SQLite

di Roberta Molinari

SQLite

- È una **libreria software** scritta in linguaggio **C** che implementa un RDBMS per SQL garantendo transazioni di tipo ACID. Il suo creatore, D. Richard Hipp, l'ha rilasciata di **pubblico dominio**.
- Permette di creare un DB (tabelle, query, form, report) incorporato in un **unico file** (estensione .db).
- SQLite non è un sw utilizzabile di per sé, ma deve essere **incorporato** (embedded) all'interno di un altro programma scritto in altri linguaggi (C/C++, ...) o essere utilizzato con un SW grafico ad hoc per la sua gestione (**DB Browser**)
- È integrato nella versione 5 di PHP ed è standard per Android

SQLite
Caratteristiche

- è **compatto** (meno di 500KB)
- è molto **veloce**
- il **codice sorgente** è liberamente disponibile
- è in grado di **interpretare stringhe SQL**
- l'API è **semplice** da utilizzare
- ha transazioni atomiche, consistenti, isolate e durabili (**ACID**), anche in caso di crash di sistema o blackout
- è **portabile**
- può essere usato con vari linguaggi di programmazione
- è incorporato in molte applicazioni, come i browser web
- ha una **gestione flessibile dei tipi di dati**: ogni campo può contenere qualsiasi tipo di dato

SQLite
Caratteristiche

- include un programma di utilità a **riga di comando** per accedere al database anche manualmente o tramite scripting;
- supporta **DB molto grandi**(< 2TB);
- un db consiste di un **unico file**, il cui formato interno è indipendente dalla piattaforma e dal relativo ordine dei byte
- SQLite è "stand-alone" or "self-contained" nel senso che ha pochissime dipendenze. Funziona su qualsiasi sistema operativo, anche SO embedded ridotti all'osso
- normalmente non richiede alcun lavoro di amministrazione, ma supporta il comando "SQL VACUUM" nel caso si renda necessario compattare esplicitamente i dati del database.

4

SQLite
Quando si usa?

- Si ha bisogno di un database leggero, veloce e flessibile, che possa funzionare senza server e senza installazione.
- Si ha bisogno di un database locale, che possa memorizzare i dati sul dispositivo e renderli disponibili anche offline.
- Si ha bisogno di un database portatile, che possa essere trasferito facilmente tra dispositivi o applicazioni tramite una chiavetta USB o una connessione wireless.

SQLite
Limiti

- non offre le **stored procedure**
- È **debolmente tipizzato** e i tipi sono assegnati **dinamicamente**
- non prevede la gestione dei **permessi d'accesso** (GRANT)
- non ha una vera gestione della **concorrenza** (blocca l'intero file di database durante la scrittura)
- per garantire la coerenza del file del database sono usati i **lock** del *file system*, e quindi vi possono essere problemi qualora quest'ultimo non li implementi correttamente (con *file system* di rete come NFS)
- non offre alcuna **cache** per le query

6

SQLite
Limiti

- non supporta alcuni importanti costrutti SQL quali **RIGHT JOIN** e **FULL OUTER JOIN** (solo il **LEFT JOIN**)
- non ha protocolli di rete; è possibile utilizzare un db remoto, ma solo tramite *file system* di rete del sistema operativo, con prestazioni difficilmente accettabili;
- non supporta le **sottoquery variabili**.
- non supporta la scrittura diretta nelle **viste** (occorre usare trigger "INSTEAD OF")
- non consente di modificare, cancellare o rinominare le colonne di una tabella: il comando **ALTER TABLE** è infatti limitato alla modifica del nome della tabella e all'aggiunta di colonne in coda alla stessa. \

7

SQLite
Limiti

- non supporta trigger di tipo "FOR EACH STATEMENT" (solo "FOR EACH ROW", eventualmente combinato con "INSTEAD OF");
- il supporto ai **trigger ricorsivi** ed ai vincoli sulle chiavi esterne, introdotto rispettivamente nelle versioni 3.6.18 e 3.6.19, deve essere attivato dal programmatore; tale comportamento, dovuto alla compatibilità con le versioni precedenti, sarà modificato a partire dalla versione 3.7.

- https://www.sqlitetutorial.net/

8

SQLite
Tipi

- SQLite supporta il concetto di "affinità di tipo" sulle colonne. **L'affinità di tipo di una colonna è il tipo consigliato per i dati** archiviati in quella colonna, non obbligatorio.
- Qualsiasi colonna può memorizzare qualsiasi tipo di dati. È solo che alcune colonne, preferiranno usare una classe di archiviazione rispetto a un'altra. La classe di archiviazione preferita per una colonna è chiamata la sua "affinità".
- SQLite è debolmente tipizzato e i tipi sono decisi dinamicamente
- SQLite crea schemi, che **vincolano il tipo di dati** in ciascuna colonna, ma **non li impongono**: non rifiuterà valori di tipo errato.

