.00 ); "     \
         "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)" \
         "VALUES (3, 'Teddy', 23, 'Norway', 20000.00 );" \
         "INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)" \
         "VALUES (4, 'Mark', 25, 'Rich-Mond', 65000.00 );";
```

3.4. Opération SELECT

Le code ci-dessous montre comment récupérer et afficher les enregistrements de la table

COMPANY créée dans l'exemple précédent.

```
#include <stdio.h>
#include <stdlib.h>
#include <sqlite3.h>

static int callback(void *NotUsed, int argc, char **argv, char **azColName)
{
      fprintf(stderr, "%s: ", (const char*)data);
      int i;
      for (i=0; i < argc; i++)
            printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL");

      printf("\n");

      return 0;
}

int main(int argc, char* argv[])
{
      sqlite3 *db;

      int rc = sqlite3_open("test.db", &db);

      if ( rc ) {
         fprintf(stderr, "Can't open database: %s\n", sqlite3_errmsg(db));
         return 1;
         }
      else
         fprintf(stderr, "Opened database successfully\n");

      /* Create SQL statement */
      const char *sql = "SELECT * from COMPANY";

      /* Execute SQL statement */
      const char* data = "Callback function called";
      char *errmsg = "";
      if ( sqlite3_exec(db, sql, callback, (void*)data, &errmsg) != SQLITE_OK ) {
            fprintf(stderr, "SQL error: %s\n", errmsg);
            sqlite3_free(errmsg);
            }
      else
            fprintf(stdout, "Operation done successfully\n");

   sqlite3_close(db);

   return 0;
}
```

Après exécution, le programme devrait afficher ceci :

```
Opened database successfully
Callback function called: ID = 1
NAME = Paul
AGE = 32
ADDRESS = California
SALARY = 20000.0
```

```
Callback function called: ID = 2
NAME = Allen
AGE = 25
ADDRESS = Texas
SALARY = 15000.0

Callback function called: ID = 3
NAME = Teddy
AGE = 23
ADDRESS = Norway
SALARY = 20000.0

Callback function called: ID = 4
NAME = Mark
AGE = 25
ADDRESS = Rich-Mond
SALARY = 65000.0

Operation done successfully
```

3.5. Opération UPDATE

Le code C ci-dessous indique comment nous utiliser l'instruction UPDATE pour mettre à jour un enregistrement, puis récupérer et afficher les enregistrements mis à jour dans la table COMPANY.

Le corps du programme est identique au précédent. Seule la requête SQL change.

```
/* Create merged SQL statement */
const char *sql = "UPDATE COMPANY set SALARY = 25000.00 where ID=1; " \
         "SELECT * from COMPANY";
```

3.6. Opération DELETE

La requête SQL devient :

```
/* Create merged SQL statement */
const char *sql = "DELETE from COMPANY where ID=2;";
```

3.7. Utiliser un champ AUTOINCREMENT

Si vous déclarez une colonne d'une table pour être INTEGER PRIMARY KEY, alors à chaque fois que vous insérez une valeur NULL dans la colonne de la table, la valeur NULL est automatiquement converti en un entier qui est plus grand que la plus grande valeur de cette colonne sur toutes les autres lignes de la table, ou 1 si la table est vide. Ou, si le plus grand entier clés existantes 9223372036854775807 est en cours d'utilisation alors une valeur de clé utilisé est choisi au hasard. Par exemple, supposons que vous avez une table comme ceci :

```
CREATE TABLE t1(
        a INTEGER PRIMARY KEY,
        b INTEGER
        );
```

Avec cette table, la déclaration :

INSERT INTO VALEURS t1 (NULL, 123);

est logiquement équivalent à dire :

INSERT INTO VALEURS T1 ((SELECT max (a) FROM t1) 1123);

Il y a une fonction nommée sqlite3_last_insert_rowid () qui retournera la clé entier pour l'opération d'insertion la plus récente.

3.8. Les codes de retour SQLite

Beaucoup de fonctions dans le langage C d'interface SQLite renvoi un code numérique indiquant un succès ou un échec, et dans le cas d'un échec, fournissant une certaine idée de la cause de l'échec.

On distingue les codes de résultat et les codes d'erreur.

Les "codes d'erreur" sont un sous-ensemble des "codes de résultat" qui indiquent que quelque chose a mal tourné. Il y a seulement quelques codes de résultat sans erreur : SQLITE_OK , SQLITE_ROW et SQLITE_DONE. Le terme "code d'erreur" désigne tout autre code de résultat autre que ces trois là.

- (0) SQLITE_OK : indique que l'opération a réussi et qu'il n'y avait pas d'erreurs.
- (100) SQLITE_ROW : retourné par sqlite3_step( ) indique qu'un autre rangée de sortie est disponible.
- (101) SQLITE_DONE : indique qu'une opération est terminée. Le code de résultat SQLITE_DONE est le plus souvent considérée comme une valeur de retour de sqlite3_step () indiquant que l'instruction SQL a été exécutée jusqu'à la fin.

Pour connaître tous les codes de résultat de l'API SQLite, reportez-vous à la documentation du site officiel.