SQLite 3 with PHP Essential Training

Shavit Ilan Oct 7, 2012, Rev 1

Table of Contents

Chapter 02\04: Setting up SID and Exercise files

Getting better XAMPP performance on Win7

- Control Panel
- Device manager
- Disk Drives
- Right Click On Disk
- Policies
- Mark 'v' on “Turn off Windows write-cache...”

Install SID and CRUD

- Update sid.php and crud.php files with the default location of: album.db, world.db, test.db files (copy these files to htdocs dir)
- chmod 777 to album.db, world.db, test.db

Test SID installation

- Open the browser and launch localhost/LOCATION/sid.php
- SID will open
- Choose “test.db”
- Run these code lines:

```
CREATE TABLE t (a, b);
INSERT INTO t VALUES (1, 2);
SELECT * FROM t;
```

- Check the results...
- Run: **DROP TABLE t;**

Test CRUD installation

- Open the browser and launch localhost/LOCATION/crud.php
- Check the you get the required result

Remarks: CRUD is abbreviation of CREATE, READ, UPDATE, DELETE – the basic function of the database

Remarks In SID: - - (MINUS MINUS) remarks the line (will not be execute)

Chapter 02\05: Using the command line tools

```
$ sqlite3 test.db
SQLite version 3.6.16
Enter ".help" for instructions
Enter SQL statements terminated with a ";"

sqlite> select * from sqlite_master;
table|customer|customer|2|CREATE TABLE customer (
    id              INTEGER PRIMARY KEY,
    name            TEXT,
    address         TEXT,
    city            TEXT,
    state           TEXT,
    zip             TEXT
)
table|item|item|3|CREATE TABLE item (
    id              INTEGER PRIMARY KEY,
    name            TEXT,
    description     TEXT
)
table|sale|sale|4|CREATE TABLE sale (
    id              INTEGER PRIMARY KEY,
    item_id         INTEGER,
    customer_id     INTEGER,
    date            TEXT,
    quantity        INTEGER,
    price           INTEGER
)

.quit
sqlite3 test.db 'select * from customer';
     running command directly from the OS
sqlite3 newtest.db < test-sqlite3.sql;
     CREATE database from sql backup file
```

Chapter 03\02: Creating a database with PHP

```
<?php
  define('DATABASE', '/opt/lampp/htdocs/test.db');
 try {
    $db = new PDO('sqlite:' . DATABASE);
    $db->exec('CREATE TABLE IF NOT EXISTS t (a, b, c)');
    print 'Table t sucessfully created';
 } catch(PDOException $e) {
        print $e->getMessage();
 }
?>
```

Chapter 03\04: Creating a Table in SQLite

```
CREATE TABLE t (a INT, b REAL, c TEXT);
select * from sqlite_master; /* We will see the new table */
```

Chapter 03\05: Creating a Table in PHP

```
$db = new PDO('sqlite:' . DATABASE);
$db->exec('CREATE TABLE IF NOT EXISTS t (a INT, b REAL , c
     TEXT)');
```

Chapter 03\06: Creating \ Droping Index

The good:
Increasing performance in searches, joins..

The Cost:
Slower inserts and updates and a larger DB files.

```
CREATE INDEX IF NOT EXISTS co_code ON country(code);
```

co_code is the index name
country is the table
code is the field
DROP INDEX co_code;

Chapter 03\07: Indexing ID fields

```
CREATE TABLE t (id INTEGER PRIMARY KEY AUTOINCREMENT,a,b,c);
```

Remarks: You can drop AUTOINCREMENT (in this case sqlite will reuse old deleted indexes)

Chapter 04\01: SQLite Data Type

Most databases uses static typing: If you insert a number to text column it will be a text (but in the opposite direction you can't put text in a number field).

In SQLite the column type determine by the value:
Every value stored in a table has a storage class. There are 5 storage classes:
NULL, REAL, INTEGER, TEXT, BLOB.
Each storage class can support more than one storage type.

- NULL – Can be used only for NULL
- INTEGER – Integer value that can be stored as 1,2,3,4,6,8 bytes
- REAL – Floating point number that always stored in 8 byte IEEE floating point num
- TEXT – Is a text string encoding in UTF-8 \ UTF-16
- BLOB – Is a blob of data and stored exactly as provided

Remark:

- SQLite has no boolean type. It can implement with INTEGER with 0,1 values

- SQLite has no date and time type. It will stored as INTEGER, TEXT or REAL (depending on the represented value)

SQLite converts TEXT to INTEGER\REAL only if the conversion is lossless: it must preserve 15 significant digits (in both directions). If lossless conversion is not possible SQLite will use TEXT (even that you declared the column value as INTEGER).

The declaration is only a recommendation to SQLite and it will be use only on a lossless conversion.

SQL:

```
CREATE TABLE t (a INT, b REAL, c TEXT);
INSERT INTO t(a,b,c) VALUES (1, 2, 3);
INSERT INTO t(a,b,c) VALUES ('a', 'b', 'c');
INSERT INTO t(a,b,c) VALUES ('one', 'two', 'three');
INSERT INTO t(a,b,c) VALUES ('1', '2', '3');
SELECT * FROM t;
SELECT TYPEOF(a), TYPEOF(b), TYPEOF(c) FROM t;
```

Query 6:

a	b	c
1	2.0	3
a	b	c
one	two	three
1	2.0	3

Query 7:

TYPEOF(a)	TYPEOF(b)	TYPEOF(c)
integer	real	text
text	text	text
text	text	text
integer	real	text

Chapter 04\02: CAST

In order to make monetary calculation you must use INTEGER and works in cents (REAL can make mistakes in monetary calculation)

```
CREATE TABLE t (product TEXT, price INT);
INSERT INTO t(product,price) VALUES ('table', 223);
SELECT product, CAST(price AS REAL)/100 AS PRICE_IN_DOLLAR
     from t;
```

product	PRICE_IN_DOLLAR
table	2.23

Chapter 04\05: Storing large data with BLOB

This chapter demonstrate how to enter a picture to SQLite DB using PHP code.

Chapter 04\06: BOOLEAN and CASE WHEN

SQLite has no Boolean type. However we can implement it by using Integer:

0 = False

1 (or any number <> 0) = True

SQL:

```
CREATE TABLE booltest(a INT, b INT);
INSERT INTO booltest(a,b) VALUES (0,1);
SELECT * FROM booltest;
SELECT
  CASE WHEN a THEN 'TRUE' ELSE 'FALSE' END as boola,
  CASE WHEN b THEN 'TRUE' ELSE 'FALSE' END as boolb
FROM booltest;
```

Query 3:

a	b
0	1

Query 4:

boola	boolb
FALSE	TRUE

Remarks: "CASE WHEN a THEN 'TRUE' ELSE 'FALSE' END as boola," means:

Display 'TRUE' if a is True and set the display title to boolb.

