eBook Gratuit

APPRENEZ sqlite

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#sqlite

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: sqlite

It is an unofficial and free sqlite ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official sqlite.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec sqlite

Versions

Version	Changements majeurs	Date de sortie
3.0		2004-06-18
3.7.11	SELECT max (x), y	2012-03-20
3.8.3	CTE	2014-02-11

Examples

Installation

SQLite est une bibliothèque C qui est généralement compilée directement dans l'application en téléchargeant le code source de la dernière version et en ajoutant le fichier `sqlite3.c` au projet.

De nombreux langages de script (par exemple, Perl , Python , Ruby , etc.) et les frameworks (par exemple, Android) prennent en charge SQLite; Cela se fait avec une copie intégrée de la bibliothèque SQLite, qui n'a pas besoin d'être installée séparément.

Pour tester SQL, il peut être utile d'utiliser le shell de ligne de commande (`sqlite3` ou `sqlite3.exe`). Il est déjà livré avec la plupart des distributions Linux; Sous Windows, téléchargez les fichiers binaires précompilés dans le package `sqlite-tools` et extrayez-les quelque part.

Documentation

SQLite possède déjà une documentation complète, qui ne doit pas être dupliquée ici.

Lire Démarrer avec sqlite en ligne: https://riptutorial.com/fr/sqlite/topic/1753/demarrer-avec-sqlite

Chapitre 2: Commandes par points de ligne de commande

Introduction

Le `sqlite3` de ligne de commande `sqlite3` implémente un ensemble supplémentaire de commandes (qui ne sont pas disponibles dans les programmes utilisant la bibliothèque SQLite). Documentation officielle: Commandes spéciales vers `sqlite3`

Examples

Exportation et importation d'une table en tant que script SQL

L'exportation d'une base de données est un processus simple en deux étapes:

```
sqlite> .output mydatabase_dump.sql
sqlite> .dump
```

L'exportation d'une table est assez similaire:

```
sqlite> .output mytable_dump.sql
sqlite> .dump mytable
```

Le fichier de sortie doit être défini avec `.output` avant d'utiliser `.dump` ; sinon, le texte est simplement sorti à l'écran.

L'import est encore plus simple:

```
sqlite> .read mytable_dump.sql
```

Lire Commandes par points de ligne de commande en ligne:
https://riptutorial.com/fr/sqlite/topic/3789/commandes-par-points-de-ligne-de-commande

Chapitre 3: Déclarations PRAGMA

Remarques

La documentation SQLite contient une référence de toutes les instructions PRAGMA .

Examples

PRAGMA avec des effets permanents

La plupart des instructions PRAGMA n'affectent que la connexion à la base de données en cours, ce qui signifie qu'elles doivent être réappliquées chaque fois que la base de données a été ouverte.

Cependant, les PRAGMA suivants écrivent dans le fichier de base de données et peuvent être exécutés à tout moment (mais dans certains cas, pas dans une transaction):

- ID d'application
- journal_mode lors de l'activation ou de la désactivation du mode WAL
- schéma_version
- version_utilisateur
- wal_checkpoint

Les paramètres PRAGMA suivants définissent les propriétés du fichier de base de données qui ne peuvent pas être modifiés après la création. Ils doivent donc être exécutés avant la première écriture sur la base de données:

- auto_vacuum (peut aussi être changé avant VACUUM)
- codage
- legacy_file_format
- page_size (peut aussi être changé avant VACUUM)

Par exemple:

```
-- open a connection to a not-yet-existing DB file
PRAGMA page_size = 4096;
PRAGMA auto_vacuum = INCREMENTAL;
CREATE TABLE t(x);              -- database is created here, with the above settings
```

Lire Déclarations PRAGMA en ligne: https://riptutorial.com/fr/sqlite/topic/5223/declarations-pragma

Chapitre 4: sqlite3_stmt: instruction préparée (API C)

Remarques

documentation officielle: objet déclaration préparé

Examples

Exécuter une déclaration

Une instruction est construite avec une fonction telle que sqlite3_prepare_v2 () .

