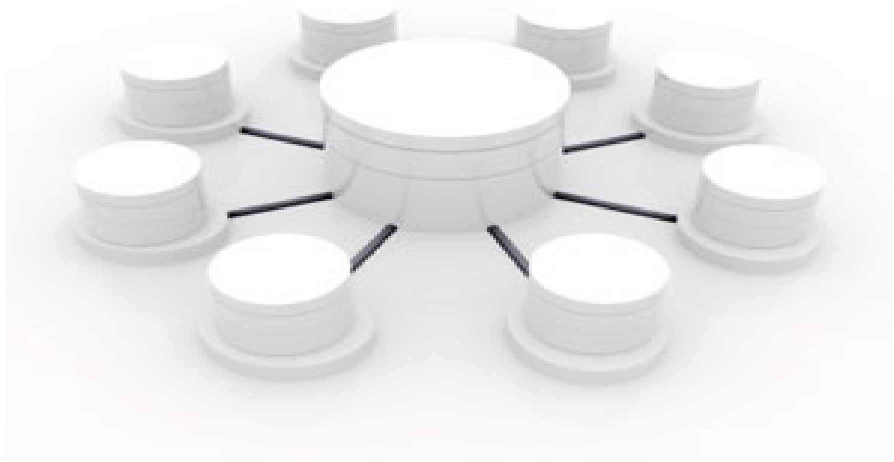
SQLite Using PHP & PDO

Reference



by Jan Zumwalt

Other references available at www.neatinfo.com/dev_notes/_cheat-sheets

Notes

Copyright © 2005-2015

Table of Contents

Section - 1 Introduction 6

About This Reference 6
What is PDO 6
SQLite2 VS SQLite3 6
What database does PDO support? 7
Setting up a development system 7
XAMPP config files 8
Database managers 8
Do I have PDO on my computer? 9
Good Programming Syntax 10
Comments 10
Single Row Data 10
Multiple Row Data 11

Section - 2 Data Types 12

Section - 3 Simple Database Example 13

Connect and error handling 13
Create and Insert Into Database 13
View and Search Data 14
Update Data 15
Delete Data 16

Section - 4 DATABASE (manipulation) 17

Add Database 17
Copy Database 17
Delete Database 18
Rename Database 18
Database Descriptions 19
Master Database 19
Command Line 19

Section - 5 TABLE (manipulation) 23

Add Table 23
Does Table Exist 23
Copy Table With Data 24
Copy Table Structure 25
Delete Table 26
Rename Table 26
View Table Structures 26
Table Description 27

Section - 6 ROW / RECORD (manipulation) 29

Insert Row (add) 29
Filling all fields 29
Selecting specific fields to fill 29
Insert - Multiple Rows 29
Copy Row 29
Delete All Rows 29
Delete Row(s) Matching Condition 29
Update Row(s) Matching Condition 30
SQL UPDATE Warning 31
Count Rows 31
Count Columns 31
Index Description 31

Copyright © 2005-2015

Section - 7 COLUMN / FIELD (manipulation) 33

Add Column 33
Copy Column and data 33
Delete Column 33
Rename Column 33
Count Columns 33

Section - 8 SEARCHES / SHOW (manipulation) 35

Simple Select Syntax 35
The four fetch types 35
WHERE *36*
SHOW RECORD ON SINGLE LINE *36*
LIKE *37*
AND *37*
OR *37*
Advanced Select Syntax 37
LIMIT *37*
ORDER BY *38*
GROUP BY *38*
HAVING *39*
DISTINCT *39*
Finding Duplicates *39*
DELETEING Duplicates *40*

Section - 9 Debug & Errors 41

Error Codes 41
1) If you copied code from a page of this reference 41
2) Some SQLite versions have incompatibilities 41
3) "Invalid resource" error 42
4) Sensitivity to white space 42
5) “unable to open file” 42
6) “SQLITE_CORRUPT error” 42
7) “WHERE Clause error” 42
8) Unexpected return value, i.e. “Array” 42
9) SYNTAX ERROR, Unexpected “End”… 43
10) “unexpected T_VARIABLE” 43

Section - 10 VIEWS (custom) 44

Section - 11 TRIGGERS (manipulation) 45

Insert Row Timestamp 45
Logging All Inserts, Updates, and Deletes 45

Section - 12 TRANSACTIONS 48

Section - 13 JOINS 49

CROSS JOIN 50
INNER JOIN 51
LEFT OUTER JOIN 51
RIGHT OUTER JOIN and FULL OUTER JOIN 52
UNION 52

Section - 14 Date and Time 53

Function Format 53
Modifiers 54
Timestamp 55
Examples 55
Save current or future date *56*
Less than time span *56*
Date equal to time span *56*
Time Zone 56

Copyright © 2005-2015

Errors and Bugs ... 57

Section – 15 Strings ... 58

List of String Functions ... 58
sqlite_escape_string ($string) ... 60
Text Scrubbing ... 60

Section – 16 Math ... 62

List of Math Functions ... 62
Math Examples ... 63
Money ... 63

Section - 17 Images - Saving and Displaying ... 65

Test Image ... 65
Saving an Image ... 66
Display an Image ... 66

Section - 18 Crypt, Hash, etc ... 69

Function Syntax ... 69
File IO ... 69

Section - 19 Printing Table Data ... 70

Formatted HTML Table ... 70
Table print ... 70
Simple print_r ... 71
Loop print_r ... 71
Var dump ... 72
Index & associative table output ... 73

Appendix - A Test Databases ... 75

Test database program ... 75

Appendix – B Command Reference ... 77

PDO Commands ... 77
SQLite3 Commands ... 78

Appendix – C Reserved Words (337) ... 80

Documenation ... 84
References ... 84

Copyright © 2005-2015

Section - 1
Introduction

About This Reference

This reference was written during the design of a project that did not need the huge overhead of a MySql database. When I researched SQLite, I found the existing examples very confusing. It was hard to tell the difference between code designed for the Unix or Win shell command line interface C, SQLite2, or SQLite3.

I decided to use **PDO** because I already had years of experience with **procedural** database programming and wanted to widen out with **object oriented** (OOP) database programming.

This reference has specific examples tested in the **XAMPP** development environment but should be able to be used on just about any **HTTP server** without modification.

SQLite3 is used for the database, **HTML** and **JavaScript** are used for the user browser interface, **PHP** is used for server file access and PHP **PDO** allows PHP to communicate with the SQLite database. Whew! That was a mouthful, but it all works easily and very well together.

The methodology just mentioned is by far the most common in use on the internet today. Thankfully, only basic computer skills and terminology will be needed. Examples are specifically designed for ease of understanding. Flashy cosmetic web design has been left out for clarity.

Since the original version of this guide, **Smart Phone** technology has embraced SQLIte. The **Android** phone OS is particularly suited to SQLite use. However, this guide is intended to be *lean and mean* so specific Smart Phone application is not shown.

Be sure to review the most common errors near the end of this document before proceeding.

What is PDO

PDO (PHP **D**ata **O**bjects**)** is a PHP extension that supports database connections with a **uniform command structure**. This allows developers to create code which is portable across many databases and system operating platforms. Simply explained, PDO lets you use the same code (object oriented) or commands for a project that could be used on a cell phone, Apple computer, Windows, or an IBM mainframe. That makes programming a lot easier!

SQLite2 VS SQLite3

Now the *BAD* news!

The volunteers that maintain **SQLite** wanted a small lightweight feature rich database that would compare to MySql. SQLite takes less than 1mb of disk storage space compared to 80-100mb for MySql. To keep SQLite *lean and mean* the developers chose to **forfeit complicated multitasking and multiuser** criteria that loaded down MySql. This is not a problem, **it is a feature**. At some point, some of the original ways of doing things in earlier versions of SQLite became such a problem that they decided to create an entirely new command and database structure between version 2 & 3.

The largest changes started with SQLite3 and that's a *BIG* problem. The changes in SQLite3 cause much of the source code of earlier versions to be **incompatibility with newer versions of SQLIte**; neither the database or command structures work smoothly across these versions.

The solution chosen by SQLite volunteers was to start **SQLite3 as a "new" DB language**; hence we ended up with PDO. As a consequence of these differences, we must use caution in choosing the commands and database structure. The good news is the commands are easily recognized; **SQLite3 commands use "=>" object syntax**.

Copyright © 2005-2015

What is not always easy to recognize is the database structure (type). Many SQLIte databases have no extension or generic extensions that do not offer any clue as to what set of commands should be used on them, so ***be careful!***

SQLite (like most other DB languages) offers a set of command line options. The **command line** and **PDO** commands are **not interchangeable** so watch out!

Here is a comparison of commands to open or create a database.

```
Default  SQLite2:          $db = new SQLiteDatabase('test.sqlite2', 0666, $error);
PDO SQLite2:               $db = new PDO("sqlite2:test.sqlite2");
Command line SQLite2:      sqlite   test. Sqlite2
Command line SQLite3:      sqlite3  test. sqlite3
PDO SQLite3:               $db = new PDO("sqlite:test.sqlite3");
```

Listing 1.1 – Version command comparison.

What database does PDO support?

PDO supports many of the popular databases. Here is a partial list...

- DBLIB: FreeTDS / Microsoft SQL Server / Sybase
- Firebird (http://firebird.sourceforge.net/): Firebird/Interbase 6
- IBM (IBM DB2)
- INFORMIX - IBM Informix Dynamic Server
- MYSQL (http://www.mysql.com/): MySQL 3.x/4.0
- OCI (http://www.oracle.com): Oracle Call Interface
- ODBC: ODBC v3 (IBM DB2 and unixODBC)
- PGSQL (http://www.postgresql.org/): PostgreSQL
- SQLITE (http://sqlite.org/): SQLite 3.x

Setting up a development system

You will need a development environment. I strongly recommend using **XAMPP** for either **Windows** or **Linux**. Prior to about 2007, WAMP had more features and less installation problems than XAMPP, but that is no longer true. I have used both **XAMPP** and **WAMP** for many years and find that XAMPP has less installation problems. A few years ago XAMPP had some control panel issues which seem to have all been fixed. For downloads and further reading, see the links below.

```
http://www.apachefriends.org/en/xampp.html
http://www.wampserver.com/en/
```

Most (but not all) **Linux** development systems come with PHP support for SQLite. On windows machines, XAMPP installs PHP support for SQLite2 ***but not necessarily SQLite3.***

If you have trouble with **SQLite3**, the first thing you should check is the **PHP config file**. Don't forget to check all the possible locations given below. Also, you should ***stop*** the **Apache** server before editing any file that affects Apache, and then ***restart*** after you are done.

```
Windows:   xampp/php/php.ini.
Some Linux xampp/apache/bin/php.ini
```

The **Windows config file** should have a couple areas with lines that look like the following…

```
extension=php_pdo_sqlite.dll
extension=php_pdo_sqlite_external.dll
extension=php_sqlite.dll
extension=php_sqlite3.dll
sqlite.assoc_case = 0
```

These lines should **not have the ;** (semi-colon) at the start of the line. The semicolon comments out the code and prevents it from working. If it has the semi-colon **;** it should be **edited out**. Linux may have slightly different values and file names that **end with .so** on the end.

Copyright © 2005-2015

XAMPP config files

Just to make this section complete, a list of configuration files for XAMPP are provide below. As we already mentioned, there may be slight differences in location or names depending on whether it is a Windows or Linux installation. Many Linux developers slightly modify the directory structure.

```
    * Apache basic configuration: .\xampp\apache\conf\httpd.conf
    * Apache SSL:                 .\xampp\apache\conf\ssl.conf
    * Apache Perl   (addon):      .\xampp\apache\conf\perl.conf
    * Apache Tomcat (addon):      .\xampp\apache\conf\java.conf
    * Apache Python (addon):      .\xampp\apache\conf\python.conf
    * PHP:                        .\xampp\php\php.ini
                                  .\xampp\apache\bin\php.ini
    * MySQL:                      .\xampp\mysql\bin\my.cnf
    * phpMyAdmin:                 .\xampp\phpMyAdmin\config.inc.php
    * FileZilla FTP:              .\xampp\FileZillaFTP\FileZilla Server.xml
    * Mercury Mail:               .\xampp\MercuryMail \MERCURY.INI
    * Sendmail:                   .\xampp\sendmail\sendmail.ini
```

Linux may have slightly different values such as names with ".so" on the end.

Database managers

Custom programming is probably only about 10-20% of a database programmer's work. The other 80% is done using a convenient database ***manager*** program. Several good ones are listed below.

```
SQLite2009 Pro Enterprise Manager http://osenxpsuite.net/
Firefox SQLITE plugin module            http://code.google.com/p/sqlite-manager/
SQLIite Administrator                   http://sqliteadmin.orbmu2k.de/
SQLite Browser                          http://sqlitebrowser.sourceforge.net/ great for beginners
SqliteRoot                        http://sqliteroot.com/
Phpliteadmin                            http://code.google.com/p/phpliteadmin/
```

At the time of this writing, **https://sourceforge.net** had over 500 high quality **free SqLite utilities**. The Google code developer site **http://code.google.com** is another good place to look. They are all worth checking out. An internet search will certainly turn up more treasure. For example, there are several utilities that make managing a database over the internet much easier.

PhpLiteAdmin does not require installation or configuration. It is a small single file that is placed in the directory where your database is located. **I recommend this for the beginner** and it is what I use most of the time. It supports both SqLite2 and SqLite3 (several other managers don't). It has a simple top menu arranged and labeled intuitively.
https://bitbucket.org/phpliteadmin/public/

SQLite2009 Pro I use (freeware version) some of the time. **SQLite2009 Pro** has all the features you could possibly want including built-in documentation. It takes some getting use to; especially figuring out your data hierarchy and where your table information is located. The installation creates a SQLite file extension type that opens the manager immediately if the mouse clicks on a database file.
http://sqlite2009pro.azurewebsites.net/

SQLite Administrator is one of the easiest managers to use and I would prefer it except that it is missing a couple key features. It is not easy to add or edit a record. I would not be surprised if a year or more of development fixes most of its issues. The installation creates a SQLite file extension type that opens the manager immediately if the mouse clicks on a database file.
http://sqliteadmin.orbmu2k.de/

Firefox Plug-in would be a superior choice except that the Firefox browser must be running before you select a database file to work on. With the browser running, you must select a menu then a tool button, afterwards you can close the

Copyright © 2005-2015

Firefox browser and the manager will remain independent. This is a bit clumsy and prevents an association of the SQLite file extension to be made that attaches to the manager tool. That being said, it is easy to use and is well laid out.

SQLite Browser is a very clean easy to use manager. It is intuitive and things are extremely easy to find. The user has the option of selecting from a set of wizards and it provides a pleasing spreadsheet-like interface. Windows allows you to create a SQLite file extension so that a mouse selection opens the manager immediately.
http://sqlitebrowser.org/

Do I have PDO on my computer?