Chapter 04\07: Storing dates and times, DATETIME, JULIANDAY

SQLite has no 'Date And Time' type. Date And Time stored as TEXT (but sometime as

INTEGER \ REAL – depending on the representation)

SQL:

```
CREATE TABLE t (d1 TEXT, d2 TEXT);
INSERT INTO t VALUES (DATETIME('now'), DATETIME('now', 'localtime'));
SELECT d1, TYPEOF(d1), d2, TYPEOF(d2) FROM t
```

Query 3:

d1	TYPEOF(d1)	d2	TYPEOF(d2)
2012-10-07 07:35:16	text	2012-10-07 09:35:16	text

DATETIME('now') – UTC local time

DATETIME('now', 'localtime') – ISRAEL locate time

Notice that date and time stored as TEXT

You can store date and time as REAL using JULIANDAY, and as INTEGER using epoch:

SQL:

```
CREATE TABLE t (d1 REAL, d2 INT);
INSERT INTO t VALUES (JULIANDAY(), STRFTIME('%s', 'now'));
SELECT d1, TYPEOF(d1), d2, TYPEOF(d2) FROM t;
-- Convering back to Date And Time format
SELECT DATETIME(d1), DATETIME(d2, 'unixepoch') FROM t;
```

Query 3:

d1	TYPEOF(d1)	d2	TYPEOF(d2)
2456207.82581493	real	1349596150	integer

Query 4:

DATETIME(d1)	DATETIME(d2, 'unixepoch')
2012-10-07 07:49:10	2012-10-07 07:49:10

unixepoch doesn't have any special function in SQLite so we use STRFTIME

Usually you will use TEXT to represent Date And Time

Chapter 05\01: UPDATE table (SET command)

SQL:

```
SELECT * FROM t;
UPDATE t SET b=5.0, c='qqq' WHERE A=1;
SELECT * FROM t;
```

Query 1:

a	b	c
1	2.0	abc
2	3.0	def
3	4.0	ghi

Query 3:

a	b	c
1	5.0	qqq
2	3.0	def
3	4.0	ghi

*Remark: Don't forget to use WHERE condition in order not to **rewrite all your date!***

Chapter 05\02: SELECT, SUB-SELECT, SUDO JOIN

SQL:

```
SELECT 'Hello World' As Bob;

-- SUB-SELECT Example
SELECT * FROM track WHERE album_id IN (
    SELECT id FROM album WHERE artist IN ('Jimi Hendrix', 'Johnny Winter')
);
```

Query 1:

Bob
Hello World

Query 2:

id	album_id	title	track_number	duration
14	11	Johnny B. Goode	1	285
15	11	Lover Man	2	185
16	11	Blue Suede xShoes	3	266
17	11	Voodoo Chile	4	469
18	11	The Queen	5	160
19	11	Sgt. Pepper's Lonely Hearts Club Band	6	76
20	11	Little Wing	7	194
21	11	Red House	8	786
50	16	Good Morning Little Schoolgirl	1	285
51	16	It's My Own Fault	2	734
52	16	Jumpin' Jack Flash	3	266

SQL:

```
-- SUDO JOIN EXAMPLE
SELECT * FROM album WHERE id=1;
SELECT * FROM track WHERE id=1;

SELECT a.title AS album, t.title AS track, t.track_number
  FROM album AS a, track AS t
  WHERE a.id = t.album_id
  ORDER BY a.title, t.track_number;
```

Query 1:

id	title	artist	label	released
1	Two Men with the Blues	Willie Nelson and Wynton Marsalis	Blue Note	2008-07-08

Query 2:

id	album_id	title	track_number	duration
1	1	Bright Lights Big City	1	320

Query 3:

album	track	track_number
Apostrophe	Don't Eat the Yellow Snow	1
Apostrophe	Nanook Rubs It	2
Apostrophe	St. Alfonzo's Pancake Breakfast	3
Apostrophe	Father O'Blivion	4
Apostrophe	Cosmik Debris	5

Chapter 05\03: JOIN

SQL:

```
SELECT * from CountryLanguage WHERE countrycode='ISR';
SELECT * FROM country WHERE code='ISR';

-- JOIN STATEMENT
SELECT c.name, l.language
   FROM countrylanguage AS l
  JOIN country as c
  ON l.countrycode = c.code
```

Query 1:

CountryCode	Language	IsOfficial	Percentage
ISR	Arabic	1	18
ISR	Hebrew	1	63.1
ISR	Russian	0	8.9

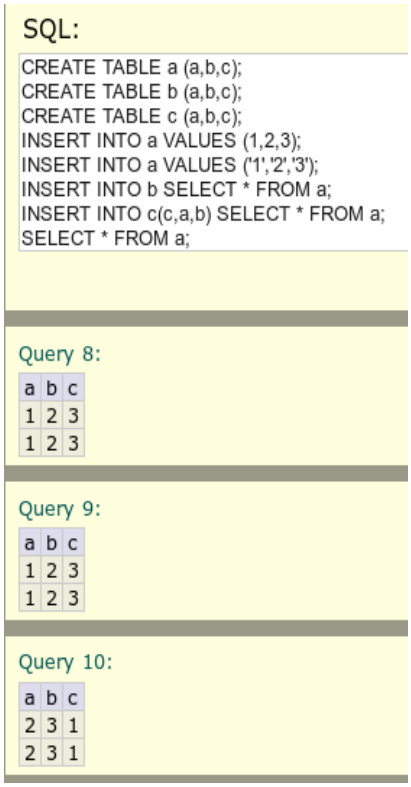
Query 2:

Code	Name	Continent	Region	SurfaceArea	IndepYear	Population	LifeExpectancy
ISR	Israel	Asia	Middle East	21056.0	1948	6217000	78.6

Query 3:

Name	Language
Israel	Arabic
Israel	Hebrew
Israel	Russian

Chapter 05\04: INSERT



Chapter 05\05: DELETE

```
DELETE * FROM table_name WHERE colomn_name='some_text';
DELETE * FROM table_name;
```

Will empty the entire table so don't forget to write the WHERE statement

Chapter 06\01: SQLITE3 EXPRESSION

IS, IS NOT	For comparison
LIKE, GLOB	Glob is like 'like' and it's unique to sqlite3
BETWEEN	X is between y and z (x>=y, x<=z)
IN, NOT IN	Inclusion in list or query result
CASE	Like IF, THAN, ELSE

Chapter 06\02: BETWEEN

```
SELECT Name, population FROM country
  WHERE population BETWEEN 5000000 AND 10000000
  ORDER BY population DESC
```

Name	Population
Portugal	9997600
Tunisia	9586000
Senegal	9481000
Zambia	9169000

BETWEEN is evaluate only once so it's better than using other expressions

SQL:

```
SELECT Name FROM country
  WHERE name BETWEEN 'G' AND 'R'
  ORDER BY name;
```

Name
Gabon
Gambia
Georgia
Germany
Ghana
Gibraltar
Greece

Result will not include R because Rxxx is greater than R. To include R use: BETWEEN 'G' AND 'Rzzz';

