77.1 Utilizzo generale 2104
77.1.1 Utilizzo di «sqlite3» 2104
77.1.2 Copie di sicurezza 2106
77.1.3 Accesso simultaneo a più basi di dati 2108
77.2 Esempi comuni 2109
77.2.1 Creazione di una relazione 2109
77.2.2 Modifica della relazione 2110
77.2.3 Inserimento dati in una relazione 2110
77.2.4 Eliminazione di una relazione 2111
77.2.5 Interrogazioni semplici 2111
77.2.6 Interrogazioni simultanee di più relazioni 2114
77.2.7 Alias 2116
77.2.8 Viste 2117
77.2.9 Aggiornamento delle tuple 2117
77.2.10 Cancellazione delle tuple 2118
77.2.11 Inserimento in una relazione esistente 2118
77.3 Riferimenti 2119

77.1 Utilizzo generale

«

SQLite[1] è una libreria in grado di fornire le funzionalità di un DBMS relazionale, basato sul linguaggio SQL, utilizzando come basi di dati dei file singoli. In pratica, SQLite gestisce una basi di dati intera in un file singolo, senza alcuna amministrazione esterna; pertanto, le utenze e i privilegi sono definiti dal sistema operativo, in quanto gli accessi sono regolati dai permessi del file che contiene la basi di dati.

In qualità di libreria, SQL consente ai programmi che la utilizzano di avere accesso a una basi di dati anche senza bisogno di collegarsi a un servizio DBMS tradizionale, facilitando così la realizzazione di piccoli progetti.

Di norma, la libreria SQL è accompagnata da un programma per l'esecuzione interattiva di istruzioni SQLite, che consente di accedere alle sue basi di dati in modo generalizzato.

77.1.1 Utilizzo di «sqlite3»

«

Per creare o accedere a una basi di dati di SQL, si può usare il programma '**`sqlite3`**', o '**`sqlite`**' nelle versioni più vecchie, che accompagna normalmente la libreria vera e propria:

```
sqlite3 [opzioni] [file_base_di_dati]
```

Generalmente, alla fine della riga di comando si indica il nome del file base di dati a cui si vuole accedere; se poi il file non dovesse esistere, il fatto di nominarlo implicherebbe comunque la sua creazione (come base di dati vuota). Se il nome di questo file non viene fornito attraverso la riga di comando, occorre usare un'istruzione apposita,

durante il funzionamento di '**sqlite**'. L'esempio seguente mostra l'avvio e la conclusione del programma, allo scopo di accedere alla base di dati contenuta nel file '`mia_prova.db`':

```
$ sqlite3 mia_prova.db[Invio]

  SQLite version …
  Enter ".help" for instructions
sqlite3> .quit[Invio]
```

Come si può intendere, si tratta di un programma che si comporta sostanzialmente come altri programmi frontali simili, usati per accedere in modo diretto ad altri DBMS basati su SQL.

Tabella 77.2. Alcune opzioni per l'avvio di '**sqlite3**'.

Opzione	Descrizione
`-init` ***file***	Consente di indicare un file contenente comandi per '**sqlite3**'; può trattarsi di comandi interni al programma, oppure di istruzioni SQL.
`-column`	Richiede di mostrare i risultati in modo incolonnato.
`-html`	Richiede di generare i risultati delle interrogazioni in forma di tabella HTML.

Oltre all'uso dell'opzione '**-init**', il programma '**sqlite3**' può ricevere un file contenente comandi e istruzioni SQL semplicemente dallo standard input.

Tabella 77.3. Alcuni comandi interni di '**sqlite3**'.

Comando	Descrizione
`.help`	Mostra una guida interna all'uso del programma.
`.exit` `.quit`	Termina il funzionamento del programma.
`.dump` [*relazione*] ...	Consente di «scaricare» il contenuto della base di dati, oppure delle sole relazioni indicate, generando le istruzioni necessarie a ricostruire le informazioni relative.
`.read` *file*	Legge ed esegue i comandi contenuti nel file indicato.
`.mode column`	Fa in modo di visualizzare il risultato delle interrogazioni in una forma incolonnata.
`.mode list`	Fa in modo di visualizzare il risultato delle interrogazioni nel modo usuale di '**sqlite3**'.
`.header on` `.header off`	attiva o disattiva l'intestazione delle colonne.