Un objet d'instruction préparé *doit* être nettoyé avec sqlite3_finalize () . Ne l'oubliez pas en cas d'erreur.

Si des paramètres sont utilisés, définissez leurs valeurs avec les fonctions sqlite3_bind_xxx () .

L'exécution réelle se produit lorsque sqlite3_step () est appelée.

```
const char *sql = "INSERT INTO MyTable(ID, Name) VALUES (?, ?)";
sqlite3_stmt *stmt;
int err;

err = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
if (err != SQLITE_OK) {
    printf("prepare failed: %s\n", sqlite3_errmsg(db));
    return /* failure */;
}

sqlite3_bind_int (stmt, 1, 42);                          /* ID */
sqlite3_bind_text(stmt, 2, "Bob", -1, SQLITE_TRANSIENT); /* name */

err = sqlite3_step(stmt);
if (err != SQLITE_DONE) {
    printf("execution failed: %s\n", sqlite3_errmsg(db));
    sqlite3_finalize(stmt);
    return /* failure */;
}

sqlite3_finalize(stmt);
return /* success */;
```

Lecture des données d'un curseur

Une requête SELECT est exécutée comme toute autre instruction. Pour lire les données renvoyées, appelez sqlite3_step () dans une boucle. Il retourne:

- SQLITE_ROW: si les données de la prochaine ligne sont disponibles ou

- SQLITE_DONE: s'il n'y a plus de lignes ou
- tout code d'erreur.

Si une requête ne renvoie aucune ligne, la toute première étape renvoie SQLITE_DONE.

Pour lire les données de la ligne en cours, appelez les fonctions sqlite3_column_xxx () :

```
const char *sql = "SELECT ID, Name FROM MyTable";
sqlite3_stmt *stmt;
int err;

err = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
if (err != SQLITE_OK) {
    printf("prepare failed: %s\n", sqlite3_errmsg(db));
    return /* failure */;
}

for (;;) {
    err = sqlite3_step(stmt);
    if (err != SQLITE_ROW)
        break;

    int         id   = sqlite3_column_int (stmt, 0);
    const char *name = sqlite3_column_text(stmt, 1);
    if (name == NULL)
        name = "(NULL)";
    printf("ID: %d, Name: %s\n", id, name);
}

if (err != SQLITE_DONE) {
    printf("execution failed: %s\n", sqlite3_errmsg(db));
    sqlite3_finalize(stmt);
    return /* failure */;
}

sqlite3_finalize(stmt);
return /* success */;
```

Exécuter une déclaration préparée plusieurs fois

Une fois qu'une instruction a été exécutée , un appel à sqlite3_reset () le ramène à son état d'origine pour qu'il puisse être ré-exécuté.

En général, alors que l'instruction elle-même reste la même, les paramètres sont modifiés:

```
const char *sql = "INSERT INTO MyTable(ID, Name) VALUES (?, ?)";
sqlite3_stmt *stmt;
int err;

err = sqlite3_prepare_v2(db, sql, -1, &stmt, NULL);
if (err != SQLITE_OK) {
    printf("prepare failed: %s\n", sqlite3_errmsg(db));
    return /* failure */;
}

for (...) {
    sqlite3_bind_int (stmt, 1, ...);   /* ID */
```

```
    sqlite3_bind_text(stmt, 2, ...);   /* name */

    err = sqlite3_step(stmt);
    if (err != SQLITE_DONE) {
        printf("execution failed: %s\n", sqlite3_errmsg(db));
        sqlite3_finalize(stmt);
        return /* failure */;
    }

    sqlite3_reset(stmt);
}

sqlite3_finalize(stmt);
return /* success */;
```

Lire sqlite3_stmt: instruction préparée (API C) en ligne:
https://riptutorial.com/fr/sqlite/topic/5456/sqlite3-stmt--instruction-preparee--api-c-

Chapitre 5: Types de données

Remarques

documentation officielle: Types de données dans SQLite Version 3

Examples

Fonction TYPEOF

```
sqlite> SELECT TYPEOF(NULL);
null
sqlite> SELECT TYPEOF(42);
integer
sqlite> SELECT TYPEOF(3.141592653589793);
real
sqlite> SELECT TYPEOF('Hello, world!');
text
sqlite> SELECT TYPEOF(X'0123456789ABCDEF');
blob
```