Notice that lower case is above upper case

Chapter 06\03: LIKE

SELECT * FROM city WHERE name LIKE 'z%' ORDER BY name

looking for city name STARTING with z

LIKE is case insensitive

SELECT * FROM city WHERE name LIKE '_z%' ORDER BY name

looking for city where second character is z

'_' means any letter

SELECT * FROM city WHERE name GLOB '?z*' ORDER BY name

GLOB works only in SQLite

GLOB is CASE SENSITIVE

'' For any number of char, '?' For one char*

GLOB is more flexible and we work like UNIX (, ?)*

SELECT * FROM city WHERE name GLOB '[zk]*' ORDER BY name

City starts with 'lowercase z' or 'k'

Chapter 06\04: Simple math with arithmetic operators

SELECT 7 * 5; => 35

SELECT 7 / 5; =>1

SELECT 7.0 / 3; => 2.3333

SELECT 7 % 5; => 2

SQLite support all SQL arithmetic operators

Chapter 06\05: Matching in a List with IN

SELECT * FROM city WHERE countrycode IN ('USA', 'GRB');

SELECT * FROM city WHERE countrycode IN

(

SELECT code FROM country WHERE name IN ('United States', 'United Kindom')

)

ORDER BY name

Chapter 06\06: CASE

CASE is the only conditional statement in SQL and is equivalent to: if,then, else

Look carefully at the following example and understand the result values:

In the 3'td example || use for concatenating and it order to understand this complex query you must read it from the inner query to the outer...

SQL:

```
CREATE TABLE booltest (a, b);
INSERT INTO booltest VALUES (1,0);
SELECT
   CASE WHEN a THEN 'TRUE' ELSE 'FALSE' END as BoolA
   FROM booltest;
```

Query 3:

BoolA
TRUE

SQL:

```
CREATE TABLE booltest (a, b);
INSERT INTO booltest VALUES (0,0);
SELECT
   CASE WHEN a THEN 'TRUE' ELSE 'FALSE' END as BoolA
   FROM booltest;
```

Query 3:

BoolA
FALSE

SQL:

```
SELECT artist, album, track, trackno,
   m || ':' || CASE WHEN s < 10 THEN '0' || s ELSE s END AS duration
   FROM (
      SELECT a.artist AS artist, a.title AS album, t.track_number AS trackno, t.title AS track,
         t.duration / 60 AS m, t.duration % 60 AS s
         FROM track AS t JOIN album AS a ON a.id = t.album_id
         WHERE t.album_id IN (
            SELECT id FROM album WHERE artist IN ('Jimi Hendrix', 'Johnny Winter')
         )
         ORDER BY album, trackno
   );
```

Go

artist	album	track	trackno	duration
Jimi Hendrix	Hendrix in the West	Johnny B. Goode	1	4:45
Jimi Hendrix	Hendrix in the West	Lover Man	2	3:05
Jimi Hendrix	Hendrix in the West	Blue Suede xShoes	3	4:26
Jimi Hendrix	Hendrix in the West	Voodoo Chile	4	7:49
Jimi Hendrix	Hendrix in the West	The Queen	5	2:40
Jimi Hendrix	Hendrix in the West	Sgt. Pepper's Lonely Hearts Club Band	6	1:16
Jimi Hendrix	Hendrix in the West	Little Wing	7	3:14
Jimi Hendrix	Hendrix in the West	Red House	8	13:06

Chapter 06\08: CAST

SQL:

```
SELECT TYPEOF(1);
SELECT TYPEOF(CAST(1 AS TEXT));
SELECT TYPEOF(CAST(1 AS REAL));
SELECT TYPEOF(CAST(1 AS NUMERIC));
SELECT TYPEOF(CAST(NULL AS TEXT));
```

Query 1:

TYPEOF(1)
integer

Query 2:

TYPEOF(CAST(1 AS TEXT))
text

Query 3:

TYPEOF(CAST(1 AS REAL))
real

Query 4:

TYPEOF(CAST(1 AS NUMERIC))
integer

Query 5:

TYPEOF(CAST(NULL AS TEXT))
null

SELECT 7 / 3 => 2

SELECT CAST(7 AS REAL) / 3 => 2.3333

Casing NULL will always be NULL

Chapter 07\01: Length Of String

SELECT name, LENGTH(name) FROM country WHERE LENGTH(name)>10;

Display name, length of name only if the name length > 10

SQL:

```
SELECT name, LENGTH(name) AS Len FROM country
  WHERE Len BETWEEN 10 AND 12
  ORDER BY Len DESC;
```

Name	Len
Burkina Faso	12
Cook Islands	12

Chapter 07\02: Changing case with UPPER and LOWER

SQL:

```
CREATE TABLE t ( a, b );
INSERT INTO t VALUES ( 'Aaa', 'Bbb' );
INSERT INTO t VALUES ( 'aaa', 'bab' );
INSERT INTO t VALUES ( 'ABA', 'BbB' );
SELECT * FROM t ORDER BY a;
SELECT * FROM t ORDER BY UPPER(a);
SELECT * FROM t ORDER BY a COLLATE NOCASE;
```

Query 5:

a	b
ABA	BbB
Aaa	Bbb
aaa	bab

Query 6:

a	b
Aaa	Bbb
aaa	bab
ABA	BbB

Query 7:

a	b
Aaa	Bbb
aaa	bab
ABA	BbB

Notice that when sorting strings 'A' comes before 'a' . To solve this problem use UPPER, LOWER or COLLATE NOCASE.

UPPER('axsd') => 'AXSD'

Chapter 07\03: Reading parts of a string with SUBSTR

SQL:

```
-- SUBSTR
CREATE TABLE t ( a, b );
INSERT INTO t VALUES ( 'NY0123', 'US4567' );
INSERT INTO t VALUES ( 'AZ0123', 'UK4567' );
INSERT INTO t VALUES ( 'CA0123', 'FR4567' );
SELECT SUBSTR(a, 1, 2) AS State, SUBSTR(a, 3) AS StValue,
    SUBSTR(b, 1, 2) AS Country, SUBSTR(b, 3) AS CoValue FROM t;
```

Query 5:

State	StValue	Country	CoValue
NY	0123	US	4567
AZ	0123	UK	4567
CA	0123	FR	4567

SUBSTR(a, 1, 2) => substring a, starting 2 character from position 1 (in SQLite 1 is the first character)

SUBSTR(a, 3) => substring a, starting from character 3 till the end of the string

Chapter 07\04: Changing parts of a string with REPLACE

SQL:

```
-- REPLACE
CREATE TABLE t ( a, b );
INSERT INTO t VALUES ( 'Thing One', 'Thing Two' );
INSERT INTO t VALUES ( 'Thing Three', 'Thing Four' );
INSERT INTO t VALUES ( 'Thing Five', 'Thing Six' );
SELECT * FROM t;
SELECT REPLACE(a, 'Thing', 'Other') FROM t;
```

Query 5:

a	b
Thing One	Thing Two
Thing Three	Thing Four
Thing Five	Thing Six

Query 6:

REPLACE(a, 'Thing', 'Other')
Other One
Other Three
Other Five

Chapter 07\05: Trimming blank spaces with TRIM, LTRIM, RTRIM

SQL:

```
-- TRIM
CREATE TABLE t ( a, b );
INSERT INTO t VALUES ( '  Thing One  ', '  Thing Two  ' );
INSERT INTO t VALUES ( '  Thing Three  ', '  Thing Four  ' );
INSERT INTO t VALUES ( '  Thing Five  ', '  Thing Six  ' );
SELECT '[' || a || ']', '[' || b || ']' FROM t;
SELECT '[' || TRIM(a) || ']', '[' || b || ']' FROM t;
```