77.1.2 Copie di sicurezza

«

È evidente che la copia di una base di dati realizzata con SQLite è un'operazione molto semplice, essendo tutto contenuto in un file. Tuttavia, per garantire la trasferibilità in un'architettura differente, è necessario procedere alla generazione di un file di testo contenente le istruzioni SQL con cui poi ricostruire la base di dati. Si ottiene lo scarico in questa forma usando il comando '**.dump**':

```
echo ".dump" | sqlite3 file_base_di_dati > file_sql
```

Per esempio, per generare il file ‘`backup.sql`’ dalla base di dati contenuta nel file ‘`mio.db`’, si potrebbe usare il comando seguente:

```
$ echo ".dump" | sqlite3 mio.db > backup.sql [Invio]
```

Quello che si ottiene nel file ‘`backup.sql`’ potrebbe avere l’aspetto seguente:

```
BEGIN TRANSACTION;
CREATE TABLE Indirizzi (codice integer, cognome char(40), ↩
↪nome char(40), indirizzo varchar(60), telefono varchar(40));
INSERT INTO "Indirizzi" VALUES(1, 'Pallino', 'Pinco', ↩
↪'Via Biglie, 1', '0222,222222');
…
CREATE TABLE presenze (codice integer, giorno date, ↩
↪ingresso time, uscita time);
INSERT INTO "presenze" VALUES(1, '2005-09-17', '11:20:00', '13:30:00');
…
CREATE VIEW ingressi as select indirizzi.cognome, indirizzi.nome, ↩
↪presenze.giorno, presenze.ingresso ↩
↪from indirizzi, presenze ↩
↪where indirizzi.codice = presenze.codice;
COMMIT;
```

Per ricostruire una base di dati da un file del genere, basta fornire il file a ‘**`sqlite3`**’ dallo standard input:

```
sqlite3 file_base_di_dati < file_sql
```

Per esempio, per creare una base di dati nuova nel file ‘`nuova.db`’, a partire dallo stesso file appena creato, si potrebbe usare il comando seguente:

```
$ sqlite3 nuova.db < backup.sql [Invio]
```

77.1.3 Accesso simultaneo a più basi di dati

«

SQLite consente di accedere a più di una basi di dati alla volta. Per fare questo è disponibile un'istruzione SQL che non è standard:

```
ATTACH [DATABASE] file_base_di_dati AS nome_interno
```

In pratica, si indica il file della basi di dati a cui collegarsi, tenendo conto che può essere necessario indicarlo tra virgolette, ma le si deve associare un nome. Nell'esempio seguente, durante il funzionamento di '**sqlite3**' viene collegata la basi di dati contenuta nel file '`seconda.db`', associandola al nome '**seconda**':

```
sqlite> ATTACH 'seconda.db' AS seconda; [Invio]
```

La base di dati già aperta all'atto dell'avvio di '**sqlite3**' acquista automaticamente il nome interno '**main**'. La distinzione delle basi di dati con questi nomi interni, consente di accedere a relazioni che altrimenti avrebbero lo stesso nome: per indicare la relazione '**indirizzi**' della base di dati '**seconda**', si indica '**seconda.indirizzi**'; per indicare la relazione con lo stesso nome, contenuta nella base di dati aperta all'avvio del programma, si indica '**main.indirizzi**'. Comunque, si osservi che non è necessario indicare dei nomi completi della base di dati se non ci sono ambiguità tra le relazioni.

Eventualmente, per chiudere il collegamento con una base di dati c'è un'altra istruzione:

```
DETACH [DATABASE] nome_interno
```

77.2 Esempi comuni

Nelle sezioni seguenti vengono mostrati alcuni esempi comuni di utilizzo del linguaggio SQL, limitato alle possibilità di SQLite. La sintassi non viene descritta. Negli esempi si fa riferimento frequentemente a una relazione di indirizzi, il cui contenuto è visibile nella figura successiva.

Figura 77.5. La relazione '**`Indirizzi (Codice, Cognome, Nome, Indirizzo, Telefono)`**' usata negli esempi del capitolo.

Indirizzi				
Codice	**Cognome**	**Nome**	**Indirizzo**	**Telefono**
1	Pallino	Pinco	Via Biglie 1	0222,222222
2	Tizi	Tizio	Via Tazi 5	0555,555555
3	Cai	Caio	Via Caini 1	0888,888888
4	Semproni	Sempronio	Via Sempi 7	0999,999999

77.2.1 Creazione di una relazione

La relazione di esempio, denominata '**`Indirizzi`**', potrebbe essere creata nel modo seguente:

```
sqlite> CREATE TABLE Indirizzi ( [Invio]
   ...>         Codice          integer, [Invio]
   ...>         Cognome         char(40), [Invio]
   ...>         Nome            char(40), [Invio]
```

```
...>          Indirizzo       varchar(60), [Invio]

...>          Telefono        varchar(40) [Invio]

...>      ); [Invio]
```