Once you have a development environment up and running, you should check if the PHP PDO driver is installed for your database. Check **phpinfo()** for a section named **"SQLITE"** and **"PDO".** You may also check the available drivers with the static method **PDO::getAvailableDrivers()**, examples are given below

```
<?php
  echo '<h2>PHP Info</h2>';
  phpinfo();
?>
```

Listing: 1.2 - All the PHP info you could possibly want!

```
<?php
echo '<h2>PDO Info</h2>';

foreach(PDO::getAvailableDrivers() as $driver) {
  echo $driver.'<br />';
}
?>
```

Listing: 1.3 – Show a list of PDO supported databases on your computer.

```
<html>
<body style="font-family:arial;">
<h2>PDO & PHP Info</h2>
</blockquote>
<span style="color:red;">
<?php
   foreach(PDO::getAvailableDrivers() as $driver)  {
     echo $driver.'<br />';
   }
   ?>
</span>
</blockquote>
If you see "<span style="color:red;">sqlite2</span>" listed, there is support for sqlite2.<br><br>
If you see "<span style="color:red;">sqlite</span>" listed, there is support for sqlite3.<br><br>

PHP version: <span style="color:red;"><?php echo phpversion(); ?></span><br>
SQLite library version: <span style="color:red;"><?php echo sqlite_libversion(); ?></span><br>
SQLite library character encoding: <span style="color:red;"><?php echo sqlite_libencoding();
?></span><br>
<?php
echo "If you don't see an error message,  a <span style='color:red;'>SQLite3</span> test database
was created.<br><br>";

#  create test SQLite3 database
$query = " test.sqlite3";
$db = new PDO("sqlite:$query") or die("Could not create:  <span style='color:red;'>$query</span>");
?>
```


Copyright © 2005-2015

```
<!--  uncomment the next line to see the comprehensive PHP information  -->
<!--  PhpInfo: <span style="color:red;"><?php echo phpinfo(); ?></span><br><br>  -->

</body>
</html>
```

Listing: 1.4 – Fancy report with an option for a comprehensive PHP listing.

```
PHP and SQLite3 Information
Supported PHP Data Object (PDO) drivers.
mysql
odbc
sqlite
sqlite2
Note: If you see "sqlite", then there is support for sqlite3.

PHP version: 5.3.1
SQLite library version: 2.8.17
SQLite library character encoding: iso8859
If you don't see an error message, a SQLite3 test database was created.
```

OUTPUT – View generated on a Win XP XAMPP system. (The php info is not shown and is about 3 pages long).

Good Programming Syntax

A frequently used standard is to assign a variable named $row for single row data retrieval and $rows (notice the 's') for multiple row data. Multiple rows of data are sometimes called a data set or result set.

COMMENTS

Single row or end of row comments start with a double dash "- -". Multiline comments use the typical "C" style comment, /* ...text... */.

```
$query =  "CREATE TABLE salespeople (
  -- single row comment
  /*
    Comment that spans
    More than one line */
  id INTEGER PRIMARY KEY,
  first_name TEXT NOT NULL, -- end of row comment
  last_name TEXT NOT NULL  /* you can also do this */
  ) ";
```

SINGLE ROW DATA

A single row of data is retrieved by using the **fetch()** statement. Be careful not to confuse it with the **fetchall()** statement. The following is a general example.

```
$query = "SELECT * FROM t1";
$result = $db->query("$query") or die("Error in query:  <span style='color:red;'>$query</span>");
$row = $result->fetch();
```

Listing: 1.5 – Getting a single row of information as an array.

Even though there may be more than one match, only the first match one will be returned in $row.

We can access this single row of data by using the associative field name.

```
echo $row['name'];
```

Listing: 1.6 – Showing a column from row data.

This would return "Frank" because that has the lowest ID number and would be the first match.

MULTIPLE ROW DATA

Multiple rows of data can be retrieved as an array by using the **fetchall()** statement. Be careful not to confuse it with the fetch() statement. The result is returned as an array of arrays. The following is a general example.

```
$query = "SELECT * FROM t1";
$result = $db->query("$query") or die("Error in query:  <span style='color:red;'>$query</span>");
$rows = $result->fetchall();
```

Listing: 1.7 – Getting multiple rows of information as an array of arrays.

We can access any row using array matrix addressing.

```
echo $rows['3'],['name'];
```

Listing: 1.8 – Showing any column of an array of arrays.

This would access the third row of information and return the "name" data.

End Of Section

Copyright © 2005-2015

Section - 2
Data Types

Each column in a SQLite3 database is assigned one of the following type affinities:

- BLOB
- INTEGER
- NULL
- NUMERIC
- REAL
- TEXT

Typename from CREATE TABLE statement	Resulting Assignment	Description
BLOB	BLOB	A Binary Long Object, i.e. images
INT INTEGER TINYINT SMALLINT MEDIUMINT BIGINT UNSIGNED BIG INT INT2 INT8	INTEGER	A number without decimals, a whole number.
NULL	NULL	Unassigned
NUMERIC DECIMAL(10,5) BOOLEAN DATE DATETIME	NUMERIC	A number with or without decimals
REAL DOUBLE DOUBLE PRECISION FLOAT	REAL	A 'floating point' number, a number with decimals
CHARACTER(20) VARCHAR(255) VARYING CHARACTER(255) NCHAR(55) NATIVE CHARACTER(70) NVARCHAR(100) TEXT CLOB	TEXT	Text that can contain both numbers and letters

---

End Of Section

Copyright © 2005-2015

Section - 3
Simple Database Example

This section provides simple examples of most database commands used by PDO. The example provides the skeletal framework on which many databases, including a company employee databases or phone contact list can be made. It is a very valuable reference on its own. The steps to creating a simple database are…

```
Design
Create db File
Create table and column definitions
Insert data
View and search data
Update data
Delete data
```

Connect and error handling

In all our examples, we assume that we have successfully created and/or connected to the database and set up our error handling. A good generic template is given below

```
/*  Listing 2.1 - create "people" db  */
   echo '<body style="font-family:arial;">';
   echo '<h2>Create "people" database</h2>';

#  setup
   $dbPath  = $_SERVER['DOCUMENT_ROOT'] . "/contacts/";
   $dbFile  = "people.sqlite3";

#   connect to SQLite3 database
   $query = "$dbPath$dbFile";
   try { $db = new PDO("sqlite:$query"); }
   catch(PDOException $e) { echo $e->getMessage()." Error: <span style='color:red;'>$query</span>";
}

#  enhanced error messages
   $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
```

Listing: 2.1 – Generic connection and error handling.

From this point on, we will always assume the programmer knows that the necessary code to connect to the database (similar to what is shown here) needs to be inserted for the db connection.

Create and Insert Into Database

Once the error definition is made, we don't need to use the "try{ }", "catch{ }", or "die" code anymore. The PDO error reporting does a wonderful job on fatal errors, but can be a little cryptic for warnings. I have found a combination of the two techniques works best. An example of typical error support that includes sqlite specific information is provided here.

```
   $result = $db->query("$query") or die("Error in query: <span style='color:red;'>$query</span>");
```

Listing: 2.2 – Typical error support.

```
<?php
/*  Listing 2.3 - create "people" db  */
   echo '<body style="font-family:arial;">';
   echo '<h2>Create "people" database</h2>';

#  connect to database
# ... code to setup, connect to database, and handle errors goes here
```


Copyright © 2005-2015

```
#  create table
   $query = "CREATE TABLE friends (
            id INTEGER PRIMARY KEY NOT NULL,
            active  TEXT(1),  -- y/n
            updated
            name  TEXT(25),
            phone TEXT(25),
            editby TEXT default 'unknown',
            editdate TEXT default (datetime('now','localtime')
         )";
   $result = $db->query("$query") or die("Error in query: <span style='color:red;'>$query</span>");

   echo "<i>Database successfully initialized.<br></i> ";

# multiple row data insert
   $query = "insert into 'friends' (name, phone)
         select 'James Bond','007-1234'
   union select 'Mickey Mouse','(200) 100-1122'
   union select 'Daffy Duck','200 100-1122' ";

   $result = $db->query("$query") or die("Error in query: <span style='color:red;'>$query</span>");

# single row data insert
   $query = "insert into 'friends' (name, phone) values ('Porky Pig', '200) 100-1122') ";

   $result = $db->query("$query") or die("Error in query: <span style='color:red;'>$query</span>");

#  show success
   echo "<i>Data successfully inserted.</i><br>";

#  show number of rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected ";

#  close db, PHP will also do this automatically
   $db = NULL;
?>
```

Listing: 2.3 – create and insert data

View and Search Data

Here is our completed code to view all table data.

```
<?php
/*  Listing 2.4 - search techniques  */
   echo '<body style="font-family:arial;">';
   echo '<h2> Search Techniques </h2>';

#  connect to database
# ... code to setup, connect to database, and handle errors goes here

#   query using simple SELECT
    $query = "SELECT * FROM 'friends'";
    $result = $db->query($query) or die("Error in query: <span style='color:red;'>$query</span>");
    $row = $result->fetchall(PDO::FETCH_ASSOC);

#   show table data
```


Copyright © 2005-2015

```
    foreach($row as $array) {
    echo "<pre>";
    print_r($array);
    echo "</pre>";
    }
   echo 'Result for: $query = "SELECT * FROM \'friends\' ";'. "<br>";

 #   show number of rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected<br><br>";

   echo "------<br>";

#   query using specific fields and SELECT with WHERE
    $query = "SELECT 'phone' FROM 'friends' WHERE name='James%' ";
    $result = $db->query($query) or die("Error in query: <span style='color:red;'>$query</span>");
    $row = $result->fetchall(PDO::FETCH_ASSOC);

#   show table data
    foreach($row as $array) {
    echo "<pre>";
    print_r($array);
    echo "</pre>";
    }
   echo 'Result for: $query = "SELECT phone FROM 'friends' WHERE name=\'James%\' ";. "<br>";

#   show number of rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected<br><br>";

#  show success
   echo "<i>Search successful.</i><br>";

#   close db, PHP will also do this  automatically
   $db = NULL;
?>
```

Listing 2.4 – search techniques

Update Data

If we use **WHERE name LIKE 'James%'** then any name starting with "James" will be selected.

So here is what we need

```
<?php
/* Listing 2.5.- updating James  */

#  connect to database
# ... code to setup, connect to database, and handle errors goes here

#  update table data
   $query = "UPDATE 'friends' SET phone='911-007-1234' WHERE name LIKE 'James%'";
   $result = $db->query("$query") or die("Error in query: <span style='color:red;'>$query</span>");

#  show success
   echo "<i>Update data was successful.</i><br>";
```


Copyright © 2005-2015

```
#   Return rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected ";

#   close db, PHP will also do this  automatically
   $db = NULL;
?>
```

Listing: 2.5 – final version of updating a record

Delete Data

If we use **WHERE name LIKE 'James%'** then any name starting with **James** will be selected. Instead we will use **WHERE name LIKE '%Bond%'** then only a name with this last name would be deleted.

So here is what we need

```
<?php
/* Listing 2.6 - delete James Bond  */
   echo '<body style="font-family:arial;">';
   echo '<h2>Update "friends" table data</h2>';

#  connect to database
# ... code to setup, connect to database, and handle errors goes here

#  delete row ofdata
   $query = "DELETE from 'friends' WHERE name LIKE '%Bond%'";
   $result = $db->query("$query") or die("Error in query: <span style='color:red;'>$query</span>");

#  show success
   echo "<i>Data deleted successful.</i><br>";

#   Return rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected ";

#   close db, PHP will also do this  automatically
   $db = NULL;
?>
```

Listing: 2.6 – delete a row of information based on search criteria

---

End Of Section

Copyright © 2005-2015

Section - 4
DATABASE (manipulation)

An often over looked aspect of database programming is the maintenance that usually continues to be an ongoing after the database is created. Commands to manipulate the whole database file or reveal the definitions of its tables or columns are necessary to do this.

Add Database

Remember that if SQLite does not find the database given in the file name, it will attempt to make it. So all that is needed to create a new db is to have a valid path and supply a new file name.

If you do not want PDO to create an empty db when none is found, you can use php's **file_exists** command to test for the file.

```
$datafile = '/path/to/test.sqlite3';

if (file_exists($datafile)) {
  ... code to
  ... connect to database
} else {
  echo "Error, did not find database at $datafile.";
  exit;
}
```

Listing: 3.1 – Testing if a database exists

He is a program listing that opens a db or creates a new one if none is found.

```
<?php
/* Listing 8.1 - add db  */
   echo '<body style="font-family:arial;">';
   echo '<h2>Add or connect to database</h2>';

#  setup
   $dbPath  = $_SERVER['DOCUMENT_ROOT'] . "/path/to/db/";
   $dbFile  = "db_name.sqlite3";

#  make or connect to SQLite3 database
   $query = "$dbPath$dbFile";
   try { $db = new PDO("sqlite:$query"); }
   catch(PDOException $e) { echo $e->getMessage(); }

   echo "<i>Database successfully initialized.<br></i> ";

#  close db, PHP will also do this  automatically
   $db = NULL;
?>
```

Listing: 3.2 - Creating a new database

Copy Database

At the time of writing this reference, I was unable to find a PDO db copy command. However, there are still several options available.

Copyright © 2005-2015

1- all the db managers provide an “import” feature
2- a database could be opened and it contents meticulously copied
3- use php to make a copy of the file

I favor method three (3). Here is an example…

```
<?php
/* Listing 8.2 - copy a db  */
   echo '<body style="font-family:arial;">';
   echo '<h2>Copy database</h2>';

#  setup
   $dbPath  = $_SERVER['DOCUMENT_ROOT'] . "/contacts/";
   $dbFile    = "people.sqlite3";
   $dbNew   = "test.sqlite3";

#  copy SQLite database
   copy("$dbPath$dbFile", "$dbPath$dbNew")
   or die("Error copying db: <span style='color:red;'>$dbFile to $dbNew</span>");

#  show success
   echo "<i>Database copied successfully.</i><br>";
?>
```

Listing: 3.3 – Copy a database

Delete Database

At the time of writing this reference, I was unable to find a PDO db delete or drop command. However, there are still several options available.

1- db managers may provide a “delete” feature
2- a database could be opened and it contents meticulously deleted
3- use php to delete the database file.

I favor method three (3). Here is an example…

```
<?php
/* Listing 8.3 - delete a db  */
   echo '<body style="font-family:arial;">';
   echo '<h2>Delete database</h2>';

#  setup
   $dbPath  = $_SERVER['DOCUMENT_ROOT'] . "/contacts/";
   $dbFile    = "test.sqlite3";

# delete file (database)
   unlink("$dbPath$dbFile")
   or die("Error deleting dataebase: <span style='color:red;'>$dbPath$dbFile</span>");

#  show success
   echo "<i>Database deleted successfully.</i><br>";
?>
```

Listing: 3.4 – Delete a database

Rename Database

Copyright © 2005-2015

At the time of writing this reference, I was unable to find a PDO db rename command. However, there are still several options available.

1- db managers may provide a “rename” feature
2- a database could be opened and it contents copied to a new database, then the old db deleted
3- use php to rename the database file

I favor method three (3). Here is an example…

```
<?php
/* Listing 8.4 - rename db  */
   echo '<body style="font-family:arial;">';
   echo '<h2>Rename database</h2>';

   $dbPath  = $_SERVER['DOCUMENT_ROOT'] . "/contacts/";
   $dbFile    = "people.sqlite3";
   $dbNew   = "test.sqlite3";

#  copy SQLite database
   rename("$dbPath$dbFile", "$dbPath$dbNew")
   or die("Error renaming db: <span style='color:red;'>$dbFile to $dbNew</span>");

#  show success
   echo "<i>Database renamed successfully.</i><br>";
?>
```

Listing: 3.5 – Delete a database

Database Descriptions

Master Database

Every SQLite database has an SQLITE_MASTER table that defines the schema for the database. The SQLITE_MASTER table is read-only. You cannot change this table using UPDATE, INSERT, or DELETE. The table is automatically updated when you use CREATE TABLE, CREATE INDEX, DROP TABLE, and DROP INDEX commands.

One thing that is particularly useful about using this command is that we are given the SQL query needed to create the db structure. Very nice indeed! For automatically created indices (used to implement the PRIMARY KEY or UNIQUE constraints) the sql field is NULL.

COMMAND LINE

If you are running the sqlite3 command-line program you can get a host of information using the **.dump** or **.schema** to see the complete database schema including all tables and indices tables to get a list of all tables. Typing **.tables** will provide a complete list of tables. These commands can be followed by a LIKE pattern that will restrict the tables that are displayed.