Query 5:

'[' || a || ']'	'[' || b || ']'
[Thing One]	[Thing Two]
[Thing Three]	[Thing Four]
[Thing Five]	[Thing Six]

Query 6:

'[' || TRIM(a) || ']'	'[' || b || ']'
[Thing One]	[Thing Two]
[Thing Three]	[Thing Four]
[Thing Five]	[Thing Six]

TRIM trim blank spaces from **both side** of the string

LTRIM trim blank spaces from the **left** side of the string

RTRIM trim blank spaces from the **right** side of the string

|| - For concatenating

'[', ']' - For showing the spaces (HTML filter out extra spaces)

Chapter 07\06: ABS – Absolute Values

SQL:

```
-- ABS
CREATE TABLE t ( a, b );
INSERT INTO t VALUES ( 1, 2 );
INSERT INTO t VALUES ( 3, 4 );
INSERT INTO t VALUES ( -1, -2 );
INSERT INTO t VALUES ( -3, -4 );
SELECT a, b FROM t;
SELECT ABS(a), b FROM t;
```

Query 6:

a	b
1	2
3	4
-1	-2
-3	-4

Query 7:

ABS(a)	b
1	2
3	4
1	-2
3	-4

Chapter 07\07: Rounding values with ROUND

SQL:

```
-- ROUND
CREATE TABLE t ( a, b );
INSERT INTO t VALUES ( 123.456, 456.789 );
INSERT INTO t VALUES ( -123.456, -456.789 );
SELECT * FROM t;
SELECT ROUND(a), ROUND(b) FROM t;
SELECT ROUND(a, 2), ROUND(b,2)  FROM t;
SELECT CAST(ROUND(a) AS INT), CAST(ROUND(b) AS INT) FROM t;
```

Query 4:

a	b
123.456	456.789
-123.456	-456.789

Query 5:

ROUND(a)	ROUND(b)
123.0	457.0
-123.0	-457.0

Query 6:

ROUND(a, 2)	ROUND(b,2)
123.46	456.79
-123.46	-456.79

Query 7:

CAST(ROUND(a) AS INT)	CAST(ROUND(b) AS INT)
123	457
-123	-457

Chapter 07\10: Getting the version of your SQLite library

SQL:

```
SELECT SQLITE_VERSION();
```

sqlite_version()
3.7.7.1

Chapter 07\11: Creating user-defined functions (UDF)

User-defined functions can be write in C or PHP

SQL:

```
SELECT album_id, SUM_SEC_TO_TIME(duration) AS length FROM track GROUP BY album_id;
```

album_id	length
1	0:53:27
11	0:49:30
12	0:35:39
13	0:40:24
16	0:40:32
17	0:31:47
18	0:45:54

In SID in “function_init” we declare the function:

```
$dbh->sqliteCreateFunction('TIME_TO_SEC', 'time_to_sec', 1);
```

The function itself appear in the boby of the PHP code:

```
// TIME_TO_SEC( time TEXT ) -- 'mm:ss' function time_to_sec( $time )
{
    if(is_null($time)) return NULL;
    $t = explode(':', $time, 2);
    $m = intval($t[0]);
    $s = intval($t[1]);
    return ( $m * 60 ) + $s;
}
```

Chapter 08\01: Understanding aggregate functions

Aggregate functions are function that operate on multiple rows at the time.

Example:

```
SELECT  count(*) FROM city;
SELECT AVG(population) FROM city WHERE countrycode = 'USA';
```

GROUP BY may be used in aggregate functions:

```
SELECT countrycode, AVG(population) FROM city
GROUP BY countrycode;
```

In this example all raws are GROUP BY their countrycode and on those raws the AVG function works.

we will get one raw per groups of raws.

SQL:

```
SELECT countrycode, AVG(population) FROM city GROUP BY countrycode;
```

CountryCode	AVG(population)
ABW	29034.0
AFG	583025.0
AGO	512320.0

The HAVING clause works like WHERE for aggregation:

SQL:

```
SELECT countrycode, AVG(population) AS avgpop FROM city
GROUP BY countrycode
HAVING avgpop > 1000000
```

CountryCode	avgpop
GIN	1090610.0
HKG	1650316.5
SGP	4017733.0
URY	1236000.0

Chapter 08\02: Counting rows with COUNT

```
SELECT COUNT(*) FROM City;
SELECT District, COUNT(*) FROM City GROUP BY District;

SELECT District, COUNT(*) AS Count FROM City
    GROUP BY District
    HAVING Count > 10
    ORDER BY Count DESC;
```

Chapter 08\03: Building with the SUM and TOTAL functions

```
SELECT SUM(Population) FROM Country;
```

In SQLite SUM will bring INTEGER result if all rows values are integer.

To get floating point number you can use a special SQLite function named TOTAL.

```
SELECT TOTAL(Population) FROM Country;

SELECT Continent, SUM(Population) AS Pop FROM Country
    GROUP BY Continent
    ORDER BY Pop DESC;
```

Continent	Pop
Asia	3705025700
Africa	784475000
Europe	730074600
North America	482993000
South America	345780000
Oceania	30401150
Antarctica	0

Chapter 08\04: MIN and MAX

```
SELECT MAX(SurfaceArea) FROM Country;

SELECT Continent, MAX(SurfaceArea) FROM Country GROUP BY
    Continent;
```

```
SELECT c.Name as Country, csa.Continent, csa.MaxSA FROM
  (SELECT Continent, MAX(SurfaceArea) as MaxSA FROM Country
    GROUP BY Continent) AS csa
JOIN Country AS c
ON c.SurfaceArea = csa.MaxSA
ORDER BY MaxSA DESC;
```

The last query is complication because of the way that GROUP BY works.

There isn't any way to know which name will be choose from the aggregates rows so we need to use

It as Subselect. Than we join another table and choose the right name

Country	Continent	MaxSA
Russian Federation	Europe	17075400.0
Antarctica	Antarctica	13120000.0
Canada	North America	9970610.0
China	Asia	9572900.0
Brazil	South America	8547403.0
Australia	Oceania	7741220.0
Sudan	Africa	2505813.0

Chapter 08\05: AVG

```
SELECT AVG(Population) FROM City;


SELECT District, AVG(Population) AS AvgPop FROM City
    GROUP BY District
    HAVING District != '' AND AvgPop > 1000000
    ORDER BY AvgPop DESC;
```

District	AvgPop
Seoul	9981619.0
Shanghai	9696300.0
Jakarta Raya	9604900.0
Kairo	6789479.0
Lima	6464693.0
Chongqing	6351600.0

Chapter 09\01: Understanding SQLite support for dates and times

SQLite does not have separate type for date and times

Date and times can be stored in any of three format:

- Text: YYYY-MM-DD HH:MM:SS.SSS
- Real: real number that represent number of days since Julian day
- Integer: Number of days since 1970-01-01

We can convert between these types with built-in functions:

SQL:

```
CREATE TABLE t ( d1, d2 );
INSERT INTO t VALUES ( DATETIME('now'), DATETIME('now', '+7 days'));
INSERT INTO t VALUES ( JULIANDAY('now'), JULIANDAY('now', '+7 days'));
SELECT d1, TYPEOF(d1), d2, TYPEOF(d2) FROM t;
SELECT DATETIME(d1), DATETIME(d2) FROM t;
```

Query 4:

d1	TYPEOF(d1)	d2	TYPEOF(d2)
2012-11-09 19:06:02	text	2012-11-16 19:06:02	text
2456241.29585876	real	2456248.29585876	real

Query 5:

DATETIME(d1)	DATETIME(d2)
2012-11-09 19:06:02	2012-11-16 19:06:02
2012-11-09 19:06:02	2012-11-16 19:06:02

Unix Epoch times are handled a little differently

SQL:

```
CREATE TABLE t ( d1 int, d2 int );
INSERT INTO t VALUES ( STRFTIME('%s', 'now'), STRFTIME('%s', 'now', '+7 days'));
SELECT d1, TYPEOF(d1), d2, TYPEOF(d2) FROM t;
SELECT DATETIME(d1, 'unixepoch'), DATETIME(d2, 'unixepoch') FROM t;
```

Query 3:

d1	TYPEOF(d1)	d2	TYPEOF(d2)
1352488230	integer	1353093030	integer

Query 4:

DATETIME(d1, 'unixepoch')	DATETIME(d2, 'unixepoch')
2012-11-09 19:10:30	2012-11-16 19:10:30

Chapter 09\02: Getting readable, sortable dates and times

```
CREATE TABLE t ( d1 TEXT, d2 TEXT );
INSERT INTO t VALUES ( DATETIME('now'), DATETIME('now', 'localtime'));
SELECT * FROM t;
```

Query 3:

d1	d2
2012-11-09 19:20:43	2012-11-09 21:20:43

'now', 'now+7' are modifiers. You can use +/- days, months, years, hours, seconds etc...

list of modifiers exist in this link: http://www.sqlite.org/lang_datefunc.html

TIME(field) – will give only the TIME part

DATE(field) – will give only the DATE part

DATETIME(field) – will give both DATE and TIME

Chapter 09\03: Getting high-resolution dates and times with JULIANDAY

JULIANDAY format are high-resolution format that give you access to 1/1000 of seconds.

In the following example you can notice the accurately of JULIANDAY.

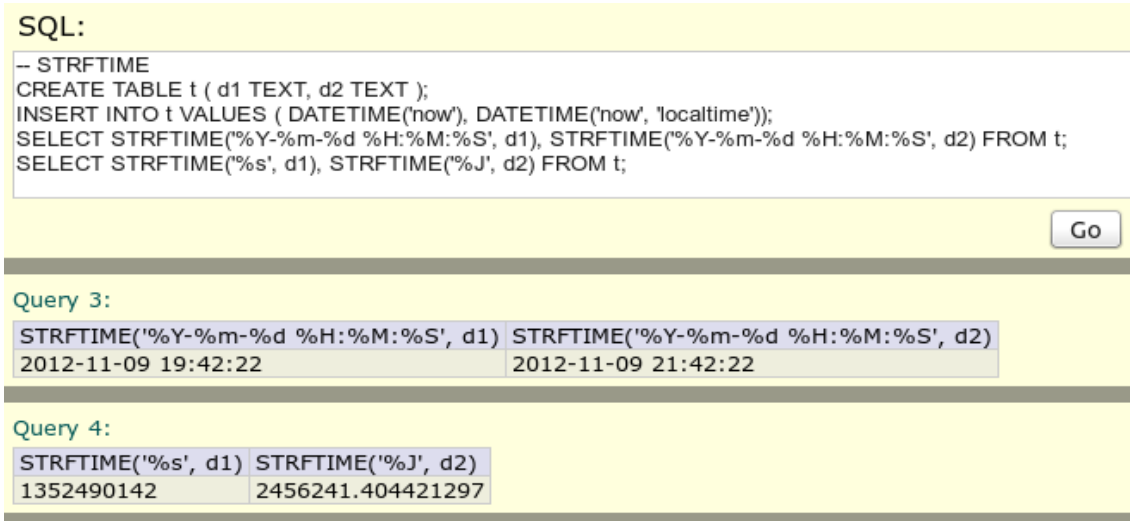
```
DROP TABLE IF EXISTS t;
CREATE TABLE t ( d1 REAL, d2 REAL );
INSERT INTO t VALUES ( JULIANDAY('now'), JULIANDAY('now', 'localtime') );
INSERT INTO t VALUES ( JULIANDAY('now'), JULIANDAY('now', 'localtime') );
INSERT INTO t VALUES ( JULIANDAY('now'), JULIANDAY('now', 'localtime') );
INSERT INTO t VALUES ( JULIANDAY('now'), JULIANDAY('now', 'localtime') );
INSERT INTO t VALUES ( JULIANDAY('now'), JULIANDAY('now', 'localtime') );
INSERT INTO t VALUES ( JULIANDAY('now'), JULIANDAY('now', 'localtime') );
INSERT INTO t VALUES ( JULIANDAY('now'), JULIANDAY('now', 'localtime') );
SELECT d1, STRFTIME('%Y-%m-%d %H:%M:%f', d1) AS d1_ISO, d2, STRFTIME('%Y-%m-%d %H:%M:%f', d2) AS d2_ISO FROM t;
```

Go

Query 10:

d1	d1_ISO	d2	d2_ISO
2456241.31714618	2012-11-09 19:36:41.430	2456241.40047951	2012-11-09 21:36:41.430
2456241.31714696	2012-11-09 19:36:41.497	2456241.40048029	2012-11-09 21:36:41.497
2456241.31714773	2012-11-09 19:36:41.564	2456241.40048106	2012-11-09 21:36:41.564
2456241.3171485	2012-11-09 19:36:41.630	2456241.40048183	2012-11-09 21:36:41.630
2456241.31714914	2012-11-09 19:36:41.686	2456241.40048248	2012-11-09 21:36:41.686
2456241.31714992	2012-11-09 19:36:41.753	2456241.40048325	2012-11-09 21:36:41.753
2456241.31715069	2012-11-09 19:36:41.820	2456241.40048403	2012-11-09 21:36:41.820

Chapter 09\04: Formatting dates and times with STRFTIME



STRFTIME is used to format date and time to other than the standard date and time format.