77.2.2 Modifica della relazione

SQLite consente soltanto l'aggiunta di attributi alle relazioni, mentre la loro eliminazione o il cambiamento del dominio (il tipo), non è ammissibile. Segue un esempio con cui si aggiunge un attributo:

```
sqlite> ALTER TABLE Indirizzi ADD COLUMN Comune char(30);
[Invio]
```

77.2.3 Inserimento dati in una relazione

L'esempio seguente mostra l'inserimento dell'indirizzo dell'impiegato «Pinco Pallino»:

```
sqlite> INSERT INTO Indirizzi VALUES ([Invio]

...>          01, [Invio]

...>          'Pallino', [Invio]

...>          'Pinco', [Invio]

...>          'Via Biglie 1', [Invio]

...>          '0222,222222'); [Invio]
```

In questo caso, si presuppone che i valori inseriti seguano la sequenza degli attributi, così come è stata creata la relazione in origine; tuttavia, si osservi che se la relazione ha degli attributi in più, si ottiene una segnalazione di errore e l'inserimento viene rifiutato.

Per indicare un comando più leggibile, evitando anche problemi quando dovessero esserci attributi ulteriori, che però non si vogliono prendere in considerazione, occorre aggiungere l'indicazione della sequenza degli attributi da compilare, come nell'esempio seguente:

```
sqlite> INSERT INTO Indirizzi (Codice, Cognome, Nome, [Invio]
   ...>                        Indirizzo, Telefono) [Invio]
   ...>        VALUES (01, 'Pallino', 'Pinco', [Invio]
   ...>                'Via Biglie 1', '0222,222222'); [Invio]
```

In questo modo, gli attributi che non vengono indicati, ricevono il valore predefinito (se esiste), oppure '**NULL**' in mancanza d'altro.

77.2.4 Eliminazione di una relazione

Una relazione può essere eliminata completamente attraverso l'istruzione '**DROP**'. L'esempio seguente elimina la relazione degli indirizzi degli esempi già mostrati:

```
sqlite> DROP TABLE Indirizzi; [Invio]
```

77.2.5 Interrogazioni semplici

L'esempio seguente emette tutto il contenuto della relazione degli indirizzi già vista negli esempi precedenti:

```
sqlite> SELECT * FROM Indirizzi; [Invio]
```

Il risultato può apparire in formati differenti; di solito si ottiene così:

```
1|Pallino|Pinco|Via Biglie 1|0222,222222
2|Tizi|Tizio|Via Tazi 5|0555,555555
3|Cai|Caio|Via Caini 1|0888,888888
4|Semproni|Sempronio|Via Sempi 7|0999,999999
```

Per ottenere un elenco incolonnato, occorre usare il comando '**.mode column**', ma in tal caso l'ampiezza delle colonne è fissa e le informazioni potrebbero apparire troncate:

```
sqlite> .mode column[Invio]

sqlite> SELECT * FROM Indirizzi;[Invio]

1          Pallino     Pinco       Via Biglie 1   0222,222222
2          Tizi        Tizio       Via Tazi 5     0555,555555
3          Cai         Caio        Via Caini 1    0888,888888
4          Semproni    Sempronio   Via Sempi 7    0999,999999
```

Per visualizzare anche l'intestazione delle colonne che appaiono, occorre utilizzare il comando '**.header on**':

```
sqlite> .header on[Invio]

sqlite> SELECT * FROM Indirizzi;[Invio]

Codice    Cognome     Nome        Indirizzo      Telefono
--------  ----------  ----------  -------------  -----------
1         Pallino     Pinco       Via Biglie 1   0222,222222
2         Tizi        Tizio       Via Tazi 5     0555,555555
3         Cai         Caio        Via Caini 1    0888,888888
4         Semproni    Sempronio   Via Sempi 7    0999,999999
```

Per ottenere un elenco ordinato in base al cognome e al nome (in caso di ambiguità), lo stesso comando si completa nel modo seguente:

```
sqlite> SELECT * FROM Indirizzi ORDER BY Cognome, Nome;[Invio]

  Codice    Cognome     Nome         Indirizzo      Telefono
  --------  ----------  ----------  -------------  -----------
  3         Cai         Caio        Via Caini 1    0888,888888
  1         Pallino     Pinco       Via Biglie 1   0222,222222
  4         Semproni    Sempronio   Via Sempi 7    0999,999999
  2         Tizi        Tizio       Via Tazi 5     0555,555555
```