The SQLITE_MASTER table looks like this:

```
CREATE TABLE sqlite_master (
  type TEXT,
  name TEXT,
  tbl_name TEXT,
  rootpage INTEGER,
  sql TEXT
);
```

To view the database structure, we need to connect to the database then query the master table. To View all the available information we would use the following select statement;

```
  $query = "SELECT * FROM sqlite_master";
```


Copyright © 2005-2015

```
$result = $db->query("$query") or die("Error in query:  <span style='color:red;'>$query</span>");
$rows = $result->fetchAll();
echo"<pre>"; print_r($rows); echo"<pre>"; # show data
```

Listing: 3.6 – View master database descriptor information

```
Array
(
    [type] => table
    [name] => friends
    [tbl_name] => friends
    [rootpage] => 2
    [sql] => CREATE TABLE friends (
            id INTEGER PRIMARY KEY,
            name  TEXT(25),
            phone TEXT(25)
         )
)
```

Example output

To see both the master and temp_master

```
$query = "SELECT * FROM sqlite_master
  UNION SELECT * FROM sqlite_temp_master ";
```

Listing: 3.7 – master and temp_master data descriptors

To see just table descriptions use this….

```
$query = "SELECT sql FROM sqlite_master WHERE type='table'";
$result = $db->query("$query") or die("Error in query:  <span style='color:red;'>$query</span>");
$rows = $result->fetchall(PDO::FETCH_ASSOC);
echo"<pre>"; print_r($rows); echo"<pre>"; # show data
```

Listing: 3.8 – All table data descriptor information

To see a specific table's description use this….

```
$result = $db->query("SELECT * FROM sqlite_master WHERE type='table' AND name='t1'");
$rows = $result->fetchall(PDO::FETCH_ASSOC);
echo "</pre>";  echo"<pre>"; print_r($rows); echo"<pre>"; # show data
```

Listing: 3.9 – Specific table descriptor information

Multiple Databases

The "ATTACH DATABASE" statement connects one or more database files to the current database connection.

For the example below, suppose we want the current and last year's sales total for one of the employees. Each years data is arranged by employee name. We will use two db files arranged like this…

Copyright © 2005-2015

File: db1.sqlite3	
Table: sales	
Col1 = smith	Other employees…
50	more data
75	more data
more data	more data

File: db2.sqlite3	
Table: sales	
Col1 = smith	Other employees…
200	more data
NULL	more data
more data	

```
<?php
  echo '<body style="font-family:arial;">';
  echo '<h2>Use Multiple Database</h2>';
  //  uses db1 & db2 as test databases
  //  note that "main" is a reserved word for referancing the primary db
  //  "SELECT * FROM t1" is the same as "SELECT main.* FROM t1"

  // connect to SQLite3 database
  $db = new PDO("sqlite:db1.sqlite3");

  // enhanced error messages
  $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  // query 1st database
  $result = $db->query("SELECT smith FROM 'sales' WHERE 1");
  $rows   = $result->fetchall(PDO::FETCH_ASSOC);
  echo "<pre>"; print_r($rows); echo "</pre>";

  // access second database, ATTACH needs the FULL path
  $result = $db->query("ATTACH database '$_SERVER[DOCUMENT_ROOT]/astrohex_parts/db2.sqlite3' AS
lastyear");

  // query 2nd database
  $result = $db->query('SELECT * FROM lastyear.sales');
  $rows   = $result->fetchall(PDO::FETCH_ASSOC);
  echo "<pre>"; print_r($rows); echo "</pre>";

  // add columns from both databases, "as" is required
  $result = $db->query('SELECT  SUM(Total) -- SUM pulls total values
    FROM (
      SELECT SUM(smith) as Total FROM main.sales  -- "as" is required so sum can pull values
      UNION
      SELECT SUM(smith) as Total FROM lastyear.sales
    )');

  $rows   = $result->fetchall(PDO::FETCH_ASSOC);
  echo "<pre>"; print_r($rows); echo "</pre>";
?>
```

Result

```
Database 1
Array (
  [0] => Array
    ([smith] => 50.0)
```


Copyright © 2005-2015

```
  [1] => Array
    ([smith] => 75.0)
)

Database 2
Array(
  [0] => Array
    ([smith] => 200.0)
)

Total
Array(
  [0] => Array
    ([SUM(Total)] => 325.0)
)
```

End Of Section

Copyright © 2005-2015

Section - 5
TABLE (manipulation)

Add Table

We already learned how to add a table to a db. So that our reference is complete, we provide the code again. Don't forget that the db "connect" process will create a new db if it does not already exist. Therefore, adding a table to a non existent db will usually result in the db and table being created even if it does not exist.

```
<?php
/* Listing 9.1 - Add table to db */
   echo '<body style="font-family:arial;">';echo '<h2>Add Table</h2>';

  ... code to connect to database, and handle errors

#  create table
   $query = "CREATE TABLE test (
            id        INTEGER PRIMARY KEY,
            editdate  TEXT DEFAULT (datetime('now','localtime')),
            col1name  TEXT(25),
            col2name  TEXT(25)
         )";
  $result = $db->query("$query") or die(" Error adding table: <span
style='color:red;'>$query</span>");

#  show success
   echo "<i>Table added successfully.</i><br> ";

#  Return rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected ";

#  close db, PHP will also do this  automatically
   $db = NULL;
?>
```

Listing: 4.1 – Add table to a database

```
Add Table
Table added successfully.
0 records affected
```

OUTPUT – for listing 4.1

Does Table Exist

If we need to check and see if the table exists, here is an enhanced version.

```
<?php
/* Listing 8.2 - Check if table exists */
   echo '<body style="font-family:arial;">';
   echo '<h2>Check if table exist</h2>';

  ... code to connect to database, and handle errors

# check master table for table name
   $query = "SELECT name FROM sqlite_master WHERE type='table' and name='table_to_make'";
```


Copyright © 2005-2015

```
   $db->query("$query") or die(" Error adding table: <span style='color:red;'>$query</span>");
   $totaltables = sqlite_num_rows($query);

# does table exist
   if ($totaltables<1) {
   #  table does not exist, create table
   $query = "CREATE TABLE test (
            id INTEGER PRIMARY KEY,
            col1name  TEXT(25),
            col2name  TEXT(25)  )";
   }
   else{
   # table exists...don't create again , do something else
   }
   . . . program continues
```

Listing: 4.2 – Check if table exists

Copy Table With Data

SQLite ALTER TABLE does not support “copy table”. However, there are still several options available.

1- db managers may provide this feature
2- table data could be exported to file then imported under a new table name
3- a database could be opened and it’s table contents copied
4- we can use a combination of CREATE and SELECT to duplicate a table

I favor method four (4). We can trick the database into creating a new table filled with the contents of an old table with the following command…

```
   $result = $db->query("CREATE TABLE '$newtable' AS SELECT * FROM '$oldtable'");
```

Listing: 4.3 – Copy tables

Here is a complete test program.

```
<?php
/* Listing 9.3 - Copy table with data */
   echo '<body style="font-family:arial;">';
   echo '<h2>Copy table with data</h2>';

#   setup
   $dbPath   = $_SERVER['DOCUMENT_ROOT'] . "/contacts/";
   $dbFile   = "sample.sqlite3";
   $oldtable = "test";
   $newtable = "newtbl";

  ... code to connect to database, and handle errors

   $result = $db->query("CREATE TABLE '$newtable' AS SELECT * FROM '$oldtable'");
   $rows = $result->fetchAll();

#  show table data
   $query = "SELECT * FROM '$newtable'";
   $result = $db->query("$query") or die("Error in query:  <span
style='color:red;'>$query</span>");
   $rows = $result->fetchall(PDO::FETCH_ASSOC);

#  show new table
```


Copyright © 2005-2015

```
   foreach($rows as $array) {
      echo "<pre>";
      print_r($array);
      echo "</pre>";
   }
?>
```

Listing 4.4 - Copy table and data to new table

Copy Table Structure

To get the table structure we can use the following code.

```
<?php
/* Listing 9.4 – Copy table  */
echo '<body style="font-family:arial;">';
echo '<h2>Copy table structure</h2>';

#   setup
$dbPath   = $_SERVER['DOCUMENT_ROOT'] . "/contacts/";
$dbFile   = "sample.sqlite3";
$oldtable = "test";
$newtable = "newtbl";

  ... code to connect to database, and handle errors

#   get table structure
$result = $db->query("SELECT * FROM sqlite_master WHERE type='table' and name='$oldtable'");
$row = $result->fetch(SQLITE_ASSOC);
$structure = $row['sql'];
...
Listing: 9.4 – Get table structure from MASTER TABLE
Next we substitute the old table name with the new table name in the structure statement using PHP
preg_replace.
#   replace old table name with new table name in the structure statement
$search  = "/CREATE TABLE $oldtable/";
$replace = "CREATE TABLE $newtable";
$string  = $structure;
$query = preg_replace($search, $replace, $string);
```

Listing: 4.5 – Replace old table name with new table name in structure from MASTER TABLE

Now we can create the new table with the same structure…

```
...
#  replace old table name with new table name in the structure statement
   $search  = "/CREATE TABLE $oldtable/";
   $replace = "CREATE TABLE $newtable";
   $string  = $structure;
   $query = preg_replace($search, $replace, $string);

#  create new table
   $result = $db->query("$query") or die(" Error adding table: <span
style='color:red;'>$query</span>");
...
```

Listing: 4.6 – Create new table

```
<?php
/* Listing 9.4 - Copy table structure */
   echo '<body style="font-family:arial;">';
```


Copyright © 2005-2015

```
   echo '<h2>Copy table structure</h2>';

#  setup
   $dbPath   = $_SERVER['DOCUMENT_ROOT'] . "/contacts/";
   $dbFile   = "sample.sqlite3";
   $oldtable = "test";
   $newtable = "newtbl";

  ... code to connect to database, and handle errors

#  get table structure
   $result = $db->query("SELECT * FROM sqlite_master WHERE type='table' and name='$oldtable' ");
   $row = $result->fetch(PDO::FETCH_ASSOC);
   $structure = $row['sql'];

#  replace old table name with new table name in the structure statement
   $search  = "/CREATE TABLE $oldtable/";
   $replace = "CREATE TABLE $newtable";
   $string  = $structure;
   $query = preg_replace($search, $replace, $string);
#  create new table
   $result = $db->query("$query") or die(" Error adding table: <span
style='color:red;'>$query</span>");

#  show success
   echo "<i>Table added successfully.</i><br> ";

#  Return rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected ";

#  close db, PHP will also do this  automatically
   $db = NULL;
?>
```

Listing: 4.7 – Complete listing for copying table structure

Delete Table

To drop (delete) a table, we use this command…

```
   $query = "DROP TABLE 'test' ";
```

Listing: 4.8 – Delete table

Rename Table

The syntax used to rename a table is

```
   $query = "ALTER TABLE 'oldTablename' RENAME TO 'newTablename' ";
```

Listing: 4.9 – Rename table

This command ***cannot be used to move a table between two different databases***, ***only to rename*** a table within the same database.

View Table Structures

The syntax used to rename a table is

```
<?php
/* Listing 9.7 - View table structures */
echo '<body style="font-family:arial;">';
echo '<h2>View table structures</h2>';

#   setup
$dbPath  = $_SERVER['DOCUMENT_ROOT'] . "/contacts/";
$dbFile  = "sample.sqlite3";

  ... code to connect to database, and handle errors

$result = $db->query("SELECT * FROM sqlite_master WHERE type='table'"); #Lists all tables
$rows = $result->fetchall(PDO::FETCH_ASSOC);

#  print_r output
   echo "<br><span style='color:red;'>Table structures for $dbFile</span>";
   foreach($rows as $array) {
      echo "<pre>";
      print_r($array);
      echo "</pre>";
   }
?>
```

Listing: 4.10 – Delete table

```
View table structures

Table structures for sample.sqlite3
Array
(
    [type] => table
    [name] => test
    [tbl_name] => test
    [rootpage] => 2
    [sql] => CREATE TABLE test (
            id INTEGER PRIMARY KEY,
            name  TEXT(25),
            phone TEXT(25)
         )
)
Array
(
    [type] => table
    [name] => new
    [tbl_name] => new
    [rootpage] => 3
    [sql] => CREATE TABLE new (c1 TEXT, c2 TEXT)
)
```

OUTPUT – For listing 4.10

Table Description

Type field will always be 'table' and the name field will be the 'name' of the table. To get a list of all tables in the database, use the following SELECT command:

```
SELECT name FROM sqlite_master
```


Copyright © 2005-2015

```
WHERE type='table'
ORDER BY name;
```

Listing: 4.11 – table descriptione

For indices, type is equal to 'index', name is the name of the index, and tbl_name is the name of the table to which the index belongs.

For both tables and indices, the sql field is the text of the original CREATE TABLE or CREATE INDEX statement that created the table or index.

End Of Section

Copyright © 2005-2015

Section - 6
ROW / RECORD (manipulation)

Insert Row (add)

As we have done before, we will put our query statement in a variable so that we can have a better error message.

FILLING ALL FIELDS

```
    $query = "INSERT INTO 'tablename' VALUES('col1Data','col2Data','col3Data')";
    $result = $db->query("$query") or die(" Error: <span style='color:red;'>$query</span>");
```

Listing: 5.1 – insert data – inserting a single row (record) of information.

We did not specify an **id**. Because **id** is a ***primary field***, SQLite is smart enough to automatically increment a unique number for us.

Listing-5.1 is a short form of the INSERT command and expects *a value for every column (field)*. If we intend to leave some columns blank or are going to edit or update certain columns in a row, we must specify which column names we are going to be working with.

Long version of INSERT

SELECTING SPECIFIC FIELDS TO FILL

```
    $query = "INSERT INTO 'tablename' (field_x,field_y) VALUES('col1Data',' col2Data')";
    $result = $db->query("$query") or die(" Error: <span style='color:red;'>$query</span>");
```

Listing: 5.2 – insert data – column names specified.

Insert - Multiple Rows

```
insert into mytable (col,col2,col3)
  select 'a1', 'b1', 'c1'
  union
  select 'a2', 'b2', 'c2'
  union
  select 'a3', 'b3', 'c3'
  ...
```

Copy Row

To copy a row we first SELECT the row to copy then INSERT it as a new row.

Delete All Rows

To delete all the rows in a table we can use this…

```
    $query = "DELETE FROM 'tablename'";
```

Listing: 5.3 –

Delete Row(s) Matching Condition

If we know the record id then we can use this…

```
    $query = "DELETE FROM tablename WHERE id='xxx' ";
```

Listing: 5.4 –

If we do not know the record **id** then we must search for the record on one of its fields and return the id for it…

```
    $query = "DELETE FROM test WHERE id=(SELECT id FROM test WHERE fieldName LIKE 'condition') ";
```

Listing: 5.5 –

Here is a complete example that deletes the record that starts with a name od "James".

```
<?php
/* Listing 10.3 - Search and delete row */
   echo '<body style="font-family:arial;">';
   echo '<h2>Search and delete row</h2>';

  ... code to connect to database, and handle errors

#  delete row
   $query = "DELETE FROM test WHERE id=(SELECT id FROM test WHERE name LIKE 'James%') ";
   $result = $db->query("$query") or die(" Error deleting row: <span
style='color:red;'>$query</span>");

#  show success
   echo "<i>Row deleted successfully.</i><br> ";

#  Return rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected ";

#  close db, PHP will also do this  automatically
   $db = NULL;
?>
```

Listing: 5.6 – Search and delete a row

```
Delete row
Row deleted successfully.
1 records affected
```

OUTPUT – result of running listing: 5.6

Update Row(s) Matching Condition

For indices, type is equal to 'index', name is the name of the index, and tbl_name is the name of the table to which the index belongs. , use the following SELECT command:

```
UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
```

Listing: 5.8 –

Another way to update is to match a string and replace,

```
UPDATE table_name
SET field=replace(field, 'george', 'George')
WHERE zip LIKE '%99123%'
```

To replace all newline characters from a field (x'0a' is also used),

```
UPDATE table_name
SET field=replace(field, x'0d', '')
WHERE condition
```

Listing: 5.8 –

Copyright © 2005-2015

SQL UPDATE WARNING

Be careful when updating records. If we had omitted the WHERE clause then *ALL* rows will be updated, like this:

```
UPDATE Persons
SET Address='Nissestien 67', City='Sandnes'
```

Listing: 5.8 –

For both tables and indices, the sql field is the text of the original CREATE TABLE or CREATE INDEX statement that created the table or index.