You can use %Y to get only the year, %H to get the hour:

```
SELECT STRFTIME('%Y',d1)  => 2012
```

Chapter 10\01: Understanding collation

Collation order is how SQLite compares or sort data

SQLite supports three collation order:

- BINARY - compare values according to the binary values of the field
- NOCASE - works with ASCII. The compare relate to the lowercase field value
- RTRIM – like BINARY but ignore trailing spaces

Collation can be declared in the column definition:

```
CREATE TABLE t (… colomn_name TYPE COLLATE BINARY, …);
```

Collation may be specified using COLLATE operator:

```
SELECT * FROM t COLLATE BINARY;
```

Chapter 10\02: Sorting results with ORDER BY

```
SELECT * FROM Country
SELECT * FROM Country ORDER BY Region
SELECT * FROM Country ORDER BY Region, Population
SELECT * FROM Country ORDER BY Region, Population DESC
```

First it will be sorted by 'Region' than by 'Population'

Chapter 10\03: Removing duplicate results with DISTINCT

```
SELECT DISTINCT Region FROM Country;
SELECT DISTINCT CountryCode FROM City;
SELECT DISTINCT CountryCode, District FROM City;
```

Distinct will show only one result and all the result will be alphabetic sorted

Distinct will work on all the combination of CountryCode, Districy so you can get resilt like this:

AFG, Herat

AFG, Kabol

In this example each RAW is distinct from the other

Chapter 10\05. Working with primary key indexes

Only one primary key is allowed in one table

UNIQUE is another index. We can't declared it as primary key so we call it UNIQUE.

```
CREATE TABLE t (code PRIMARY KEY, value TEXT, ycode UNIQUE );
SELECT * FROM SQLITE_MASTER;
INSERT INTO t VALUES ( 'a', 'thing one', 'one' );
INSERT INTO t VALUES ( 'b', 'thing two', 'two' );
INSERT INTO t VALUES ( 'c', 'thing three', 'three' );
INSERT INTO t VALUES ( 'd', 'thing four', 'four' );
INSERT INTO t VALUES ( 'e', 'thing five', 'five' );
SELECT * FROM t;
```

We can see in SQLITE_MASTER that there is two indexes

Query 2:

type	name	tbl_name	rootpage	sql
table	t	t	2	CREATE TABLE t (code PRIMARY KEY, value TEXT, ycode UNIQUE)
index	sqlite_autoindex_t_1	t	3	NULL
index	sqlite_autoindex_t_2	t	4	NULL

Query 8:

code	value	ycode
a	thing one	one
b	thing two	two
c	thing three	three
d	thing four	four
e	thing five	five

Chapter 10\06. How to use INTEGER PRIMARY KEY function

```
CREATE TABLE t (id INTEGER PRIMARY KEY AUTOINCREMENT,a,b,c);
INSERT INTO t (a, b, c) VALUES ('a', 'b', 'c');
INSERT INTO t (a, b, c) VALUES ('a', 'b', 'c');
INSERT INTO t (a, b, c) VALUES ('a', 'b', 'c');
INSERT INTO t (a, b, c) VALUES ('a', 'b', 'c');
INSERT INTO t (a, b, c) VALUES ('a', 'b', 'c');
DELETE FROM t WHERE ID = 5;
INSERT INTO t (a, b, c) VALUES ('a', 'b', 'c');
SELECT * FROM t;
SELECT * FROM SQLITE_MASTER;
SELECT * FROM SQLITE_SEQUENCE;
```

Query 9:

id	a	b	c
1	a	b	c
2	a	b	c
3	a	b	c
4	a	b	c
6	a	b	c

Query 10:

type	name	tbl_name	rootpage	sql
table	t	t	2	CREATE TABLE t (id INTEGER PRIMARY KEY AUTOINCREMENT, a, b, c)
table	sqlite_sequence	sqlite_sequence	3	CREATE TABLE sqlite_sequence(name,seq)

Query 11:

name	seq
t	6

Chapter 11\01. Understanding transactions

Transaction:

- May be used to improve performance
- Is one or more operation that may modify the database
- Transaction written to the database when it's complete successfully
- When it can't be complete successfully a rollback will accour
- The database is locked when the transaction is in progress

Chapter 11\02. Using transactions in SQLite

```
BEGIN;
INSERT INTO sale (item_id,quantity,price)VALUES (4,12,1995);
UPDATE inventory SET quantity = ( SELECT quantity FROM
     inventory WHERE id = 4 ) - 12 WHERE id = 4;
COMMIT;
```

These two operations works as one unit (this is the meaning of transaction):

If, from some reason, one command will fail, the database will remain the same

BEGIN - for starting the transaction, COMMIT for execute all commands as one unit

In this lesson BW demonstrate that 1000 separate inserts action takes 1 second to enter to the database, but as a transaction it will take 40ms (because 1 write to the disk is much faster than 1000 writes)

Chapter 12\02. Creating a simple subselect

SQL:

```
DROP TABLE IF EXISTS t;
CREATE TABLE t ( a, b );
INSERT INTO t VALUES ( 'NY0123', 'US4567' );
INSERT INTO t VALUES ( 'AZ9437', 'GB1234' );
INSERT INTO t VALUES ( 'CA1279', 'FR5678' );
SELECT SUBSTR(b, 1, 2) AS Country, SUBSTR(b, 3) AS CoValue FROM t;

SELECT co.Name, tt.CoValue FROM (
   SELECT SUBSTR(a, 1, 2) AS State, SUBSTR(a, 3) AS StValue,
     SUBSTR(b, 1, 2) AS Country, SUBSTR(b, 3) AS CoValue FROM t
 ) AS tt
 JOIN Country AS co ON tt.Country = co.Code2;
```

Query 6:

Country	CoValue
US	4567
GB	1234
FR	5678

Query 7:

co.Name	tt.CoValue
United States	4567
United Kingdom	1234
France	5678

The SUBSELECT evaluate first. It create table name 'tt' with these fields:

State, StValues, Country, CoValue.

Then we join the table country (as 'co') on tt.country = co.code.

In the joined table we select co.Name and tt.CoValue

Chapter 12\03. Searching within a result set

```
DROP TABLE IF EXISTS t;
CREATE TABLE t ( a, b );
INSERT INTO t VALUES ( 'NY0123', 'US4567' );
INSERT INTO t VALUES ( 'AZ9437', 'GB1234' );
INSERT INTO t VALUES ( 'CA1279', 'FR5678' );
SELECT SUBSTR(a, 1, 2) AS State, SUBSTR(a, 3) AS StValue,
  SUBSTR(b, 1, 2) AS Country, SUBSTR(b, 3) AS CoValue FROM t;

SELECT co.Name as Country, ci.Name AS City FROM City AS ci
  JOIN Country AS co ON ci.CountryCode = co.Code
  WHERE co.Code2 IN (
    SELECT  SUBSTR(b, 1, 2)FROM t
  );
```

Query 6:

State	StValue	Country	CoValue
NY	0123	US	4567
AZ	9437	GB	1234
CA	1279	FR	5678

Query 7:

Country	City
United Kingdom	Aberdeen
United Kingdom	Basildon
United Kingdom	Belfast
United Kingdom	Birkenhead

Chapter 12\04. Searching within a joined result

```
SELECT artist, album, track, trackno,
  m || ':' || CASE WHEN s < 10 THEN '0' || s ELSE s END AS duration
  FROM (
    SELECT a.artist AS artist, a.title AS album, t.title AS track, t.track_number AS trackno,
        t.duration / 60 AS m, t.duration % 60 AS s
      FROM track AS t JOIN album AS a ON a.id = t.album_id
    ) ORDER BY artist, album, trackno;
```