SQLite
Tipi

- I valori possono appartenere alle seguenti classi di archiviazione :
  - **NULL**. The value is a NULL value.
  - **INTEGER**. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value. **Boolean** values are stored as integers 0 (false) and 1 (true).
  - **REAL**. The value is a floating point value, stored as an 8-byte IEEE floating point number.
  - **TEXT**. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
  - **BLOB**. The value is a blob of data, stored exactly as it was input.

La funzione **typeof** (X) restituisce una stringa che indica il tipo di dati dell'espressione X: "null", "integer", "real", "text" o "blob"

SQLite
Affinità di tipo

- **NUMBER** può contenere valori che utilizzano tutte e cinque le classi di archiviazione.
  - Se il testo è rispettivamente un intero ben formato o un letterale reale la classe di memorizzazione del testo viene convertita in INTEGER o REAL (in ordine di preferenza).
  - Se il valore TEXT è un valore letterale intero ben formato, troppo grande per essere contenuto in un intero con segno a 64 bit, viene convertito in REAL.
  - Per le conversioni tra le classi di archiviazione TEXT e REAL, vengono conservate solo le prime 15 cifre decimali significative del numero.
  - Se il valore TEXT non è un numero intero ben formato o un valore letterale reale, il valore viene memorizzato come testo (i letterali interi esadecimali non sono considerati ben formati e vengono memorizzati come testo.
  - Non si convertono i valori NULL o BLOB.

SQLite
Affinità di tipo

Una colonna con affinità:

- **TEXT** memorizza tutti i dati utilizzando le classi di archiviazione NULL, TEXT o BLOB. I dati numerici vengono convertiti in testo
- **BLOB** non viene effettuato alcun tentativo di forzare i dati da una classe di archiviazione a un'altra.
- **INTEGER** si comporta come una colonna con affinità NUMBER. Se inserisco 2.5 memorizza 2.5, mentre 2.0 viene salvato come 2
- **REAL** si comporta come una colonna con affinità NUMBER tranne per il fatto che forza i valori interi nella rappresentazione in virgola mobile.

SQLite
Affinità di tipo

L'affinità di una colonna è determinata dal tipo dichiarato nella colonna, secondo le seguenti regole: se il tipo contiene la stringa…

1. "INT" → INTEGER
2. "CHAR", "CLOB" o "TEXT"→ TEXT. Nota che il tipo VARCHAR contiene la stringa "CHAR" e pertanto viene assegnata l'affinità TEXT.
3. "BLOB" o se non viene specificato alcun tipo → BLOB
4. "REAL", "FLOA" o "DOUB", →REAL
5. Altrimenti (NUMERIC, DECIMAL, BOOLEAN, DATE, DATETIME) → NUMBER

L'ordine delle regole per determinare l'affinità di colonna è importante. Una colonna il cui tipo dichiarato è "CHARINT" corrisponderà a entrambe le regole 1 e 2 ma la prima regola ha la precedenza e quindi l'affinità di colonna sarà INTEGER.

SQLite
DML

- Per creare una tabella

```
CREATE TABLE "Responsabili" (
  "CodResp"    INTEGER,
  "Cognome"    TEXT NOT NULL,
  "Nome"       TEXT NOT NULL,
  PRIMARY KEY("CodResp" AUTOINCREMENT)
);
```

SQLite
Check

- È possibile inserire dei **check** con la sintassi seguente

```
CREATE TABLE "SedeA" (
  "CodOperaio" TEXT NOT NULL,
  "Sesso"      TEXT NOT NULL CHECK("Sesso" IN ('M', 'F')),
  "AssuntoIl"  TEXT  NOT  NULL  CHECK("AssuntoIl"  LIKE  '____-__-
  __'),
  "Stipendio"  REAL NOT NULL DEFAULT 0 CHECK("Stipendio" >= 0),
  "CodR"       INTEGER,
  FOREIGN KEY("CodR") REFERENCES "Responsabili"("CodResp"),
  PRIMARY KEY("CodOperaio")
);
```

SQLite
Integrità referenziale

- Quando si definisce una FK si può applicare l'integrità referenziale durante l'eliminazione (ON DELETE) o la modifica (ON UPDATE) della PK associata. Le possibili azioni sono:
  - .**NO ACTION**: when a parent key is modified or deleted from the database, return an error (default)
  - **RESTRICT**: SQLite return an error immediately if a parent key with dependent child keys is deleted or modified.
  - **SET NULL**: when a parent key is deleted or modified, the child key columns of all rows in the child table that mapped to the parent key are set to contain SQL NULL values.
  - **SET DEFAULT**: similar to "SET NULL", but each of the child key columns is set to contain the columns default value.
  - **CASCADE**: A "CASCADE" action propagates the delete or update operation on the parent key to each dependent child key.