Count Rows

There is no way for PDO to know the rowCount of a SELECT result because the SQLite API itself doesn't offer this ability. Therefore rowCont() does not work. We can count columns using PDO’s built in columnCount() function. Here is how to get these counts.

1. Use PDO's fetch_all() function to fetch all the rows into an array, then use count() on it.
2. Do an extra query to SELECT COUNT(*).

```
$query = "SELECT * FROM 'tablename'";
$result = $db->query("$query")
or die("Error counting rows: <span style='color:red;'>$query</span>");
$rowcount = count($result);
echo "Total rows = $rowcount<br>";

$query = "SELECT * FROM 'tablename'";
$result = $db->query("$query")
or die("Error counting rows: <span style='color:red;'>$query</span>");
$rowcount = count($result->fetchAll());
echo "Total rows = $rowcount<br>";

$query = "SELECT COUNT(*) FROM 'tablename'";
$result = $db->query("$query")
or die("Error counting rows: <span style='color:red;'>$query</span>");
$rows = $result->fetchAll();
$rowcount = $rows[0][0];
echo "Total rows = $rowcount<br>";
```

Count Columns

```
$query = "SELECT * FROM 'tablename'";
$result = $db->query("$query")
or die("Error counting rows: <span style='color:red;'>$query</span>");
$colcount = $result->columnCount();
echo "Column count = $colcount<br>";
```

Listing: 5.7 – Count rows and columns

Index Description

For indices, type is equal to 'index', name is the name of the index, and tbl_name is the name of the table to which the index belongs. , use the following SELECT command:

```
SELECT name FROM sqlite_master
WHERE type='index'
```


Copyright © 2005-2015

```
ORDER BY name;
```

Listing: 5.8 –

For both tables and indices, the sql field is the text of the original CREATE TABLE or CREATE INDEX statement that created the table or index.

End Of Section

Copyright © 2005-2015

Section - 7
COLUMN / FIELD (manipulation)

Add Column

```
$query = "ALTER TABLE '[database.]tablename' ADD COLUMN 'newColumn' TEXT ";
```

Listing: 6.1 - Creating a new database

Copy Column and data

SQLite ALTER TABLE only supports "add column" to the end of a table, or "change table name". If you want to make other changes than the table will have to be copied in the proper order and renamed.

```
$result = $db->query("CREATE TABLE '$newtable' AS SELECT  'colToCopy ' FROM '$oldtable'");
```

Listing: 6.2 – copy a column to a new table

Delete Column

Suppose you have a table named "t1" with columns names "a", "b", and "c". If you want to delete column "c" from this table, than the following steps illustrate how this could be done:

```
CREATE TEMPORARY TABLE t1_backup(a,b);
INSERT INTO t1_backup SELECT a,b FROM t1;
DROP TABLE t1;
CREATE TABLE t1(a,b);
INSERT INTO t1 SELECT a,b FROM t1_backup;
DROP TABLE t1_backup;
```

Listing: 6.3 - Creating a new database

Rename Column

Say you have a table and need to rename "old_b" to "new_b":

```
1) Rename the old table:
    ALTER TABLE orig_table_name RENAME TO tmp_table_name;

2) Create a new table, based on the old table but with the updated column name:

    CREATE TABLE orig_table_name (
    col_a INT,
    new_b INT
    );

3) Copy the contents across from the original table.

    INSERT INTO orig_table_name(col_a, new_b)
    SELECT col_a, old_b
    FROM tmp_table_name;

4) Drop the old table.
    DROP TABLE tmp_table_name;
```

Listing: 6.4 - Creating a new database

Count Columns

Copyright © 2005-2015

We can count columns using PDO’s built in columnCount() function. There is no way for PDO to know the rowCount of a SELECT result because the SQLite API itself doesn't offer this ability. Therefore rowCont() does not work. Here is how to get these counts.

```
$rowcount = count($result->fetchAll());
echo "Total rows = $rowcount<br>";

$colcount = $result->columnCount();
echo "Column count = $colcount<br>";
```

Listing: 6.5 – Count rows and columns

End Of Section

Copyright © 2005-2015

Section - 8
SEARCHES / SHOW (manipulation)

Simple Select Syntax

Recall from section-5 that we can search our database using SELECT then extract the information using FETCH. The simple form of this command is shown below.

```
$query = "SELECT <column(s)> FROM 'table' ";
$result = $db->query($query, PDO::FETCH_ASSOC);
$row = $result->fetchall(PDO::FETCH_ASSOC);
```

Listing: 7.1 – Simple searches using SELECT

We often use an asterisk * for the column information so that all columns in the row(s) are returned.

```
$query = "SELECT * FROM 'table' ";
```

Listing: 7.2 – Generic form of SELECT

The four fetch types

```
/* PDOStatement::fetch styles */
$query = "select * from t1";

print("<br><br>PDO::FETCH_ASSOC: next row as an array indexed by column name.<br>");
$result = $db->query($query);
$row = $result->fetch(PDO::FETCH_ASSOC);
echo "<pre>"; print_r($row); echo "</pre>";
print("<br><br>");

print("PDO::FETCH_BOTH: next row as an array indexed by both column name and number<br>");
$result = $db->query($query);
$row = $result->fetch(PDO::FETCH_BOTH);
echo "<pre>"; print_r($row); echo "</pre>";
print("<br><br>");

print("PDO::FETCH_LAZY: next row as an anonymous object with column names as properties<br>");
$result = $db->query($query, PDO::FETCH_LAZY);
$row = $result->fetch(PDO::FETCH_LAZY);
echo "<pre>"; print_r($row); echo "</pre>";
print("<br><br>");

print("PDO::FETCH_OBJ: next row as an anonymous object with column names as properties<br>");
$result = $db->query($query, PDO::FETCH_OBJ);
$row = $result->fetch(PDO::FETCH_OBJ);
echo "<pre>"; print_r($row); echo "</pre>";
print("<br><br>");
```

Listing: 7.3 –

Results from the above listing

```
PDO::FETCH_ASSOC: next row as an array indexed by column name.
Array
(
    [id] => 1
    [name] => Frank
    [dob] => 2011.12.09
```


Copyright © 2005-2015

```
)


PDO::FETCH_BOTH: next row as an array indexed by both column name and number
Array
(
    [id] => 1
    [0] => 1
    [name] => Frank
    [1] => Frank
    [dob] => 2011.12.09
    [2] => 2011.12.09
)


PDO::FETCH_LAZY: next row as an anonymous object with column names as properties PDORow Object
(
    [queryString] => select * from t1
    [id] => 1
    [name] => Frank
    [dob] => 2011.12.09
)


PDO::FETCH_OBJ: next row as an anonymous object with column names as properties stdClass Object
(
    [id] => 1
    [name] => Frank
    [dob] => 2011.12.09
)
```

WHERE

The WHERE clause includes a comparison predicate, which restricts the rows returned by the query. The WHERE clause eliminates all rows from the result set for which the comparison predicate does not evaluate to True. Remember that if we use the WHERE statement by itself, it requires an exact match.

```
$query = "SELECT * FROM 'table' WHERE name='James Bond' ";
or
$query = "SELECT * FROM 'table' WHERE name='$lname' "; // must use ' ticks with vars
```

Listing: 7.4 – Simple searches using WHERE

If we want to use wild cards then we must add the LIKE statement. When using LIKE, we have the following rules:

```
_   =    under score "_" will match any single character (like the ?  in batch files, dos, or unix)
%   =    percent "%" will match zero or more of any character (like the * in batch files, dos, or
unix)
```

SHOW RECORD ON SINGLE LINE

```
# show each record on single line
  $result = $db->query("SELECT * FROM 'table' WHERE 1 ");
  $rows = $result->fetchall(PDO::FETCH_ASSOC);
```

```
foreach($rows as $row) {
 $record = implode(' | ',$row);
 echo "$record<br>";
}
```

LIKE

The LIKE clause allows use to use wildcards or less specific match criteria

We can also use more than one condition.

```
$query = "SELECT * FROM 'table' WHERE name LIKE 'James%'";
```

Listing: 7.5 – Simple searches using LIKE

AND

We can use AND to require more than one condition. Consider this example…

```
$query = "SELECT * FROM 'table' WHERE name LIKE 'James%' AND WHERE phone LIKE '007%' ";
```

Listing: 7.6 – Simple searches using AND for multiple conditions

OR

We also could use OR to select between different conditions. Consider this example…

```
$query = "SELECT * FROM 'table' WHERE name LIKE 'James%' OR WHERE name LIKE 'Jane%' ";
```

Listing: 7.7 – Simple searches using AND for multiple conditions

We have several other options and that leads us to our section on advanced searches.

Advanced Select Syntax

LIMIT

What if we need a set of rows from a given offset such as the *nth* record? We can use the LIMIT statement to draw results starting with a given index and get the specified number of rows from that point. The syntax is…

```
LIMIT index, count
```

Here is a typical example where we want the third record from the match set…

```
$query = "SELECT * FROM 'tablename' LIMIT 2,1";
```

Listing: 7.8 –

This returns the third record because ewe used an index of "2" and indexes start with zero (0,1,2…) We will get 1 record back because we gave a record count of 1. If we wanted the first 5 records after 100 then we would use LIMIT 99,5.

There are two very common uses for LIMIT. The first is to show a reasonable number of records on the screen or printer page. If we were going to list 100 records from a certain date, we might want half of them on the first page and the other half printed on the second page.

Our statements might look something like this…

```
$query = "SELECT * FROM 'tablename' WHERE 'dob'='1975' LIMIT 0, 49";
```

Listing: 7.9 –

Our next query would be…

```
$query = "SELECT * FROM 'tablename' WHERE 'dob'='1975' LIMIT 50, 50";
```

Listing: 7.10 –

The other common use of limit is to get the previous or next record from the current one. If we are at index 65, we can add or subtract 1 from this index and get a record in either direction.

```
# get next record
$rowindex = 65;
$query = "SELECT * FROM 'tablename' LIMIT ($rowindex+1), 1";

# more...code

# get previous record
$rowindex = 65;
$query = "SELECT * FROM 'tablename' LIMIT ($rowindex-1), 1";
```

Listing: 7.11 –

We could use LIMIT $rowindex+1,1 but I think the ( ) makes the code more readable so I use LIMIT ($rowindex+1),1. Of course we can have more than one row returned by using different values for count.

ORDER BY

The `ORDER BY` clause identifies which columns are used to sort the resulting data, and in which direction they should be sorted (options are ascending or descending). Without an `ORDER BY` clause, the order of rows returned by an SQL query usually correlates to the **primary key** field such as an ID number.

The syntax is…

```
ORDER BY column
   or
 ORDER BY column "DIRECTION"
```

Consider this example…

```
# sort records by last name in DESCENDING order
$query = "SELECT * FROM 'tablename' ORDER BY 'lname''DESC';
# more...code
# sort records by last name in ASCENDING order (default)
$query = "SELECT * FROM 'tablename' ORDER BY 'lname''ASC';
```

Listing: 7.12 –

If no sort order is specified, it will be ASCENDING which is alphabetical and/or small to large numbers. To reverse the default sort order we must use "DESC". We can use more than one column id there is a match. For example suppose we are sorting student names in a school and there is more than one "Smith". We can use the date of birth too…

```
# sort records by last name in ASCENDING order (default)
$query = "SELECT * FROM 'tablename' ORDER BY 'lname''DESC','dob''ASC';
```

Listing: 7.13 –

This sorts the last names alphabetically (normal order), but the **'dob''ASC'** lists the students with the same name; oldest to youngest according to date of birth (opposite the normal order).

GROUP BY

Copyright © 2005-2015

Only shows fields explicitly mentioned. When the records are grouped, all fields except those which are explicitly given in GROUP BY are removed. The `GROUP BY` clause is used to project rows having common values into a smaller set of rows. `GROUP BY` is often used in conjunction with SQL aggregation functions or to eliminate duplicate rows from a result set. The `WHERE` clause is applied before the `GROUP BY` clause.

HAVING

The `HAVING` clause includes a predicate used to filter rows resulting from the `GROUP BY` clause. Because it acts on the results of the `GROUP BY` clause, aggregation functions can be used in the `HAVING` clause predicate. A **`HAVING`** clause in SQL specifies that an SQL `SELECT` statement should only return rows where aggregate values meet the specified conditions. It was added to the SQL language because the `WHERE` keyword could not be used with aggregate functions

To return a list of department IDs whose total sales exceeded $1000 on the date of January 1, 2000, along with the sum of their sales on that date:

```
SELECT DeptID, SUM(SaleAmount)
FROM Sales
WHERE SaleDate = '01-Jan-2000'
GROUP BY DeptID
HAVING SUM(SaleAmount) > 1000
```

Listing: 7.14 –

The following query will return the list of departments who have more than 1 employee:

```
SELECT DepartmentName, COUNT(*)
FROM employee,department
WHERE employee.DepartmentID = department.DepartmentID
GROUP BY DepartmentName
HAVING COUNT(*)>1;
```

Listing: 7.15 –

DISTINCT

If data is duplicated, returns only one copy.

```
SELECT distinct f1 FROM t1 WHERE f1=condition
```

If we had TABLE of names and fields like this….

id	fname	lname
1	Mickey	Mouse
2	Donald	Duck
3	Mini	Mouse
4	Might	Mouse

```
SELECT distinct lname FROM names WHERE 1
```

```
RESULT: Mouse, Duck
```

FINDING DUPLICATES

Copyright © 2005-2015

With the SQL statement below you can find duplicate values in any table, just change the table field into the column you want to search and change the table into the name of the table you need to search.

This will show the duplicated record

```
SELECT *
FROM table
GROUP BY field
HAVING (COUNT( field ) > 1)
```

This shows table field and how many times it is found as a duplicate.

```
SELECT     tablefield, COUNT(tablefield) AS dup_count
FROM         table
GROUP BY tablefield
HAVING     (COUNT(tablefield) > 1)
```

A faster method is

```
SELECT t1.*
FROM table AS t1
LEFT JOIN table AS t2
ON t1.id != t2.id
AND t1.field = t2.field
WHERE t2.id IS NOT NULL
GROUP BY t1.id
```

Some further tempering with the statement gets the complete records that are doubled.

```
SELECT *
FROM table
WHERE tablefield IN (
 SELECT tablefield
 FROM table
 GROUP BY tablefield
 HAVING (COUNT(tablefield ) > 1)
)
```

DELETEING DUPLICATES

```
delete from MyTable
where exists (select * from MyTable t2 where t2.field = MyTable.field and t2.id < MyTable.id)
```

End Of Section

Copyright © 2005-2015

Section - 9
Debug & Errors

Error Codes

```
The error codes for SQLite version 3 are unchanged from version 2.
They are as follows:
OK           0   Successful result
ERROR        1   SQL error or missing database
INTERNAL     2   An internal logic error in SQLite
PERM         3   Access permission denied
ABORT        4   Callback routine requested an abort
BUSY         5   The database file is locked
LOCKED       6   A table in the database is locked
NOMEM        7   A malloc() failed
READONLY     8   Attempt to write a readonly database
INTERRUPT    9   Operation terminated by sqlite_interrupt()
IOERR       10   Some kind of disk I/O error occurred
CORRUPT     11   The database disk image is malformed
NOTFOUND    12   (Internal Only) Table or record not found
FULL        13   Insertion failed because database is full
CANTOPEN    14   Unable to open the database file
PROTOCOL    15   Database lock protocol error
EMPTY       16   (Internal Only) Database table is empty
SCHEMA      17   The database schema changed
TOOBIG      18   Too much data for one row of a table
CONSTRAINT  19   Abort due to constraint violation
MISMATCH    20   Data type mismatch
MISUSE      21   Library used incorrectly
NOLFS       22   Uses OS features not supported on host
AUTH        23   Authorization denied
ROW        100   sqlite_step() has another row ready
DONE       101   sqlite_step() has finished executing
```

1) If you copied code from a page of this reference

If you copied and pasted code from this MS Word document. Be forewarned that it transposes straight ticks ('), apostrophes ‘, and quotes ("") into forward and reverse versions which are incompatible with the PHP parser. I have made an attempt to remove as many as possible, but a few may linger.