Go

artist	album	track	trackno	duration
Frank Zappa	Apostrophe	Don't Eat the Yellow Snow	1	2:07
Frank Zappa	Apostrophe	Nanook Rubs It	2	4:38
Frank Zappa	Apostrophe	St. Alfonzo's Pancake Breakfast	3	1:50
Frank Zappa	Apostrophe	Father O'Blivion	4	2:18
Frank Zappa	Apostrophe	Cosmik Debris	5	4:14
Frank Zappa	Apostrophe	Excentrifugal Forz	6	1:33
Frank Zappa	Apostrophe	Apostrophe	7	5:50
Frank Zappa	Apostrophe	Uncle Remus	8	2:44
Frank Zappa	Apostrophe	Stink-Foot	9	6:33
Jimi Hendrix	Hendrix in the West	Johnny B. Goode	1	4:45
Jimi Hendrix	Hendrix in the West	Lover Man	2	3:05

Chapter 12\05. Creating a view

```
DROP TABLE IF EXISTS t;
CREATE TABLE t ( a, b );
INSERT INTO t VALUES ( 'NY0123', 'US4567' );
INSERT INTO t VALUES ( 'AZ9437', 'GB1234' );
INSERT INTO t VALUES ( 'CA1279', 'FR5678' );
DROP VIEW IF EXISTS unpackData;
CREATE VIEW unpackData AS
  SELECT SUBSTR(a, 1, 2) AS State, SUBSTR(a, 3) AS StValue,
    SUBSTR(b, 1, 2) AS Country, SUBSTR(b, 3) AS CoValue FROM t;
SELECT * FROM unpackData AS tt;
SELECT co.Name, tt.CoValue FROM unpackData AS tt
  JOIN Country AS co ON tt.Country = co.Code2;
```

Query 8:

State	StValue	Country	CoValue
NY	0123	US	4567
AZ	9437	GB	1234
CA	1279	FR	5678

Query 9:

co.Name	tt.CoValue
United States	4567
United Kingdom	1234
France	5678

Chapter 12\06. Searching within a joined view

One of the common use of views is to reduce the complexity of the sub-select statements.

The view will bring us the data without the need to run complicated statements

SQL:

```
DROP VIEW IF EXISTS JoinedAlbum;
CREATE VIEW JoinedAlbum AS
   SELECT a.artist AS artist, a.title AS album, t.title AS track, t.track_number AS trackno,
      t.duration / 60 AS m, t.duration % 60 AS s
    FROM track AS t JOIN album AS a ON a.id = t.album_id;

SELECT artist, album, track, trackno,
   m || ':' || CASE WHEN s < 10 THEN '0' || s ELSE s END AS duration
   FROM JoinedAlbum;
```

Go

Query 3:

artist	album	track	trackno	duration
Willie Nelson and Wynton Marsalis	Two Men with the Blues	Bright Lights Big City	1	5:20
Willie Nelson and Wynton Marsalis	Two Men with the Blues	Night Life	2	5:44
Willie Nelson and Wynton Marsalis	Two Men with the Blues	Basin Street Blues	5	4:56
Willie Nelson and Wynton Marsalis	Two Men with the Blues	Caldonia	3	3:25
Willie Nelson and Wynton Marsalis	Two Men with the Blues	Stardust	4	5:08
Willie Nelson and Wynton Marsalis	Two Men with the Blues	Georgia On My Mind	6	4:40

Chapter 13\02. Automatically updating a table with a trigger

Triggers are operation that automatically perform when a specific database event accoure

```
CREATE TABLE customer ( id INTEGER PRIMARY KEY, name TEXT, last_order_id INT );
CREATE TABLE sale ( id INTEGER PRIMARY KEY, item_id INT, customer_id INT, quan INT, price INT );
INSERT INTO customer (name) VALUES ('Bob');
INSERT INTO customer (name) VALUES ('Sally');
INSERT INTO customer (name) VALUES ('Fred');
SELECT * FROM customer;
```

Go

Query 6:

id	name	last_order_id
1	Bob	NULL
2	Sally	NULL
3	Fred	NULL

```
CREATE TRIGGER newsale AFTER INSERT ON sale
    BEGIN
        UPDATE customer SET last_order_id = NEW.id WHERE customer.id = NEW.customer_id;
    END;
SELECT * FROM sqlite_master;

INSERT INTO sale (item_id, customer_id, quan, price) VALUES (1, 3, 5, 1995);
INSERT INTO sale (item_id, customer_id, quan, price) VALUES (2, 2, 3, 1495);
INSERT INTO sale (item_id, customer_id, quan, price) VALUES (3, 1, 1, 2995);
SELECT * FROM sale;
SELECT * FROM customer;

DROP TRIGGER newsale;
```

Query 8:

type	name	tbl_name	rootpage	sql
table	customer	customer	2	CREATE TABLE customer (id INTEGER PRIMARY KEY, name TEXT, last_order_id INT)
table	sale	sale	3	CREATE TABLE sale (id INTEGER PRIMARY KEY, item_id INT, customer_id INT, quan INT, price INT)
trigger	newsale	sale	0	CREATE TRIGGER newsale AFTER INSERT ON sale BEGIN UPDATE customer SET last_order_id = NEW.id WHERE customer.id = NEW.customer_id; END

Query 12:

id	item_id	customer_id	quan	price
1	1	3	5	1995
2	2	2	3	1495
3	3	1	1	2995

Query 13:

id	name	last_order_id
1	Bob	3
2	Sally	2
3	Fred	1

NEW is a virtual table that contain the data from the event that trigger the trigger.

In this example NEW.id contain the id (the primary key value) of the insert to sale event.

Triggers can cause a headache to the developers and it's operation looks like a side effect

Chapter 13\03. Logging transactions with triggers

Triggers good for logging and auditing purpose (update log table with time of the insert).

```
CREATE TABLE customer ( id INTEGER PRIMARY KEY, name TEXT, last_order_id INT );
CREATE TABLE sale ( id INTEGER PRIMARY KEY, item_id INT, customer_id INT, quan INT, price INT );
CREATE TABLE triggerlog ( id INTEGER PRIMARY KEY, stamp TEXT, event TEXT, triggername TEXT, tablename TEXT, table_id INT);
INSERT INTO customer (name) VALUES ('Bob');
INSERT INTO customer (name) VALUES ('Sally');
INSERT INTO customer (name) VALUES ('Fred');
SELECT * FROM customer;
```

Go

Query 7:

id	name	last_order_id
1	Bob	NULL
2	Sally	NULL
3	Fred	NULL

```
CREATE TRIGGER newsale AFTER INSERT ON sale
    BEGIN
        UPDATE customer SET last_order_id = NEW.id WHERE customer.id = NEW.customer_id;
        INSERT INTO triggerlog (stamp, event, triggername, tablename, table_id)
            VALUES (DATETIME('now'), 'UPDATE last_order_id', 'newsale', 'customer', NEW.customer_id);
    END;
```

```
INSERT INTO sale (item_id, customer_id, quan, price) VALUES (1, 3, 5, 1995);
INSERT INTO sale (item_id, customer_id, quan, price) VALUES (2, 2, 3, 1495);
INSERT INTO sale (item_id, customer_id, quan, price) VALUES (3, 1, 1, 2995);
SELECT * FROM sale;
SELECT * FROM customer;
SELECT * FROM triggerlog;
```