En utilisant des booléens

Pour les booléens, SQLite utilise les entiers `0` et `1` :

```
sqlite> SELECT 2 + 2 = 4;
1
sqlite> SELECT 'a' = 'b';
0
sqlite> SELECT typeof('a' = 'b');
integer
```

```
> CREATE TABLE Users ( Name, IsAdmin );
> INSERT INTO Users VALUES ('root', 1);
> INSERT INTO Users VALUES ('john', 0);
> SELECT Name FROM Users WHERE IsAdmin;
root
```

Application des types de colonnes

SQLite utilise le typage dynamique et ignore les types de colonnes déclarés:

```
> CREATE TABLE Test (
      Col1 INTEGER,
      Col2 VARCHAR(2),       -- length is ignored, too
      Col3 BLOB,
      Col4,                  -- no type required
      Col5 FLUFFY BUNNIES    -- use whatever you want
  );
> INSERT INTO Test VALUES (1, 1, 1, 1, 1);
```

```
> INSERT INTO Test VALUES ('xxx', 'xxx', 'xxx', 'xxx', 'xxx');
> SELECT * FROM Test;
1   1   1   1   1
xxx xxx xxx xxx xxx
```

(Cependant, les types de colonnes déclarés sont utilisés pour l' affinité de type .)

Pour appliquer des types, vous devez ajouter une contrainte avec la fonction typeof () :

```
CREATE TABLE Tab (
    Col1 TEXT   CHECK (typeof(Col1) = 'text' AND length(Col1) <= 10),
    [...]
);
```

(Si une telle colonne doit être NULL, vous devez explicitement autoriser `'null'` .)

Types date / heure

SQLite n'a pas de type de données séparé pour les valeurs de date et d'heure.

Chaînes ISO8601

Les mots-clés intégrés `CURRENT_DATE` , `CURRENT_TIME` et `CURRENT_TIMESTAMP` renvoient des chaînes au format ISO8601:

```
> SELECT CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP;
CURRENT_DATE  CURRENT_TIME  CURRENT_TIMESTAMP
------------  ------------  -------------------
2016-07-08    12:34:56      2016-07-08 12:34:56
```

Ces valeurs sont également comprises par toutes les fonctions de date / heure intégrées :

```
> SELECT strftime('%Y', '2016-07-08');
2016
```

Nombre de jours juliens

Les fonctions de date / heure intégrées interprètent les nombres en jours juliens :

```
> SELECT datetime(2457578.02425926);
2016-07-08 12:34:56
```

La fonction `julianday()` convertit toute valeur de date / heure prise en charge en un nombre de jours juliens:

```
> SELECT julianday('2016-07-08 12:34:56');
2457578.02425926
```

Horodatage Unix

Les fonctions de date / heure intégrées peuvent interpréter les nombres comme des horodatages Unix avec le modificateur `unixepoch` :

```
> SELECT datetime(0, 'unixepoch');
1970-01-01 00:00:00
```

La fonction `strftime()` peut convertir toute valeur de date / heure prise en charge en un horodatage Unix:

```
> SELECT strftime('%s', '2016-07-08 12:34:56');
1467981296
```

formats non supportés

Il serait possible de stocker les valeurs de date / heure dans tout autre format de la base de données, mais les fonctions de date / heure intégrées ne les analyseront pas et renverront NULL:

```
> SELECT time('1:30:00');   -- not two digits

> SELECT datetime('8 Jul 2016');
□
```

Lire Types de données en ligne: https://riptutorial.com/fr/sqlite/topic/5252/types-de-donnees

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec sqlite	CL., Community, e4c5, H. Pauwelyn
2	Commandes par points de ligne de commande	CL., e4c5, James Toomey, Lasse Vågsæther Karlsen, ravenspoint, Thinkeye
3	Déclarations PRAGMA	CL., springy76
4	sqlite3_stmt: instruction préparée (API C)	CL.
5	Types de données	CL.