A full set of proven routines should have accompanied this document.

2) Some SQLite versions have incompatibilities

NOTE: Databases are unsupported between 3.1.x and 3.2.x versions of SQLite.

file_format 1 Version 3.0.0.
file_format 2 Version 3.1.3.
file_format 3 Version 3.1.4.

Version 3.0 can only use files with file_format 1.
Version 3.1.3 can read and write files with file_format 1 or file_format 2.
Version 3.1.4 can read and write file formats 1, 2 and 3.

Copyright © 2005-2015

A typical error is something like this, "malformed database schema - unable to open a database file…"

3) "Invalid resource" error

If you get an error reporting "invalid resource" when attempting a query on a database table and looping through it, the version of the SQLite extension compiled in to PHP might be incompatible with the version that was used to create the database (like SQLite 2.x vs 3.x).

The database might open (connect) ok, but will fail when querying.

$db = new PDO('sqlite:/tmp/foo.db'); // success
foreach ($db->query('SELECT * FROM bar') as $row) // prints invalid resource

4) Sensitivity to white space

In certain circumstances white space can cause errors, be careful!

```
Correct ->   $db = new PDO("sqlite:$dbPath$dbFile");
Error   ->   $db = new PDO("sqlite: $dbPath$dbFile");
                                   ↑
Correct ->   $db = new PDO("sqlite:$dbPath$dbFile");
Error   ->   $db = new PDO("sqlite:$dbPath$dbFile ");
                                                ↑
Correct ->   ")
Error   ->   " )
              ↑
```

5) "unable to open file"

If you receive an error while trying to write to a sqlite database (update, delete, drop):

```
"Warning: PDO::query() [function.query]: SQLSTATE[HY000]: General error: 1 unable to open database".
```

A unix system may have folder or file permission problems. The folder containing the database file must be writeable.

6) "SQLITE_CORRUPT error"

An SQLITE_CORRUPT error is returned when SQLite detects an error in the structure, format, or other control elements of the database file. Your can use PRAGMA integrity_check to do a thorough an intensive test of the database integrity or you can use PRAGMA quick_check to do a faster but less thorough test of the database integrity.

7) "WHERE Clause error"

The SQL standard requires **single-quotes**, ***not double-quotes***, around **string literals**. A WHERE clause expression should read: column1='column2' ***not*** column1= "column2".

8) Unexpected return value, i.e. "Array"

After query using something similar to…

```
$result = $db->query("SELECT * FROM 'test'");
$rows = $result->fetchAll();
echo "rows = $rows";
```


Copyright © 2005-2015

echo shows…

```
"Array"
```

The result array has not been fully processed. In other words, the result is an array and needs to be accessed using index values such as **foreach($rows as $row)** clause or **$var = $rows[xx].**

9) SYNTAX ERROR, Unexpected "End"…

A bracket is missing from a for-next or for-each construct…

```
foreach($rows as $row){
do something...;

← missing closing "}"
```

10) "unexpected T_VARIABLE"

This error usually indicates a syntax error. The most common causes are…

missing ";" in a previous line

matching character pairs missing, i.e ", ', [, {, (etc.

End Of Section

Copyright © 2005-2015

Section - 10
VIEWS (custom)

A **view** is a specific look on data in from one or more tables. It can arrange data in some specific order, higlight or hide some data. A view consists of a stored query accessible as a virtual table composed of the result set of a query. Unlike ordinary tables a view does not form part of the physical schema. It is a dynamic, virtual table computed or collated from data in the database.

In the next example, we create a simple view.

```
sqlite> SELECT * FROM Cars;
```

Listing: 8.1 –

```
Id          Name        Cost
----------  ----------  ----------
1           Audi        52642
2           Mercedes    57127
3           Skoda       9000
4           Volvo       29000
5           Bentley     350000
6           Citroen     21000
7           Hummer      41400
8           Volkswagen  21600
```

Output: 8.1 –

This is our data, upon which we create the view.

```
sqlite> CREATE VIEW CheapCars AS SELECT Name FROM Cars WHERE Cost < 30000;
sqlite> SELECT * FROM CheapCars;
```

Listing: 8.2 –

```
Name
----------
Skoda
Volvo
Citroen
Volkswagen
```

Output: 8.2 –

Technically a view is a virtual table. So we can list all views with a **.tables** command. To remove a view, we use the **DROP VIEW** SQL statement.

```
sqlite> .tables
Books          CheapCars      Friends        Names          Reservations
Cars           Customers      Log            Orders         Testing
sqlite> DROP VIEW CheapCars;
sqlite> .tables
Books          Customers      Log            Orders         Testing
Cars           Friends        Names          Reservations
```

End Of Section

Copyright © 2005-2015

Section - 11
TRIGGERS (manipulation)

Triggers are database operations that are automatically performed when a specified database event occurs.

Insert Row Timestamp

It is a good practice to keep tract of the last time a file was edited. We do this by saving a time stamp. An insert trigger is created below in the file "trigger1". The Coordinated Universal Time (UTC) will be entered into the field "timeEnter", and this trigger will fire *after* a row has been inserted into the table t1.

```
-- ********************************************************************
--   Creating a trigger for timeEnter
--     Run as follows:
--            $ sqlite3 test.db < trigger1
-- ********************************************************************
CREATE TRIGGER update_editdate AFTER INSERT ON t1
BEGIN
 UPDATE t1 SET editdate = DATETIME('NOW')  WHERE rowid = new.rowid;
END;
-- ********************************************************************
```

Listing: 9.1 –

Another way of keeping a timestamp (perhaps easier than a trigger) is to use the default value of the table scheme. The field definition is given below.

```
       editdate  DATETIME DEFAULT (datetime('now','localtime')),
```

Listing: 15.2 – Timestamp using default column text

Logging All Inserts, Updates, and Deletes

The script below creates the table examlog and three triggers (update_examlog, insert_examlog, and delete_examlog) to record updates, inserts, and deletes made to the exam table. In other words, whenever a change is made to the exam table, the changes will be recorded in the examlog table, including the old value and the new value. If you are familiar with MySQL, the functionality of this log table is similar to MySQL's binlog. See Tips 2, 24, and 25 if you would like more information on MySQL's log file.

```
--  Create an update trigger
CREATE TRIGGER update_examlog AFTER UPDATE ON exam
BEGIN

  INSERT INTO examlog  (ekey,ekeyOLD,fnOLD,fnNEW,lnOLD,
                        lnNEW,examOLD,examNEW,scoreOLD,
                        scoreNEW,sqlAction,examtimeEnter,
                        examtimeUpdate,timeEnter)

          values (new.ekey,old.ekey,old.fn,new.fn,old.ln,
                  new.ln,old.exam, new.exam,old.score,
                  new.score, 'UPDATE',old.timeEnter,
                  DATETIME('NOW'),DATETIME('NOW') );

END;
--
--  Also create an insert trigger
--    NOTE  AFTER keyword ------v
CREATE TRIGGER insert_examlog AFTER INSERT ON exam
BEGIN
```


Copyright © 2005-2015

```
INSERT INTO examlog  (ekey,fnNEW,lnNEW,examNEW,scoreNEW,
                      sqlAction,examtimeEnter,timeEnter)

          values (new.ekey,new.fn,new.ln,new.exam,new.score,
                  'INSERT',new.timeEnter,DATETIME('NOW') );

END;

--  Also create a DELETE trigger
CREATE TRIGGER delete_examlog DELETE ON exam
BEGIN

INSERT INTO examlog  (ekey,fnOLD,lnNEW,examOLD,scoreOLD,
                      sqlAction,timeEnter)

          values (old.ekey,old.fn,old.ln,old.exam,old.score,
                  'DELETE',DATETIME('NOW') );

END;
```

Listing: 9.2 –

In the following example, we will use the Friends table and create a new Log table.

```
CREATE TABLE Log(Id integer PRIMARY KEY, OldName text, NewName text, Date text);
```

Listing: 9.3 –

We will create 2 new tables for the next example with triggers.

```
CREATE TRIGGER mytrigger UPDATE OF Name ON Friends
BEGIN
INSERT INTO Log(OldName, NewName, Date) VALUES(old.Name, new.Name, datetime('now'));
END;
```

Listing: 9.4 –

We create a trigger called mytrigger with the CREATE TRIGGER statement. This trigger will launch a INSERT statement whenever we update the name column of the Friends table. The INSERT statement will insert the old name, the new name and the time stamp into the Log table.

```
sqlite> SELECT * FROM Friends;
Id          Name        Sex
----------  ----------  ----------
1           Jane        F
2           Thomas      M
3           Franklin    M
4           Elisabeth   F
5           Mary        F
6           Lucy        F
7           Jack        M
```

This is our data.

Next, we are going to update one row of the Friends table.

```
sqlite> UPDATE Friends SET Name='Frank' WHERE Id=3;
```

Listing: 9.5 –

We update the third row of the table. The trigger is launched.

```
sqlite> SELECT * FROM Log;
Id          OldName     NewName     Date
```

```
----------  ----------  ----------  -------------------
1           Franklin    Frank       2009-11-14 16:36:28
```

We check the Log table. This log confirms the update operation we performed.

End Of Section

Copyright © 2005-2015

Section - 12
TRANSACTIONS

A **transaction** is an atomic unit of database operations against the data in one or more databases. The effects of all the SQL statements in a transaction can be either all committed to the database or all rolled back.

In SQLite, any command other than the SELECT will start an implicit transaction. Manual transactions are started with the **BEGIN TRANSACTION** statement and finished with the **COMMIT** OR **ROLLBACK** statements.

```
BEGIN TRANSACTION;
CREATE TABLE Test(Id integer NOT NULL);
INSERT INTO Test VALUES(1);
INSERT INTO Test VALUES(2);
INSERT INTO Test VALUES(3);
INSERT INTO Test VALUES(NULL);
COMMIT;
```

Listing: 10.1 –

We have a NOT NULL constraint set on the Id column. Thus, the fourth insert will not succeed. SQLite does transactions specifically. For some errors, it reverts all changes. For others, it reverts only the last statement and leaves other changes intact. In our case, the table is created and the first three inserts are written into the table. The fourth one is not.

Say, we already had an empty table named Test. Executing the above transaction would fail completely. No changes would be written. If we changed the **CREATE TABLE** statement into **CREATE TABLE IF NOT EXISTS**, the first three statements would execute.

```
BEGIN TRANSACTION;
CREATE TABLE IF NOT EXISTS Test(Id integer NOT NULL);
INSERT INTO Test VALUES(1);
INSERT INTO Test VALUES(2);
INSERT INTO Test VALUES(3);
INSERT INTO Test VALUES(NULL);
ROLLBACK;
```

Listing: 10.2 –

A transaction can end with a **COMMIT** or a **ROLLBACK** statement. The **ROLLBACK** reverts all changes.

End Of Section

Copyright © 2005-2015

Section - 13
JOINS

JOIN: Combine data from two or more tables

INNER JOIN: Default Join type. Create new table where one or more table rows meet criteria.

NATURAL JOIN: Combines all columns in tables with same column-name.

LEFT OUTER JOIN: Retain all records in table on the left side (regardless of match), merge only matches from right side.

RIGHT OUTER JOIN: Retain all records in table on the right side (regardless of match), merge only matches from left side.

Note: Left and right Join refer to the two sides of the JOIN keyword. Left and RIGHT Joins are identical by reversing table positions.

FULL JOIN: Merge all tables regardless of match.

SELF JOIN: Search one table for matches in a column with two ore more identical entries. i.e. Same dept name –excludes single entries.

CROSS JOIN: Gets a record from the first table and then creates a new row for every row in the 2nd table. It then does the same for the next record in the first table etc. Caution should be used, when there is no WHERE filter applied the number of records returned will be T1 * T2 records.

MERGE ROW: Concatenates field data against match column

UNION: The UNION operator is used to combine the result-set of two or more SELECT statements.

Copyright © 2005-2015

Suppose we have the following tables

```
sqlite> .headers ON
sqlite> SELECT * FROM Artists;
ArtistID|ArtistName
1       |Peter Gabriel
2       |Bruce Hornsby
3       |Lyle Lovett
4       |Beach Boys
5       |Supernatural

sqlite> SELECT * FROM CDs;
CDID|ArtistID|Title              |Date
1   |1       |So                 |1984
2   |1       |Us                 |1992
3   |2       |The Way It Is      |1986
4   |2       |Scenes from the Southside|1990
5   |1       |Security           |1990
6   |3       |Joshua Judges Ruth |1992
7   |4       |Pet Sounds         |1966
```

CROSS JOIN

It is also called the "direct join" or Cartesian product:

```
sqlite>SELECT * FROM Artists, CDs;
```

Listing: 11.1 –

In some SQL dialect it can be also used

```
sqlite>SELECT * FROM Artists CROSS JOIN CDs;
```

Listing: 11.2 –

Let us try to filter the result with the WHERE clause:

```
sqlite>SELECT ArtistName, Title FROM Artists, CDs WHERE Artists.ArtistID=CDs.ArtistID;
```

Listing: 11.3 –

It gives

```
Peter Gabriel|So
Peter Gabriel|Us
Peter Gabriel|Security
Bruce Hornsby|The Way It Is
Bruce Hornsby|Scenes from the Southside
Lyle Lovett  |Joshua Judges Ruth
Beach Boys   |Pet Sounds
```

To avoid confusion, use this statement instead

```
sqlite>SELECT Artists.ArtistName, CDs.Title FROM Artists, CDs WHERE Artists.ArtistID=CDs.ArtistID;
or with aliases
sqlite>SELECT a.ArtistName, c.Title FROM Artists a, CDs c WHERE a.ArtistID=c.ArtistID;
```

Listing: 11.4 –

INNER JOIN

The same result can be achieved with the INNER JOIN clause:

```
sqlite> SELECT Artists.ArtistName, CDs.Title FROM Artists INNER JOIN CDs ON
Artists.ArtistID=CDs.ArtistID;
```

Listing: 11.5 –

The INNER JOIN returns all rows from both tables where there is a match. If there are rows in Artists that do not have matches in CDs, those rows will **not** be listed.

LEFT OUTER JOIN

The LEFT OUTER JOIN operator ensures that all rows on the "left" side of the join, in this case the Artists table, will be included.

```
sqlite>SELECT * FROM Artists LEFT OUTER JOIN CDs ON Artists.ArtistID = CDs.ArtistID;
```

Listing: 11.6 –

There is no Supernatural in CDs but you can still see Supernatural with empty fields from Artists:

```
ArtistID|ArtistName    |CDID|ArtistID|Title                 |Date
1       |Peter Gabriel|1    |1       |So                    |1984
1       |Peter Gabriel|2    |1       |Us                    |1992
1       |Peter Gabriel|5    |1       |Security              |1990
2       |Bruce Hornsby|3    |2       |The Way It Is         |1986
2       |Bruce Hornsby|4    |2       |Scenes from the Southside|1990
3       |Lyle Lovett  |6    |3       |Joshua Judges Ruth    |1992
4       |Beach Boys   |7    |4       |Pet Sounds            |1966
5       |Supernatural |     |        |                      |
```

Some SQL dialects use LEFT JOIN.

In other dialects (Oracle) instead of

```
sqlite> SELECT Artists.ArtistName, CDs.Title
------> FROM Artists
------> LEFT OUTER JOIN CDs
------> ON Artists.ArtistID = CDs.ArtistID;
```

Listing: 11.7 –

the following statement is used

```
SELECT Artists.ArtistName, CDs.Title
FROM Artists, CDs
WHERE Artists.ArtistID = CDs.ArtistID(+)
```

Listing: 11.8 –

The (+) symbol denotes the table (side) that may have no matching rows to the other table (side). Think of this as everything from the left plus matching values from the right.

RIGHT OUTER JOIN and FULL OUTER JOIN

Similarly this SQL

```
SELECT Artists.ArtistName, CDs.Title
FROM Artists
RIGHT OUTER JOIN CDs
ON Artists.ArtistID = CDs.ArtistID;
```

Listing: 11.9 –

Or

```
SELECT Artists.ArtistName, CDs.Title
FROM Artists, CDs
WHERE Artists.ArtistID(+) = CDs.ArtistID
```

Listing: 11.10 –

would return all records from the right side.