Query 12:

id	item_id	customer_id	quan	price
1	1	3	5	1995
2	2	2	3	1495
3	3	1	1	2995

Query 13:

id	name	last_order_id
1	Bob	3
2	Sally	2
3	Fred	1

Query 14:

id	stamp	event	triggername	tablename	table_id
1	2012-12-08 04:49:47	UPDATE last_order_id	newsale	customer	3
2	2012-12-08 04:49:47	UPDATE last_order_id	newsale	customer	2
3	2012-12-08 04:49:47	UPDATE last_order_id	newsale	customer	1

Chapter 13\04. Improving performance with triggers

With triggers we can make reports (as new tables) to users that reduce traffic from busy tables:

```
CREATE TRIGGER newsale AFTER INSERT ON sale
    BEGIN
        UPDATE customer SET last_order_id = NEW.id WHERE customer.id = NEW.customer_id;
        INSERT INTO report (item, customer, quan, price)
            SELECT i.name, c.name, NEW.quan, NEW.price
                FROM item AS i
                JOIN customer AS c
                    ON c.id = NEW.customer_id
                WHERE i.id = NEW.item_id;
    END;
```

Chapter 13\05. Preventing unintended updates with triggers

Triggers can prevent updating tables that you want to prevent from changing

```
CREATE TABLE customer ( id integer primary key, name TEXT, last_order_id INT );
CREATE TABLE sale ( id integer primary key, item_id INT, customer_id INTEGER, quan INT, price INT,
    reconciled INT );
INSERT INTO customer (name) VALUES ('Bob');
INSERT INTO customer (name) VALUES ('Sally');
INSERT INTO customer (name) VALUES ('Fred');
INSERT INTO sale (item_id, customer_id, quan, price, reconciled) VALUES (1, 3, 5, 1995, 0);
INSERT INTO sale (item_id, customer_id, quan, price, reconciled) VALUES (2, 2, 3, 1495, 1);
INSERT INTO sale (item_id, customer_id, quan, price, reconciled) VALUES (3, 1, 1, 2995, 0);
```

```
CREATE TRIGGER update_sale BEFORE UPDATE ON sale
    BEGIN
        SELECT RAISE(ROLLBACK, 'cannot update table "sale"') FROM sale
            WHERE id = NEW.id AND reconciled = 1;
    END;
```

Chapter 13\06. Adding automatic time stamps

```
-- TIMESTAMPS
CREATE TABLE customer ( id integer primary key, name TEXT, last_order_id INT, stamp TEXT );
CREATE TABLE sale ( id integer primary key, item_id INT, customer_id INTEGER, quan INT, price INT, stamp TEXT );
CREATE TABLE log ( id integer primary key, stamp TEXT, event TEXT, username TEXT, tablename TEXT, table_id INT);
INSERT INTO customer (name) VALUES ('Bob');
INSERT INTO customer (name) VALUES ('Sally');
INSERT INTO customer (name) VALUES ('Fred');
SELECT * FROM customer;

CREATE TRIGGER newsale AFTER INSERT ON sale
    BEGIN
        UPDATE sale SET stamp = DATETIME('now') WHERE id = NEW.id;
        UPDATE customer SET last_order_id = NEW.id, stamp = DATETIME('now') WHERE customer.id = NEW.customer_id;
        INSERT INTO log (stamp, event, username, tablename, table_id)
            VALUES (DATETIME('now'), 'INSERT sale', 'TRIGGER', 'sale', NEW.id);
    END;

BEGIN;
INSERT INTO sale (item_id, customer_id, quan, price) VALUES (1, 3, 5, 1995);
INSERT INTO sale (item_id, customer_id, quan, price) VALUES (2, 2, 3, 1495);
INSERT INTO sale (item_id, customer_id, quan, price) VALUES (3, 1, 1, 2995);
COMMIT;

SELECT * FROM sale;
SELECT * FROM customer;
SELECT * FROM log;
```

Query 7:

id	name	last_order_id	stamp
1	Bob	NULL	NULL
2	Sally	NULL	NULL
3	Fred	NULL	NULL

Query 14:

id	item_id	customer_id	quan	price	stamp
1	1	3	5	1995	2012-12-08 05:16:52
2	2	2	3	1495	2012-12-08 05:16:52
3	3	1	1	2995	2012-12-08 05:16:52

Query 15:

id	name	last_order_id	stamp
1	Bob	3	2012-12-08 05:16:52
2	Sally	2	2012-12-08 05:16:52
3	Fred	1	2012-12-08 05:16:52

Query 16:

id	stamp	event	username	tablename	table_id
1	2012-12-08 05:16:52	INSERT sale	TRIGGER	sale	1
2	2012-12-08 05:16:52	INSERT sale	TRIGGER	sale	2
3	2012-12-08 05:16:52	INSERT sale	TRIGGER	sale	3

Chapter 14\01. Choosing PHP interface

There are 2 interfaces to use SQLite with PHP:

- PDO:
  - Works across platforms with many type of databases
  - Well maintained
  - Has rich suit of methods
  - Excellent performance
- SQLite3 interface:
  - A native interface to SQLite
  - Poor error handling

PDO is the recommended interface to work with .

Chapter 14\03. Using the PDO interface

```php
<?php

define('DATABASE', '/Users/bweinman/sqlite3_data/chap14.sqlite3');
main();

function main()
{
    global $G;
    try {
        $db = new PDO('sqlite:' . DATABASE);
        $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
        $db->exec('DROP TABLE IF EXISTS t');
        $db->exec('CREATE TABLE t (a, b, c)');
        message('Table t sucessfully created');
        $sth = $db->prepare('INSERT INTO t VALUES (?, ?, ?)');
        $sth->execute(array('a', 'b', 'c'));
        $sth->execute(array(1, 2, 3));
        $sth->execute(array('one', 'two', 'three'));
        $sth = $db->prepare('SELECT * FROM t');
        $sth->setFetchMode(PDO::FETCH_ASSOC);
        $sth->execute();
        foreach ( $sth as $row ) {
            message('%s, %s, %s', $row['a'], $row['b'], $row['c']);
        }
    } catch(PDOException $e) {
        error($e->getMessage());
    }

}
```

Remarks:

```php
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
```

Turn on PDO exceptions for the entire library (this will caught most pdo exceptions

```
$sth = $db->prepare('INSERT INTO t VALUES (?, ?, ?)');
$sth->execute(array('a', 'b', 'c'));
$sth->execute(array(1, 2, 3));
$sth->execute(array('one', 'two', 'three'));
```

First statements prepare a query that will use over and over...

The following statements will execute queries using $sth (statement handler)

```
$sth->setFetchMode(PDO::FETCH_ASSOC);
```

SetTetchMode to be associate array