La selezione degli attributi permette di ottenere un risultato che contenga solo quelli desiderati, permettendo anche di cambiarne l'intestazione. L'esempio seguente permette di mostrare solo i nominativi e il telefono, cambiando un po' le intestazioni:

```
sqlite> SELECT Cognome as cognomi, Nome as nomi,[Invio]

   ...>         Telefono as numeri_telefonici[Invio]

   ...>         FROM Indirizzi;[Invio]
```

Quello che si ottiene è simile all'elenco seguente:

```
  cognomi     nomi        numeri_telefonici
  ----------  ----------  -----------------
  Pallino     Pinco       0222,222222
  Tizi        Tizio       0555,555555
  Cai         Caio        0888,888888
  Semproni    Sempronio   0999,999999
```

La selezione delle tuple può essere fatta attraverso la condizione che segue la parola chiave '**WHERE**'. Nell'esempio seguente vengono selezionate le tuple in cui l'iniziale dei cognomi è compresa tra '**N**' e '**T**':

```
sqlite> SELECT * FROM Indirizzi[Invio]
```

```
   ...>        WHERE Cognome >= 'N' AND Cognome <= 'T'; [Invio]
```

Dall'elenco che si ottiene, si osserva che '**Caio**' è stato escluso:

```
Codice     Cognome      Nome         Indirizzo       Telefono
--------   ----------   ----------   -------------   -----------
1          Pallino      Pinco        Via Biglie 1    0222,222222
2          Tizi         Tizio        Via Tazi 5      0555,555555
4          Semproni     Sempronio    Via Sempi 7     0999,999999
```

Per evitare ambiguità possono essere indicati i nomi degli attributi prefissati dal nome della relazione a cui appartengono, separando le due parti con l'operatore punto ('.'). Nell'esempio seguente si selezionano solo il cognome, il nome e il numero telefonico, specificando il nome della relazione a cui appartengono gli attributi:

```
sqlite> SELECT Indirizzi.Cognome, Indirizzi.Nome,
Indirizzi.Telefono [Invio]

   ...>        FROM Indirizzi; [Invio]
```

Ecco il risultato:

```
Cognome      Nomi         Telefono
----------   ----------   -----------
Pallino      Pinco        0222,222222
Tizi         Tizio        0555,555555
Cai          Caio         0888,888888
Semproni     Sempronio    0999,999999
```

77.2.6 Interrogazioni simultanee di più relazioni

«

Se dopo la parola chiave '**FROM**' si indicano più relazioni (ciò vale anche se si indica più volte la stessa relazione), si intende fare riferimento a una relazione generata dal «prodotto» di queste. Si immagi-

ni di abbinare alla relazione '**Indirizzi**' la relazione '**Presenze**' contenente i dati visibili nella figura 77.13.

Figura 77.13. La relazione '**Presenze (Codice, Giorno, Ingresso, Uscita)**'.

Presenze			
Codice	**Giorno**	**Ingresso**	**Uscita**
1	01/01/2012	07:30	13:30
2	01/01/2012	07:35	13:37
3	01/01/2012	07:45	14:00
4	01/01/2012	08:30	16:30
1	01/02/2012	07:35	13:38
2	01/02/2012	08:35	14:37
4	01/02/2012	07:30	13:30

Ecco le istruzioni per crearla e per inserire la prima tupla dell'esempio:

```
sqlite> CREATE TABLE Presenze (Codice INTEGER, Giorno DATE,
[Invio]

   ...>                              Ingresso TIME, USCITA Time);
[Invio]

sqlite> INSERT INTO Presenze[Invio]

   ...>     VALUES (1, '2012-01-01', '07:30', '13:30');[Invio]
```

Come si può intendere, il primo attributo, '**Codice**', serve a identificare la persona per la quale è stata fatta l'annotazione dell'ingresso

e dell'uscita. Tale codice viene interpretato in base al contenuto della relazione '**Indirizzi**'. Si immagini di volere ottenere un elenco contenente tutti gli ingressi e le uscite, indicando chiaramente il cognome e il nome della persona a cui si riferiscono.

```
sqlite> SELECT Presenze.Giorno, Presenze.Ingresso,
Presenze.Uscita, [Invio]
   ...>        Indirizzi.Cognome, Indirizzi.Nome[Invio]
   ...>        FROM Presenze, Indirizzi[Invio]
   ...>        WHERE Presenze.Codice = Indirizzi.Codice; [Invio]
```