Currently SQLite supports **neither** of the above syntax. However there is easy workaround: use LEFT OUTER JOIN with interchanged tables. The FULL OUTER JOIN clause is not supported as well.

UNION

The UNION operator is used to combine the result-set of two or more SELECT statements.

Notice that each SELECT statement within the UNION must have the same number of columns. The columns must also have similar data types. Also, the columns in each SELECT statement must be in the same order.

SQL UNION Syntax

```
SELECT column_name(s) FROM table_name1
UNION
SELECT column_name(s) FROM table_name2
```

Listing: 11.11 –

Note: The UNION operator selects only distinct values by default. To allow duplicate values, use UNION ALL.

SQL UNION ALL Syntax

```
SELECT column_name(s) FROM table_name1
UNION ALL
SELECT column_name(s) FROM table_name2
```

Listing: 11.12 –

End Of Section

Copyright © 2005-2015

Section - 14
Date and Time

Function Format

SQLite supports six date and time functions as follows:

```
1. now()  same as datetime('now','localtime')
2. date(timestring, modifier, modifier, ...)
3. time(timestring, modifier, modifier, ...)
4. datetime(timestring, modifier, modifier, ...)
5. julianday(timestring, modifier, modifier, ...)
6. strftime(format, timestring, modifier, modifier, ...)
```

Listing: 12.1 –

All five date and time functions take a time string as an argument. The time string is followed by zero or more modifiers. The strftime() function also takes a format string as its first argument.

The date and time functions use a subset of IS0-8601 date and time formats. The date() function returns the date in this format: YYYY-MM-DD. The time() function returns the time as HH:MM:SS. The datetime() function returns "YYYY-MM-DD HH:MM:SS". The julianday() function returns the Julian day - the number of days since noon in Greenwich on November 24, 4714 B.C. (Proleptic Gregorian calendar). The strftime() routine returns the date formatted according to the format string specified as the first argument. The format string supports the most common substitutions found in the strftime() function from the standard C library plus two new substitutions, %f and %J. The following is a complete list of valid strftime() substitutions:

```
%d   day of month: 00
%f   fractional seconds: SS.SSS
%H   hour: 00-24
%j   day of year: 001-366
%J   Julian day number
%m   month: 01-12
%M   minute: 00-59
%s   seconds since 1970-01-01
%S   seconds: 00-59
%w   day of week 0-6 with Sunday==0
%W   week of year: 00-53
%Y   year: 0000-9999
%%   %
```

Listing: 12.2 –

Notice that all other date and time functions can be expressed in terms of strftime():

Function Equivalent strftime()

```
date(...)          strftime('%Y-%m-%d', ...)
time(...)          strftime('%H:%M:%S', ...)
datetime(...)      strftime('%Y-%m-%d %H:%M:%S', ...)
julianday(...)     strftime('%J', ...)
```

Listing: 12.3 –

The only reasons for providing functions other than strftime() is for convenience and for efficiency.

Time Strings A time string can be in any of the following formats:

Copyright © 2005-2015

```
1.  YYYY-MM-DD
2.  YYYY-MM-DD HH:MM
3.  YYYY-MM-DD HH:MM:SS
4.  YYYY-MM-DD HH:MM:SS.SSS
5.  YYYY-MM-DDTHH:MM
6.  YYYY-MM-DDTHH:MM:SS
7.  YYYY-MM-DDTHH:MM:SS.SSS
8.  HH:MM
9.  HH:MM:SS
10. HH:MM:SS.SSS
11. now
12. DDDDDDDDDD
```

Listing: 12.4 –

In formats 5 through 7, the "T" is a literal character separating the date and the time, as required by ISO-8601. Formats 8 through 10 that specify only a time assume a date of 2000-01-01. Format 11, the string 'now', is converted into the current date and time as obtained from the xCurrentTime method of the sqlite3_vfs object in use. Universal Coordinated Time (UTC) is used. Format 12 is the Julian day number expressed as a floating point value.

Modifiers

The time string can be followed by zero or more modifiers that alter the date and time string. Each modifier is a transformation that is applied to the time value to its left. Modifiers are applied from left to right; order is important.

The available modifiers are as follows.

```
1. NNN days
2. NNN hours
3. NNN minutes
4. NNN.NNNN seconds
5. NNN months
6. NNN years
7. start of month
8. start of year
9. start of day
10.weekday N
11.unixepoch
12.localtime
13.utc
```

Listing: 12.5 –

The first six modifiers (1 through 6) simply add the specified amount of time to the date and time specified by the preceding timestring and modifiers. Note that "+/- NNN months" works by rendering the original date into the YYYY-MM-DD format, adding the +/- NNN to the MM month value, then normalizing the result. Thus, for example, the data 2001-03-31 modified by '+1 month' initially yields 2001-04-31, but April only has 30 days so the date is normalized to 2001-05-01. A similar effect occurs when the original date is February 29 of a leapyear and the modifier is +/- N years where N is not a multiple of four.

The "start of" modifiers (7 through 9) shift the date backwards to the beginning of the current month, year or day.

The "weekday" modifier advances the date forward to the next date where the weekday number is N. Sunday is 0, Monday is 1, and so forth.

The "unixepoch" modifier (11) only works if it immediately follows a timestring in the DDDDDDDDDD format. This modifier causes the DDDDDDDDDD to be interpreted not as a Julian day number as it normally would be, but as Unix Time - the number of seconds since 1970. If the "unixepoch" modifier does not follow a timestring of the form DDDDDDDDDD which expresses the number of seconds since 1970 or if other modifiers separate the "unixepoch" modifier from prior

Copyright © 2005-2015

DDDDDDDDDD then the behavior is undefined. Due to precision limitations imposed by the implementations use of 64-bit integers, the “unixepoch” modifier only works for dates between 0000-01-01 00:00:00 and 5352-11-01 10:52:47 (unix times of -62167219200 through 10675199167).

The “localtime” modifier (12) assumes the time string to its left is in Universal Coordinated Time (UTC) and adjusts the time string so that it displays localtime. If “localtime” follows a time that is not UTC, then the behavior is undefined. The “utc” is the opposite of “localtime”. “utc” assumes that the string to its left is in the local timezone and adjusts that string to be in UTC. If the prior string is not in localtime, then the result of “utc” is undefined.

Timestamp

It is a good practice to keep tract of the last time a file was edited. We do this by saving a time stamp. The field definition is given below.

```
            editdate  TIMEDATE DEFAULT (datetime('now','localtime')),
```

Examples

```
Compute the current date.
 SELECT date('now');
 2009-11-05

Compute the current date & time.
 SELECT datetime('now');
 2009-11-05 20:01:07

Compute the last day of the current month.
 SELECT date('now','start of month','+1 month','-1 day');
 26

Compute the date and time given a unix timestamp 1092941466.
 SELECT datetime(1092941466, 'unixepoch');
 2009-11-05 20:01:07

Compute date and time given a unix timestamp 1092941466, and compensate for your local timezone.
 SELECT datetime(1092941466, 'unixepoch', 'localtime');
 2009-11-05 20:01:07

Compute the current unix timestamp.
 SELECT strftime('%s','now');
 1092941466

Subtract one timestamp from the other, use this only for dates after 1970!
 SELECT trip_id, start_time, end_time,STRFTIME('%s',end_time)-STRFTIME('%s',start_time) FROM t1; #
 2:56:22

Compute number of days since signing the US Declaration of Independence.
 SELECT julianday('now') - julianday('1776-07-04');
 76543

Compute the number of seconds since a particular moment in 2004:
 SELECT strftime('%s','now') - strftime('%s','2004-01-01 02:34:56');
 476977647

Compute the date of the first Tuesday in October for the current year.
 SELECT date('now','start of year','9 months','weekday 2');
 2009-10-05
```

```
Compute time since unix epoch in seconds (like strftime('%s','now') except includes fractional
part):
 SELECT (julianday('now') - 2440587.5)*86400.0;

Compute date in different format
 SELECT "Current day " || strftime('%d');
 Current day 05

Compute days till Christmas. The %j modifier gives the day of the year for the time string.
 SELECT 'Days to XMas:' || (strftime('%j', '2009-12-24') - strftime('%j', 'now'));
 Days to XMas:49

Save the current date & time to a database…
$query = "INSERT INTO 't1' ('name', 'mydate') VALUES ('John',datetime('now') )";
  or
$query = "INSERT INTO 't1' ('name', 'mydate') VALUES ('John',date("Y-m-d H-i-s") )";
2009-11-05 20:01:07
```

Listing: 12.6 –

SAVE CURRENT OR FUTURE DATE

Save the current date and calculate the date 30 days from now…

```
$query = "INSERT INTO 't1' ('datenow', 'date30') VALUES (datetime('now'),datetime('now', '30 days')
)";
```

Listing: 12.7 –

LESS THAN TIME SPAN

To get the rows where there is less than 30 days between two dates
where 'newdate' and 'olddate' are date fields…

```
$query = "SELECT * FROM 't1' WHERE julianday(newdate) <  julianday(olddate)+30";
```

Listing: 12.8 –

DATE EQUAL TO TIME SPAN

To get the rows where a date equals a span of 30 days,
where 'somedate' and 'otherdate' are date fields…

```
$query = "SELECT * FROM 't1' WHERE julianday(somedate) =  julianday(otherdate)+30";
```

Listing: 12.9 –

Time Zone

If the time zone is not set, PHP will report an error if “report all warnings” is enabled.

```
  date_default_timezone_set(date_default_timezone_get()); // default server timezone
or
  date_default_timezone_set('Pacific/Honolulu');    // utc-10 hawaiian
  date_default_timezone_set('America/Anchorage);    // utc-9  pacific
  date_default_timezone_set('America/Los_Angeles'); // utc-8  pacific
  date_default_timezone_set('America/Denver');      // utc-7  mountain
  date_default_timezone_set('America/Chicago');     // utc-6  central
```


Copyright © 2005-2015

```
date_default_timezone_set('America/New_York);      // utc-5  eastern
```

A complete list of time zone codes is available at http://en.wikipedia.org/wiki/List_of_tz_database_time_zones

The computation of local time depends heavily on the whim of politicians and is thus difficult to get correct for all locales. In this implementation, the standard C library function localtime_r() is used to assist in the calculation of local time. The localtime_r() C function normally only works for years between 1970 and 2037. For dates outside this range, SQLite attempts to map the year into an equivalent year within this range, do the calculation, then map the year back.

Errors and Bugs

The computation of local time depends heavily on the whim of politicians and is thus difficult to get correct for all locales. In this implementation, the standard C library function localtime_r() is used to assist in the calculation of local time. The localtime_r() C function normally only works for years between 1970 and 2037. For dates outside this range, SQLite attempts to map the year into an equivalent year within this range, do the calculation, then map the year back.

These functions only work for dates between 0000-01-01 00:00:00 and 9999-12-31 23:59:59 (julidan day numbers 1721059.5 through 5373484.5). For dates outside that range, the results of these functions are undefined.

Non-Vista Windows platforms only support one set of DST rules. Vista only supports two. Therefore, on these platforms, historical DST calculations will be incorrect. For example, in the US, in 2007 the DST rules changed. Non-Vista Windows platforms apply the new 2007 DST rules to all previous years as well. Vista does somewhat better getting results correct back to 1986, when the rules were also changed.

All internal computations assume the Gregorian calendar system. It is also assumed that every day is exactly 86400 seconds in duration.

End Of Section

Copyright © 2005-2015

Section – 15
Strings

List of String Functions

ascii(char)	return ascii code
char(code)	return char from ascii code
coalesce(*X*,*Y*,...)	Return a copy of the first non-NULL argument. If all arguments are NULL then NULL is returned. There must be at least 2 arguments.
concat(arg ?, arg ...?)	join args
concat_ws(sep, arg ?, arg?)	join args with sep
convert(str, from, to)	encoding converter. from and to must be tcl's encoding name.
elt(n, arg ?, arg ...?)	MySQL's elt.
glob(*X*,*Y*)	This function is used to implement the "X GLOB Y" syntax of SQLite. The sqlite3_create_function() interface can be used to override this function and thereby change the operation of the GLOB operator.
hex(n)	return hex value.
hex(*X*)	The argument is interpreted as a BLOB. The result is a hexadecimal rendering of the content of that blob.
ifnull(*X*,*Y*)	Return a copy of the first non-NULL argument. If both arguments are NULL then NULL is returned. This behaves the same as coalesce() above.
initcap(str)	Oracle's initcap.
insert(s, pos, len, ns)	
instr(str, sstr, ?st ?n??)	search sstr in str.
last_insert_rowid()	Return the ROWID of the last row insert from this connection to the database. This is the same value that would be returned from the sqlite_last_insert_rowid() API function.
left(str, n)	
length(str)	SQLite Built-in
length(*X*)	Return the string length of *X* in characters. If SQLite is configured to support UTF-8, then the number of UTF-8 characters is returned, not the number of bytes.
like(*X*,*Y*) like(*X*,*Y*,*Z*)	This function is used to implement the "X LIKE Y [ESCAPE Z]" syntax of SQL. If the optional ESCAPE clause is present, then the user-function is invoked with three arguments. Otherwise, it is invoked with two arguments only. The sqlite_create_function() interface can be used to override this function and thereby change the operation of the LIKE operator. When doing this, it may be important to override both the two and three argument versions of the like() function. Otherwise, different code may be called to implement the LIKE operator depending on whether or not an ESCAPE clause was specified.
load_extension(*X*)	Load SQLite extensions out of the shared library file named *X* using the entry point *Y*. The

Copyright © 2005-2015

load_extension(*X*,*Y*)	result is a NULL. If *Y* is omitted then the default entry point of sqlite3_extension_init is used. This function raises an exception if the extension fails to load or initialize correctly.
locate(sstr, str, pos)	return first sstr position in str. start from pos.
lower(str)	SQLite Built-in
lower(*X*)	Return a copy of string *X* will all characters converted to lower case. The C library tolower() routine is used for the conversion, which means that this function might not work correctly on UTF-8 characters.
lpad(str, len ?, pad?)	
ltrim(*X*) ltrim(*X*,*Y*)	Return a string formed by removing any and all characters that appear in *Y* from the left side of *X*. If the *Y* argument is omitted, spaces are removed.
max(*X*,*Y*,...)	Return the argument with the maximum value. Arguments may be strings in addition to numbers. The maximum value is determined by the usual sort order. Note that max() is a simple function when it has 2 or more arguments but converts to an aggregate function if given only a single argument.
mid(str, pos, len)	
min(*X*,*Y*,...)	Return the argument with the minimum value. Arguments may be strings in addition to numbers. The minimum value is determined by the usual sort order. Note that min() is a simple function when it has 2 or more arguments but converts to an aggregate function if given only a single argument.
nullif(*X*,*Y*)	Return the first argument if the arguments are different, otherwise return NULL.
position(sstr, IN, str)	return first sstr position in str.
quote(*X*)	This routine returns a string which is the value of its argument suitable for inclusion into another SQL statement. Strings are surrounded by single-quotes with escapes on interior quotes as needed. BLOBs are encoded as hexadecimal literals. The current implementation of VACUUM uses this function. The function is also useful when writing triggers to implement undo/redo functionality.
random(*)	Return a pseudo-random integer between -9223372036854775808 and +9223372036854775807.
randomblob(*N*)	Return a *N*-byte blob containing pseudo-random bytes. *N* should be a positive integer.
repeat(str, n)	
replace(*X*,*Y*,*Z*)	Return a string formed by substituting string *Z* for every occurrence of string *Y* in string *X*. The BINARY collating sequence is used for comparisons.
reverse(str)	reverse string.
right(str, n)	
round(*X*) round(*X*,*Y*)	Round off the number *X* to *Y* digits to the right of the decimal point. If the *Y* argument is omitted, 0 is assumed.
rpad(str, n ?, pad?)	
rtrim(str ?, chars?)	trimright chars(or space) from str.
rtrim(*X*)	Return a string formed by removing any and all characters that appear in *Y* from the right

rtrim(*X*,*Y*)	side of *X*. If the *Y* argument is omitted, spaces are removed.
soundex(*X*)	Compute the soundex encoding of the string *X*. The string "?000" is returned if the argument is NULL. This function is omitted from SQLite by default. It is only available the -DSQLITE_SOUNDEX=1 compiler option is used when SQLite is built.
space(n)	return space.
sqlite_version(*)	Return the version string for the SQLite library that is running. Example: "2.8.0"
substr(*X*,*Y*,*Z*)	Return a substring of input string *X* that begins with the *Y*-th character and which is *Z* characters long. The left-most character of *X* is number 1. If *Y* is negative the the first character of the substring is found by counting from the right rather than the left. If *X* is string then characters indices refer to actual UTF-8 characters. If *X* is a BLOB then the indices refer to bytes.
to_char(num, fmt)	Support only B,FM,S,G,D,",",9,0. Only number format. If you want to format datetime, use some sqlite built-in functions like strftime or datetime.
translate(str, from, to)	PostgreSQL's translate.
trim(*X*) trim(*X*,*Y*)	Return a string formed by removing any and all characters that appear in *Y* from both ends of *X*. If the *Y* argument is omitted, spaces are removed.
typeof(*X*)	Return the type of the expression *X*. The only return values are "null", "integer", "real", "text", and "blob". SQLite's type handling is explained in Datatypes in SQLite Version 3.
upper(*X*)	Return a copy of input string *X* converted to all upper-case letters. The implementation of this function uses the C library routine toupper() which means it may not work correctly on UTF-8 strings.
zeroblob(*N*)	Return a BLOB consisting of N bytes of 0x00. SQLite manages these zeroblobs very efficiently. Zeroblobs can be used to reserve space for a BLOB that is later written using incremental BLOB I/O.

sqlite_escape_string ($string)

sqlite_escape_string() will correctly quote the string specified by item for use in an SQLite SQL statement. This includes doubling up single-quote characters (') and checking for binary-unsafe characters in the query string. However, it does not catch all HTML characters that may conflict with a browser display. Notice the difference with the custome routine below.

