EBook Gratuito

APPRENDIMENTO sqlite

Free unaffiliated eBook created from
Stack Overflow contributors.

#sqlite

Sommario

Di

Capitolo 1: Iniziare con sqlite

Versioni

Versione	Principali cambiamenti	Data di rilascio
3.0		2004-06-18
3.7.11	SELEZIONA max (x), y	2012-03-20
3.8.3	CTE	2014/02/11

Examples

Installazione

SQLite è una libreria C che viene in genere compilata direttamente nell'applicazione scaricando il codice sorgente dell'ultima versione e aggiungendo il file `sqlite3.c` al progetto.

Molti linguaggi di script (es. Perl , Python , Ruby , ecc.) E framework (ad es. Android) supportano SQLite; questo viene fatto con una copia integrata della libreria SQLite, che non ha bisogno di essere installata separatamente.

Per il test di SQL, potrebbe essere utile utilizzare la shell della riga di comando (`sqlite3` o `sqlite3.exe`). È già fornito con la maggior parte delle distribuzioni Linux; su Windows, scarica i binari precompilati nel pacchetto `sqlite-tools` ed estraili da qualche parte.

Documentazione

SQLite ha già una vasta documentazione , che non dovrebbe essere duplicata qui.

Leggi Iniziare con sqlite online: https://riptutorial.com/it/sqlite/topic/1753/iniziare-con-sqlite

Capitolo 2: Comandi punto linea di comando

introduzione

La `sqlite3` riga di comando `sqlite3` implementa un ulteriore set di comandi (che non sono disponibili nei programmi che usano la libreria SQLite). Documentazione ufficiale: comandi speciali per `sqlite3`

Examples

Esportazione e importazione di una tabella come script SQL

Esportare un database è un semplice processo in due fasi:

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

Esportare una tabella è abbastanza simile:

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

Il file di output deve essere definito con `.output` prima dell'uso `.dump` ; in caso contrario, il testo viene semplicemente visualizzato sullo schermo.

L'importazione è ancora più semplice:

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

Leggi Comandi punto linea di comando online: https://riptutorial.com/it/sqlite/topic/3789/comandi-punto-linea-di-comando

Capitolo 3: Dichiarazioni PRAGMA

Osservazioni

La documentazione SQLite ha un riferimento a tutte le dichiarazioni PRAGMA .

Examples

PRAGMA con effetti permanenti

La maggior parte delle istruzioni PRAGMA influisce solo sulla connessione al database corrente, il che significa che devono essere riapplicate ogni volta che il database è stato aperto.

Tuttavia, i seguenti PRAGMA scrivono nel file di database e possono essere eseguiti in qualsiasi momento (ma in alcuni casi, non all'interno di una transazione):

- ID applicazione
- journal_mode quando si abilita o disabilita la modalità WAL
- SCHEMA_VERSION
- user_version
- wal_checkpoint

Le seguenti impostazioni di PRAGMA impostano le proprietà del file di database che non possono essere modificate dopo la creazione, quindi devono essere eseguite prima della prima scrittura effettiva nel database:

- auto_vacuum (può anche essere cambiato prima di VACUUM)
- codifica
- legacy_file_format
- page_size (può anche essere modificato prima di VACUUM)

Per esempio:

```
-- 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
```

Leggi Dichiarazioni PRAGMA online: https://riptutorial.com/it/sqlite/topic/5223/dichiarazioni-pragma

Capitolo 4: sqlite3_stmt: istruzione preparata (API C)

Osservazioni

documentazione ufficiale: oggetto Statement preparato

Examples

Esecuzione di una dichiarazione

Una dichiarazione è costruita con una funzione come sqlite3_prepare_v2 () .

Un oggetto statement preparato *deve* essere ripulito con sqlite3_finalize () . Non dimenticarlo in caso di errore.

Se si utilizzano i parametri , impostare i loro valori con le funzioni sqlite3_bind_xxx () .

L'esecuzione effettiva avviene quando viene chiamato sqlite3_step () .

```
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 */;
```

Lettura dei dati da un cursore

Una query SELECT viene eseguita come qualsiasi altra istruzione. Per leggere i dati restituiti, chiama sqlite3_step () in un ciclo. Restituisce:

- SQLITE_ROW: se i dati per la riga successiva sono disponibili, o

- SQLITE_DONE: se non ci sono più righe, o
- qualsiasi codice di errore.

Se una query non restituisce alcuna riga, il primo passaggio restituisce SQLITE_DONE.

Per leggere i dati dalla riga corrente, chiama le funzioni 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 */;
```

Esecuzione di una dichiarazione preparata più volte

Dopo l' esecuzione di una dichiarazione, una chiamata a sqlite3_reset () lo riporta nello stato originale in modo che possa essere rieseguito.

In genere, mentre l'istruzione rimane invariata, i parametri vengono modificati:

```
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 */;
```

Leggi sqlite3_stmt: istruzione preparata (API C) online:
https://riptutorial.com/it/sqlite/topic/5456/sqlite3-stmt--istruzione-preparata--api-c-

Capitolo 5: Tipi di dati

Osservazioni

documentazione ufficiale: Datatypes In SQLite Version 3

Examples

Funzione 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
```

Usando i booleani

Per i booleani, SQLite utilizza gli interi `0` e `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
```

Imporre i tipi di colonna

SQLite utilizza la digitazione dinamica e ignora i tipi di colonna dichiarati:

```
> 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
```

(Tuttavia, i tipi di colonna dichiarati vengono utilizzati per l' affinità del tipo .)

Per applicare i tipi, devi aggiungere un vincolo con la funzione typeof () :

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

(Se una colonna di questo tipo deve essere NULLable, devi autorizzare esplicitamente `'null'` .)

Tipi di data / ora

SQLite non ha un tipo di dati separato per i valori di data o ora.

Stringhe ISO8601

Le parole chiave incorporate `CURRENT_DATE` , `CURRENT_TIME` e `CURRENT_TIMESTAMP` restituiscono le stringhe nel formato 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
```

Tali valori sono compresi anche da tutte le funzioni di data / ora incorporate :

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

Numeri del giorno giuliano

Le funzioni incorporate di data / ora interpretano i numeri come giorni di Julian :

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

La funzione `julianday()` converte qualsiasi valore data / ora supportato in un numero del giorno giuliano:

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

Timestamp Unix

Le funzioni di data / ora incorporate possono interpretare i numeri come timestamp Unix con il modificatore `unixepoch` :

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

La `strftime()` può convertire qualsiasi valore data / ora supportato in un timestamp Unix:

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

formati non supportati

Sarebbe possibile memorizzare i valori di data / ora in qualsiasi altro formato nel database, ma le funzioni incorporate di data / ora non li analizzeranno e restituiranno NULL:

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

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

Leggi Tipi di dati online: https://riptutorial.com/it/sqlite/topic/5252/tipi-di-dati

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con sqlite	CL., Community, e4c5, H. Pauwelyn
2	Comandi punto linea di comando	CL., e4c5, James Toomey, Lasse Vågsæther Karlsen, ravenspoint, Thinkeye
3	Dichiarazioni PRAGMA	CL., springy76
4	sqlite3_stmt: istruzione preparata (API C)	CL.
5	Tipi di dati	CL.