Ecco quello che si dovrebbe ottenere:

```
giorno      ingresso    uscita      cognome     nome
----------  ----------  ----------  ----------  ----------
01-01-2012    07:30:00    13:30:00    Pallino     Pinco
01-01-2012    07:35:00    13:37:00    Tizi        Tizio
01-01-2012    07:45:00    14:00:00    Cai         Caio
01-01-2012    08:30:00    16:30:00    Semproni    Sempronio
01-02-2012    07:35:00    13:38:00    Pallino     Pinco
01-02-2012    08:35:00    14:37:00    Tizio       Tizi
01-02-2012    07:40:00    13:30:00    Semproni    Sempronio
```

77.2.7 Alias

«

Una stessa relazione può essere presa in considerazione come se si trattasse di due o più relazioni differenti. Per distinguere tra questi punti di vista diversi, si devono usare degli alias, che sono in pratica dei nomi alternativi. Gli alias si possono usare anche solo per questioni di leggibilità. L'esempio seguente è la semplice ripetizione di quello mostrato nella sezione precedente, con l'aggiunta però della definizione degli alias '**Pre**' e '**Nom**'.

```
sqlite> SELECT Pre.Giorno, Pre.Ingresso, Pre.Uscita, [Invio]
   ...>      Nom.Cognome, Nom.Nome [Invio]
   ...>      FROM Presenze AS Pre, Indirizzi AS Nom [Invio]
   ...>      WHERE Pre.Codice = Nom.Codice; [Invio]
```

77.2.8 Viste

Attraverso una vista, è possibile definire una relazione virtuale:

```
sqlite> CREATE VIEW Presenze_dettagliate AS [Invio]
   ...>      SELECT Presenze.Giorno, Presenze.Ingresso, [Invio]
   ...>             Presenze.Uscita, [Invio]
   ...>             Indirizzi.Cognome, Indirizzi.Nome [Invio]
   ...>          FROM Presenze, Indirizzi [Invio]
   ...>          WHERE Presenze.Codice = Indirizzi.Codice;
[Invio]
```

L'esempio mostra la creazione della vista '**Presenze_dettagliate**', ottenuta dalle relazioni '**Presenze**' e '**Indirizzi**'. In pratica, questa vista permette di interrogare direttamente la relazione virtuale '**Presenze_dettagliate**', invece di utilizzare ogni volta un comando '**SELECT**' molto complesso, per ottenere lo stesso risultato.

77.2.9 Aggiornamento delle tuple

La modifica di tuple già esistenti avviene attraverso l'istruzione '**UPDATE**', la cui efficacia viene controllata dalla condizione posta

dopo la parola chiave '**WHERE**'. Se tale condizione manca, l'effetto delle modifiche si riflette su tutte le tuple della relazione.

L'esempio seguente, aggiunge un attributo alla relazione degli indirizzi, per contenere il nome del comune di residenza degli impiegati; successivamente viene inserito il nome del comune '**Sferopoli**' in base al prefisso telefonico.

```
sqlite> ALTER TABLE Indirizzi ADD COLUMN Comune char(30);
[Invio]

sqlite> UPDATE Indirizzi[Invio]

   ...>     SET Comune='Sferopoli'[Invio]

   ...>     WHERE Telefono >= '022' AND Telefono < '023';
[Invio]
```

In pratica, viene aggiornata solo la tupla dell'impiegato '**Pinco Pallino**'.

77.2.10 Cancellazione delle tuple

«

L'esempio seguente elimina dalla relazione delle presenze le tuple riferite alle registrazioni del giorno 01/01/2012 e le eventuali antecedenti.

```
sqlite> DELETE FROM Presenze WHERE Giorno <= '2012-01-01';
[Invio]
```

77.2.11 Inserimento in una relazione esistente

«

L'esempio seguente aggiunge alla relazione dello storico delle presenze le registrazioni vecchie che poi vengono cancellate:

```
sqlite> INSERT INTO PresenzeStorico ([Invio]
```

```
   ...>          PresenzeStorico.Codice, [Invio]
   ...>          PresenzeStorico.Giorno, [Invio]
   ...>          PresenzeStorico.Ingresso, [Invio]
   ...>          PresenzeStorico.Uscita) [Invio]
   ...>      SELECT Presenze.Codice, Presenze.Giorno, [Invio]
   ...>             Presenze.Ingresso, Presenze.Uscita [Invio]
   ...>          FROM Presenze [Invio]
   ...>          WHERE Presenze.Giorno <= '2012-01-01'; [Invio]
sqlite> DELETE FROM Presenze [Invio]
   ...>         WHERE Giorno <= '2012-01-01'; [Invio]
```

77.3 Riferimenti

«

- *SQLite*, *http://www.sqlite.org*

[1] **SQLite** dominio pubblico