Although the encoding makes it safe to insert the data, it will render simple text comparisons and LIKE clauses in your queries unusable for the columns that contain the binary data. In practice, this shouldn't be a problem, as your schema should be such that you don't use such things on binary columns (in fact, it might be better to store binary data using other means, such as in files).

```
$str = "Hello! I just don't <care>!";
$str = sqlite_escape_string($str);
echo $str;
```

Text Scrubbing

The following function will prevent foreign characters from being misinterpreted in text. It converts the most common punctuation characters into html equivalents.

```
function cleantxt($str) {
  $replace = array(
```


Copyright © 2005-2015

```
    '&' => '&amp;',
    '"' => '&quot;',
    "'" => '&#39;',
    ';' => '&#59;',
    '<' => '&lt;',
    '>' => '&gt;'
  );
  $str = str_replace(array_keys($replace), array_values($replace), $str);
  return $str;
}
```

Listing: 15.1 –

End Of Section

Copyright © 2005-2015

Section – 16
Math

List of Math Functions

abs(*X*)	Return the absolute value of argument *X*.
acos(x)	
asin(x)	
atan(x)	
atan2(x)	
avg(*X*)	Return the average value of all non null X within the group.
ceil(x)	
cos(x)	
cot(x)	
count(*)	The number of times X is non null in the group.
count(X)	The number of times X is non null in the group.
croup_count(X)	The number of times X is non null in the group.
croup_count(X,Y)	The number of times X is non null in the group.
degrees(x)	
exp(x)	
floor(x)	
greatest(x)	
least(x)	
log(x)	
log10(x)	
max(*X*)	
max(*X*,*Y*,...)	Return the argument with the maximum value. Arguments may be strings in addition to numbers. The maximum value is determined by the usual sort order. Note that max() is a simple function when it has 2 or more arguments but converts to an aggregate function if given only a single argument.
min(*X*)	
min(*X*,*Y*,...)	Return the argument with the minimum value. Arguments may be strings in addition to numbers. The minimum value is determined by the usual sort order. Note that min() is a simple function when it has 2 or more arguments but converts to an aggregate function if given only a single argument.
mod(x)	
pi(x)	
pow(x)	
radians(x)	
rand(x)	
random(*)	Return a pseudo-random integer between -9223372036854775808 and +9223372036854775807.
round(*X*) round(*X*,*Y*)	Round off the number *X* to *Y* digits to the right of the decimal point. If the *Y* argument is omitted, 0 is assumed.
sign(x)	
sin(x)	
sqrt(x)	
sum(X)	
tan(x)	
total(X)	
trunc(x)	
typeof(*X*)	Return the type of the expression *X*. The only return values are "null", "integer", "real", "text", and "blob". SQLite's type handling is explained in Datatypes in SQLite Version 3.

Copyright © 2005-2015

Math Examples

```
Examples
return the number of rows in the table. (includes NULL values)
 SELECT count(*) AS '# of rows' FROM Testing;
 # of rows
 ----------
 5

return the number of rows in table. (does not include NULL values)
 SELECT count(id) AS '# of non NULL values' FROM Testing;
 # of non NULL values
 --------------------
 3

Return average of values. (does not include NULL values)
 SELECT avg(cost) AS 'Average price' FROM Cars;
 Average price
 ----------------
 88528.1666666667

Return the sum() of values. (does not include NULL values)
 SELECT sum(OrderPrice) AS Sum FROM Orders;
 Sum
 ----------
 4930

  // add columns from two databases or tables, note: "as" is required
  $result = $db->query('SELECT  SUM(Total) -- SUM pulls total values
    FROM (
      SELECT SUM(smith) as Total FROM main.sales  -- "as" is required so sum can pull values
      UNION
      SELECT SUM(smith) as Total FROM lastyear.sales
    )');

Return maximum value of a column value
 SELECT max(cost) FROM t1;
 max(cost)
 ------------
 3500

Return minimum value of a column value
 SELECT min(cost) FROM t1;
 min(cost)
 ------------
 750

Return random number
 SELECT random() AS Random;
 Random
 --------------------
 1056892254869386643
```

Money

Sqlite does not support any special currency format. You can use integer (123.00) or 8-byte floating point number (123.45) with 15 significant digits.

Copyright © 2005-2015

End Of Section

Copyright © 2005-2015

Section - 17
Images - Saving and Displaying

GD command reference http://php.net/manual/en/book.image.php

The example shown here saves an image in a file to a database then retrieves and displays it. The database only has one row so that this example is not cluttered by *for* loops and multiple entries.

Test Image

While this document will not cover GD graphic images made with PHP, it is handy to be able to make a quick test image using the PHP GD image package. Also note that this program uses a simplified approach to displaying the picture in the browser using the header("Content-Type: image/png"). This program will only allow the picture to be show, so you will not be able to use PHP's **echo** command to add other text to the browser view.

To display HTML text on the same page as this image, the image must be saved as a file then the HTML <img…> tag used to import it. An example of how to do this is shown in a later example.

```
<?php
header("Content-Type: image/png");
$im = @imagecreate(200, 200) or die("Cannot Initialize new GD image stream");

#create colors
$white       = imagecolorallocate($im, 255, 255, 255);
$yellow      = imagecolorallocate($im, 255, 255, 200);
$red         = imagecolorallocate($im, 255,   0,   0);
$black       = imagecolorallocate($im,   0,   0,   0);

# Make the background transparent
imagecolortransparent($im, $white);

# Draw transparent background
imagefilledrectangle($im, 0, 0, 200, 200, $white);

imagesetthickness($im, 2);
imagefilledarc($im, 100, 100, 190, 190,  0, 360, $yellow, IMG_ARC_PIE); # head
imagearc($im, 100, 100, 190, 190,  0, 360, $black);                     # edge
imagefilledarc($im, 60, 75, 30, 20,  0, 360, $black, IMG_ARC_PIE);      # left eye
imagefilledarc($im, 140, 75, 30, 20,  0, 360, $black, IMG_ARC_PIE);     # right eye
imagesetthickness($im, 4);
imagearc($im, 100, 125, 125, 50,  0, 180, $red);                        # mouth
imagestring($im, 5, 67, 165,  "Testing!", $black);                      # text
imagepng($im); show in browser, picture only!
imagepng($im, 'test.png'); # save to file
imagedestroy($im); # recover memory
?>
```

Listing: 13.1 –

Copyright © 2005-2015

Saving an Image

To save an image, we need to have a special data type called a ***blob***. A ***blob*** is a signal to the database engine that it should not try to figure out what the data is; for example it is not a string with line ends. Because a database does not know what type of information is in a ***blob***, we have to keep tract of what it is. We should always have a field that keeps the media content type (png, jpg, gif, video, sound, etc). We will need this image type later so we can display it.

For this sample program we know what the image is. For simplicity sake we don't show the steps needed to retrieve the **content type** information. The **content type** field is created in this example for good practice. Here is an example of how a table might look for someone saving photographs.

```
# create table
  $query = "create table 'sample' (
    id          integer  primary key,
    description text(25),
    contenttype text(25),
    imagedata   blob
  )";
  $result = $db->query("$query") or die("Error in query: <span style='color:red;'>$query</span>");
```

Listing: 13.2 –

With a table properly designed, we can now save the data. For our example we assume that the picture is a **png** file named **"test.png".** We must use object oriented programming and a transaction (explained later) to do our work.

```
# insert row of data to table
  $stmt = $db->prepare("insert into sample (id, description, contenttype, imagedata)
    values (?,?,?,?)");

# assign data for each field
  $id        = NULL;
  $describe  = "Sample image file";  # row data
  $type      = "image/png";
  $imagefile = fopen('test.png', 'rb');

# build the transaction for each field
  $stmt->bindParam(1, $id);
  $stmt->bindParam(2, $describe);
  $stmt->bindParam(3, $type);
  $stmt->bindParam(4, $imagefile, PDO::PARAM_LOB);

# perform transaction
  $db->beginTransaction(); # create transaction
  $stmt->execute();        # do it
  $db->commit();           # finished
```

Listing: 13.3 –

That's it! The data with image has been saved.

Display an Image

To get the image from the database and display it, we must reverse the above process. We grab the row data using a normal query, then save the image data as a file that can be retrieved using the HTML <img…> tag.

```
# retrive row of data
  $query  = "SELECT * FROM 'sample'";
  $result = $db->query($query);  # get a row of data
  $row    = $result->fetch(PDO::FETCH_ASSOC); # assign to array value
```


Copyright © 2005-2015

```
# save image data to file
  $image   = $row['imagedata'];
  file_put_contents("temp.png", $image);
```

Listing: 13.4 –

Once the image is in a file that we can access, we use normal html code to display the image file.

```
# show data
  echo "<img src='temp.png'> <br/>";
  echo "Fig-1 <i>". $row['description'] ."</i><br/>";
```

Listing: 13.5 –

Here is a complete working program that assumes a file named "test.png" is in the current directory.

```
<?php
# setup
  /* $dbpath  = $_SERVER['DOCUMENT_ROOT'] . "/some path/"; */
  $dbpath = "";
  $dbfile  = "sqlite_image.sqlite3";
  $testimage = "test.png";
  date_default_timezone_set('America/Boise');
  $today = date("Y/m/d H:i:s"); # date w/leading zeros, dec 1 = 2010.12.01 00:00:00

# title
   echo '<body style="font-family:arial;">';
   echo "<h2>Blob Image</h2>";
   echo "$today<br><br>";

# check for image file
  if (!file_exists($testimage)) {
    echo "<i>image file &quot;$testimage&quot; does not exist.</i><br>";
    echo "<span style='color:red;'><i>I am quiting!</i></span><br>";
  exit;
  } else {
    echo "<i>image file found &quot;$testimage&quot;...</i><br>";
  }

# delete existing db
  if (file_exists("$dbpath$dbfile")) {
    unlink("$dbpath$dbfile");
    echo "<i>existing db &quot;$dbpath$dbfile&quot; removed.</i><br>";
  }

# new sqlite3 db and connection
  $query = "$dbpath$dbfile";
  try { $db = new PDO("sqlite:$query");
    echo "<i>new db created &quot;$dbfile&quot;...</i><br>";
    echo "<i>connecting...</i><br>"; }
  catch(PDOException $e) { echo $e->getMessage()." Error: <span style='color:red;'>$query</span>";
}

# enhanced errors
  $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

# create table
  $query = "create table 'sample' (
    id          integer  primary key,
    description text(25),
```


Copyright © 2005-2015

```
    contenttype text(25),
    imagedata   blob
  )";
  $result = $db->query("$query") or die("Error in query: <span style='color:red;'>$query</span>");
  echo "<i>new table and data initialized...</i><br>";

# setup transaction to insert row of data to table,
  $stmt = $db->prepare("insert into sample (id, description, contenttype, imagedata) values
(?,?,?,?)");

  $id         = NULL;
  $describe   = "Sample image file";
  $type       = "image/png";
  $imagefile = fopen($testimage, 'rb');

# perform transaction
  $stmt->bindParam(1, $id);
  $stmt->bindParam(2, $describe);
  $stmt->bindParam(3, $type);
  $stmt->bindParam(4, $imagefile, PDO::PARAM_LOB);

  $db->beginTransaction(); # create transaction
  $stmt->execute();        # do it
  $db->commit();           # finished
  echo "<i>data saved...</i><br>";

# retrive row of data
  $query   = "SELECT * FROM 'sample'";
  $result = $db->query($query);                # get row of data
  $row     = $result->fetch(PDO::FETCH_ASSOC); # assign to array of values
  $image   = $row['imagedata'];
  echo "<i>data retrieved...</i><br><br>";

# save image data to file
  file_put_contents("temp.png", $image);

# show data
  echo "<img src='temp.png'> <br/>";
  echo "Fig-1 <i>". $row['description'] ."</i><br/>";

#  close db, PHP will also do this automatically
   $db = NULL;

?>
```

Listing: 13.6 – complete image program

End of Section

Copyright © 2005-2015

Section - 18
Crypt, Hash, etc

Function Syntax

Requires Tcllib and Trf

md5(data ?, binary?)	MD5
md5_hmac(key, data ?, binary?)	HMAC-MD5
md5_crypt(pass, salt ?, binary?)	BSD compatible MD5 crypt
apr_crypt(pass, salt ?, binary?)	Apache compatible MD5 crypt
sha1(data ?, binary?)	SHA1
sha1_hmac(key, data ?, binary?)	HMAC-SHA1
aes_encrypt(key, data ?, binary?)	ASE encryption
aes_decrypt(key, data ?, binary?)	ASE decryption
blowfish_encrypt(key, data ?, binary?)	Blowfish encryption
blowfish_decrypt(key, data ?, binary?)	Blowfish decryption
des_encrypt(key, data ?, binary?)	DES encryption
des_decrypt(key, data ?, binary?)	DES decryption
base64_encode(data ?, binary?)	Base64 encode
base64_decode(data ?, binary?)	Base64 decode
compress(data ?, binary?)	compress data by zip
decompress(data ?, binary?)	decompress data by zip
uuid()	

Listing: 14.1 –

File IO

write_file(path, data ?,binary?)	Base64 decode
read_file(path ?,binary?)	Base64 encode

Listing: 14.2 –

End of Section

Copyright © 2005-2015

Section - 19
Printing Table Data

Formatted HTML Table

```
<?php
# show table data in formatted html table
  $result = $db->query("select * from $dbtable where 1 ");    // table data
  $rows    = $result->fetchall(PDO::FETCH_ASSOC);              // table data to array
  $html="";

  foreach ($rows as $row) {
    $rowtext = implode("||",$row)."<br>";                     // add || between array values
    $result = preg_replace("/\|\|/", "</td><td>", $rowtext); // replace || with <td>...
    $html = $html."<tr><td>".$result."</td></tr>\n";           // concatenate each row
  }

  echo "<table style='border-collapse:collapse;'
          cellpadding=3 border=1>\n$html</table>";
?>
```

1	Donald	Disney	1934	admin	2015-05-24 17:51:12
2	Goofy	Disney	1932	admin	2015-05-24 17:51:12
3	Heckle/Jeckle	Terrytoons	1942	admin	2015-05-24 17:51:12
4	Micky	Disney	1928	admin	2015-05-24 17:51:12
5	Minnie	Disney	1942	admin	2015-05-24 17:51:12
6	Yosemite Sam	Warner Brothers	1945	admin	2015-05-24 17:51:12

Table print

The following code creates the formatted table output shown below. Notice that the programmer must hard code the number of rows in the loop. Hard coding the loop simplifies the readability of the code.

```
// output table data to a simple html table...
echo "<table cellpadding=5 border=1>";
echo "<tr>";      # bold table header
echo "   <th>ID</th>";
echo "   <th>NAME</th>";
echo "   <th>DOB</th>";
echo "</tr>";

$result = $db->query("SELECT * FROM 't1' ");
foreach($result as $row)
{
   echo "<tr>";  // table data
   echo "<td>".$row['id']."</td>";
   echo "<td>".$row['name']."</td>";
   echo "<td>".$row['dob']."</td>";
   echo "</tr>";
}
echo "</table>";
```

ID	NAME	DOB
1	Frank	2010.10.19
2	John	1972.09.22
3		62.03.15
6	John	
10	Henry	1973.12.01

Simple print_r

The following code creates the text output shown here. This is the simplest and most often used method to print data information.

```
// show data
$query = "SELECT * FROM t1";
   $result = $db->query("$query") or die("Error in query:  <span style='color:red;'>$query</span>");
   $rows = $result->fetchall();

 //  print_r output
 echo "<br><span style='color:red;'><h1>Print_r Output</h1></span>";
 echo "<pre>"; print_r($rows); echo "</pre>";
```

```
Print_r Output

Array
(
    [id] => 1
    [0] => 1
    [name] => Frank
    [1] => Frank
    [dob] => 2010.10.19
    [2] => 2010.10.19
)

Array
(
    [id] => 2
    [0] => 2
    [name] => John
    [1] => John
    [dob] => 1972.09.22
    [2] => 1972.09.22
)
```

Data output continues...