SQLite
Tipi data

SQLite non ha una classe di archiviazione per le date. Ma le sue funzioni integrate per data e ora sono in grado di memorizzare date e ore come:

- **TEXT** come stringhe ISO8601 ("AAAA-MM-GG HH:MM:SS.SSS").
- **REAL** come numero di giorni trascorsi dal mezzogiorno di Greenwich del 24 novembre 4714 a.C. secondo il calendario gregoriano.
- **INTEGER** come Unix Time, numero di secondi trascorsi 1970-01-01 00:00:00 UTC.

SQLite
Gestione date

- SQLite non dispone di tipo specifico per date e orari. Però le funzioni di data e ora integrate di SQLite sono in grado di memorizzare date e ore come
  - **TEXT** as ISO8601 strings ("YYYY-MM-DD HH:MM:SS.SSS").
  - **REAL** as Julian day numbers, the number of days since noon in Greenwich on November 24, 4714 B.C. according to the proleptic Gregorian calendar.
  - **INTEGER** as Unix Time, the number of seconds since 1970-01-01 00:00:00 UTC.
- Le parole chiave CURRENT_DATE, CURRENT_TIME e CURRENT_TIMESTAMP restituiscono le stringhe nel formato ISO8601 ovvero come
  - `2016-07-08`
  - `12:34:56 20`
  - `16-07-08 12:34:56`

SQLite
Funzioni per le date

- Le seguenti funzioni possono avere 0 o più modificatori. Restituiscono TEXT

date (TimeString, modif, ...)	Restituisce una data nel formato AAAA-MM-DD.
time (TimeString, modif, ...)	Ora nel formato HH: MM: SS.
datetime (TimeString, modif, ...)	AAAA-MM-GG HH: MM: SS
julianday (TimeString, modif,, ...)	numero di giorni dal 24 novembre 4714 aC alle 12.00
strftime (format, TimeString, modif, ...)	data formattata in base al primo parametro

TimeString accettata	**Formato data**
YYYY-MM-DD	2010-12-30
YYYY-MM-DD HH:MM	2010-12-30 12:10
YYYY-MM-DD HH:MM:SS.SSS	2010-12-30 12:10:04.100
MM-DD-YYYY HH:MM	30-12-2010 12:10
HH:MM	12:10
YYYY-MM-DD**T**HH:MM	2010-12-30 12:10
HH:MM:SS	12:10:01
YYYYMMDD HHMMSS	20101230 121001
now	2020-10-22

SQLite
Funzioni per le date

I modificatori possono essere le seguenti stringhe:

- 'N day/ month/year/hour/minute/second'
- 'start of month'
- 'start of year'
- 'start of day'
- 'weekday N'
- 'unixepoch'
- 'localtime'
- 'utc'

```
SELECT date ('now');
SELECT date ('now', 'start of month', '+1 month', '-2 days');
SELECT datetime (1092941466, 'unixepoch');  //2004-08-19 18:51:06
```

SQLite
Funzioni per le date

- **strftime** (format, TimeString, modif, ...)
- format può essere

%d	Giorno 01-31
%H	Ore 00-23
%j	Giorno dell'anno 001-366
%m	Mese 00-12
%M	Minuti 00-59
%s	Giorni dal 1970-01-01
%S	Secondi 00-59
%w	Giorni della settimana 0-6 (0 is Sunday)
%W	Settimana dell'anno 01-53
%Y	Anno YYYY
%%	% symbol

```
SELECT strftime ('%m-%Y', 'now') → '10-2020'
SELECT * FROM … WHERE strftime ('%Y', natoIl) ='2000'
```

invece di `natoIl LIKE('2000%')`

SQLite
Funzioni per le date

Quali film sono stati proiettati a Natale dell'anno scorso

```
SELECT FILM.titolo
FROM FILM, PROIEZIONI
WHERE FILM.codFilm = PROIEZIONI.codFilm
AND strftime("%m-%d",PROIEZIONI.dataProiezione) = "12-25"
AND strftime("%Y",CURRENT_DATE) - 1 = strftime("%Y", PROIEZIONI.dataProiezione)
```