Loop print_r

The following code creates the text output shown here. While the output looks identical to the previous simple routine, this modified version has the advantage of giving access to each row of information within the 'for' loop.

```
/  show data
$query = "SELECT * FROM t1";
   $result = $db->query("$query") or die("Error in query:  <span style='color:red;'>$query</span>");
   $rows = $result->fetchall();

   // get each row as an array and print values
   //  print_r output
   echo "<br><span style='color:red;'><h1>Print_r Output</h1></span>";
   foreach($rows as $array) {
```


Copyright © 2005-2015

```
      echo "<pre>";
      print_r($array);
      echo "</pre>";
   }
```

Print_r Output

```
Array
(
    [id] => 1
    [0] => 1
    [name] => Frank
    [1] => Frank
    [dob] => 2010.10.19
    [2] => 2010.10.19
)

Array
(
    [id] => 2
    [0] => 2
    [name] => John
    [1] => John
    [dob] => 1972.09.22
    [2] => 1972.09.22
)
```

Data output continues...

Var dump

The next code snippet creates the text output shown here. Notice that fetchall(PDO::FETCH_ASSOC) was used. This suppresses the numbered array index definition which causes a cluttered appearance.

```
#  show data
$query = "SELECT * FROM t1";
   $result = $db->query("$query") or die("Error in query:  <span style='color:red;'>$query</span>");
   $rows = $result->fetchall(PDO::FETCH_ASSOC);

   # get each row as an array and print values
   #  print_r output
   echo "<br><span style='color:red;'><h1>Var Dump Output</h1></span>";
   echo "<pre>";
   var_dump($result);
   var_dump($rows);
   echo "</pre>";
```


Copyright © 2005-2015

Var Dump Output

```
object(PDOStatement)#2 (1) {
  ["queryString"]=>
  string(16) "SELECT * FROM t1"
}
array(5) {
  [0]=>
  array(3) {
    ["id"]=>
    string(1) "1"
    ["name"]=>
    string(5) "Frank"
    ["dob"]=>
    string(10) "2010.10.19"
  }
  [1]=>
  array(3) {
    ["id"]=>
    string(1) "2"
    ["name"]=>
    string(4) "John"
    ["dob"]=>
    string(10) "1972.09.22"
  }
```

Data output continues...

Index & associative table output

The next code snippet creates the text output shown here. This creates both the index and associtive header columns for the data. Of interest is placeing the use of fetchall(PDO::FETCH_ASSOC) fetchall(PDO::FETCH_NUM) will re-format the table to show the simpler information properly

```
#  show data
  $query = "SELECT * FROM t1";
  $result = $db->query("$query") or die("Error in query:  <span style='color:red;'>$query</span>");
  $rows = $result->fetchall(); # usually use fetchall(PDO::FETCH_ASSOC)

#  Table Output, get each row as an array and print values
  echo "<br><span style='color:red;'><h1>Pretty Table Output</h1></span>";
  echo "<table border=1 width=\"100%\">\n";  # open the table
  print "<tr>\n";

  foreach ($rows[0] as $key => $useless){   # add the table headers
    print "<th>$key</th>";
  }

  print "</tr>";
  foreach ($rows as $row){      # display data
    print "<tr>";
    foreach ($row as $key => $val){
       print "<td>$val</td>";
    }
    print "</tr>\n";
  }
  print "</table><br>";   # close the table
```

Pretty Table Output

id	0	name	1	dob	2
1	1	Frank	Frank	2010.10.19	2010.10.19
2	2	John	John	1972.09.22	1972.09.22
3	3			62.03.15	62.03.15
6	6	John	John		
10	10	Henry	Henry	1973.12.01	1973.12.01

End Of Section

Copyright © 2005-2015

Appendix - A
Test Databases

Test database program

The following code creates test databases.

```
<?php
/* Initalize "test.sqlite3" database */
   echo '<body style="font-family:arial;">';
   echo '<h2>Initalize "test.sqlite3" database</h2>';

#  setup
   $dbFile  = "test.sqlite3";
   $today   = date("Y.m.d"); # date w/leading zeros, dec 1 = 2010.12.01

#  if db already exists delete it
   unlink("$dbFile");

#  create SQLite3 database
   $query = "$dbFile";
   try { $db = new PDO("sqlite:$query"); }
   catch(PDOException $e) { echo $e->getMessage()." Error: <span style='color:red;'>$query</span>";
}

#  set the PDO error mode to EXCEPTION, gives lots of information
   $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

#   create t1 table
$query = "CREATE TABLE 't1' (
   'id' INTEGER PRIMARY KEY,
   'name' TEXT(10),
   'dob' DATE
)";
   $result = $db->query("$query") or die("Error in query:  <span
style='color:red;'>$query</span>");
   echo "<i>Created 't1' table...<br></i> ";

#   create t2 table
$query = "CREATE TABLE 't2' (
   'id' INTEGER PRIMARY KEY,
   'name' TEXT(10),
   'phone' TEXT(15)
)";
   $result = $db->query("$query") or die("Error in query:  <span
style='color:red;'>$query</span>");
   echo "<i>Created 't2' table ...<br></i> ";

#   fill t1 data
$query = "INSERT INTO 't1' ('id','name','dob')
         SELECT '1' ,'Frank' ,'$today'
   UNION SELECT '2' ,'John'  ,'1972.09.22'
   UNION SELECT '3' ,''      ,'62.03.15'
   UNION SELECT '6' ,'John'  ,''
   UNION SELECT '10','Henry' ,'1973.12.01'
";
```


Copyright © 2005-2015

```
   $result = $db->query("$query") or die("Error in query:  <span
style='color:red;'>$query</span>");
   echo "<i>Created 't1' data... </i> ";

#  show number of rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected <br>";


#   fill t2 data
$query = "INSERT INTO 't2' ('id','name','phone')
         SELECT '12' ,'Frank' ,'(100)-123-4567'
   UNION SELECT '32' ,'John'  ,'200-234-5678'
   UNION SELECT '28' ,''      ,'890-1234'
   UNION SELECT '66' ,'John'  ,''
   UNION SELECT '61' ,'Henry' ,'ph. 321.2112'
";
   $result = $db->query("$query") or die("Error in query:  <span
style='color:red;'>$query</span>");
   echo "<i>Created 't2' data... </i> ";

   #  Return rows affected by last operation
   $rows_affected = $result->rowCount();
   echo "<span style='color:red;'>$rows_affected</span> records affected ";

/* +----------------------------------------+
   |          Test Code Goes Here           |
   +----------------------------------------+  */

#  show data
$query = "SELECT * FROM t1";
   $result = $db->query("$query") or die("Error in query:  <span
style='color:red;'>$query</span>");
   $rows = $result->fetchAll();
   echo "<pre>";
   print_r($rows);
   echo "</pre>";
   echo "Access array row information, 2nd row DOB info...<br>";
   echo "Row ID = ".$rows['1']['id']." DOB = ".$rows['1']['dob']."<br><br>"; # array [0,1,...]

   #  close db, PHP will also do this  automatically
   $db = NULL;
?>
```

End Of Section

Copyright © 2005-2015

Appendix – B
Command Reference

PDO Commands

- Introduction
- Installing/Configuring
    - Requirements
    - Installation
    - Runtime Configuration
    - Resource Types
- Predefined Constants
- Connections and Connection management
- Transactions and auto-commit
- Prepared statements and stored procedures
- Errors and error handling
- Large Objects (LOBs)
- PDO — The PDO class
    - PDO::beginTransaction — Initiates a transaction
    - PDO::commit — Commits a transaction
    - PDO::construct — Creates a PDO instance representing a connection to a database
    - PDO::errorCode — Fetch the SQLSTATE associated with the last operation on the database handle
    - PDO::errorInfo — Fetch extended error information associated with the last operation on the database handle
    - PDO::exec — Execute an SQL statement and return the number of affected rows
    - PDO::getAttribute — Retrieve a database connection attribute
    - PDO::getAvailableDrivers — Return an array of available PDO drivers
    - PDO::lastInsertId — Returns the ID of the last inserted row or sequence value
    - PDO::prepare — Prepares a statement for execution and returns a statement object
    - PDO::query — Executes an SQL statement, returning a result set as a PDOStatement object
    - PDO::quote — Quotes a string for use in a query.
    - PDO::rollBack — Rolls back a transaction
    - PDO::setAttribute — Set an attribute
- PDOStatement — The PDOStatement class
    - PDOStatement->bindColumn — Bind a column to a PHP variable
    - PDOStatement->bindParam — Binds a parameter to the specified variable name
    - PDOStatement->bindValue — Binds a value to a parameter
    - PDOStatement->closeCursor — Closes the cursor, enabling the statement to be executed again.
    - PDOStatement->columnCount — Returns the number of columns in the result set
    - PDOStatement->debugDumpParams — Dump an SQL prepared command
    - PDOStatement->errorCode — Fetch the SQLSTATE associated with the last operation on the statement handle
    - PDOStatement->errorInfo — Fetch extended error information associated with the last operation on the statement handle
    - PDOStatement->execute — Executes a prepared statement
    - PDOStatement->fetch — Fetches the next row from a result set
    - PDOStatement->fetchAll — Returns an array containing all of the result set rows
    - PDOStatement->fetchColumn — Returns a single column from the next row of a result set
    - PDOStatement->fetchObject — Fetches the next row and returns it as an object.
    - PDOStatement->getAttribute — Retrieve a statement attribute

        - PDOStatement->getColumnMeta — Returns metadata for a column in a result set
        - PDOStatement->nextRowset — Advances to the next rowset in a multi-rowset statement handle
        - PDOStatement->rowCount — Returns the number of rows affected by the last SQL statement
        - PDOStatement->setAttribute — Set a statement attribute
        - PDOStatement->setFetchMode — Set the default fetch mode for this statement
- PDOException — The PDOException class
- PDO Drivers
    - MS SQL Server (PDO) — Microsoft SQL Server and Sybase Functions (PDO_DBLIB)
    - Firebird/Interbase (PDO) — Firebird/Interbase Functions (PDO_FIREBIRD)
    - IBM (PDO) — IBM Functions (PDO_IBM)
    - Informix (PDO) — Informix Functions (PDO_INFORMIX)
    - MySQL (PDO) — MySQL Functions (PDO_MYSQL)
    - Oracle (PDO) — Oracle Functions (PDO_OCI)
    - ODBC and DB2 (PDO) — ODBC and DB2 Functions (PDO_ODBC)
    - PostgreSQL (PDO) — PostgreSQL Functions (PDO_PGSQL)
    - SQLite (PDO) — SQLite Functions (PDO_SQLITE)
    - 4D (PDO) — 4D Functions (PDO_4D)

SQLite3 Commands

- Introduction
- Installing/Configuring
    - Requirements
    - Installation
    - Runtime Configuration
    - Resource Types
- Predefined Constants
- SQLite3 — The SQLite3 class
    - SQLite3::changes — Returns the number of database rows that were changed (or inserted or deleted) by the most recent SQL statement
    - SQLite3::close — Closes the database connection
    - SQLite3::__construct — Instantiates an SQLite3 object and opens an SQLite 3 database
    - SQLite3::createAggregate — Registers a PHP function for use as an SQL aggregate function
    - SQLite3::createFunction — Registers a PHP function for use as an SQL scalar function
    - SQLite3::escapeString — Returns a string that has been properly escaped
    - SQLite3::exec — Executes a result-less query against a given database
    - SQLite3::lastErrorCode — Returns the numeric result code of the most recent failed SQLite request
    - SQLite3::lastErrorMsg — Returns English text describing the most recent failed SQLite request
    - SQLite3::lastInsertRowID — Returns the row ID of the most recent INSERT into the database
    - SQLite3::loadExtension — Attempts to load an SQLite extension library
    - SQLite3::open — Opens an SQLite database
    - SQLite3::prepare — Prepares an SQL statement for execution
    - SQLite3::query — Executes an SQL query
    - SQLite3::querySingle — Executes a query and returns a single result
    - SQLite3::version — Returns the SQLite3 library version as a string constant and as a number
- SQLite3Stmt — The SQLite3Stmt class
    - SQLite3Stmt::bindParam — Binds a parameter to a statement variable

Copyright © 2005-2015

    - SQLite3Stmt::bindValue — Binds the value of a parameter to a statement variable
    - SQLite3Stmt::clear — Clears all current bound parameters
    - SQLite3Stmt::close — Closes the prepared statement
    - SQLite3Stmt::execute — Executes a prepared statement and returns a result set object
    - SQLite3Stmt::paramCount — Returns the number of parameters within the prepared statement
    - SQLite3Stmt::reset — Resets the prepared statement
- SQLite3Result — The SQLite3Result class
    - SQLite3Result::columnName — Returns the name of the nth column
    - SQLite3Result::columnType — Returns the type of the nth column
    - SQLite3Result::fetchArray — Fetches a result row as an associative or numerically indexed array or both
    - SQLite3Result::finalize — Closes the result set
    - SQLite3Result::numColumns — Returns the number of columns in the result set
    - SQLite3Result::reset — Resets the result set back to the first row

---

End Of Section

Copyright © 2005-2015

Appendix – C
Reserved Words (337)

A

ABSOLUTE, ACTION, ADD, AFTER, ALL, ALLOCATE, ALTER, AND, ANY, ARE, ARRAY, AS, ASC, ASENSITIVE, ASSERTION, ASYMMETRIC, AT, ATOMIC, AUTHORIZATION, AVG

B

BEFORE, BEGIN, BETWEEN, BIGINT, BINARY, BIT, BIT_LENGTH, BLOB, BOOLEAN, BOTH, BREADTH, BY

C

CALL, CALLED, CASCADE, CASCADED, CASE, CAST, CATALOG, CHAR, CHARACTER, CHARACTER_LENGTH, CHAR_LENGTH, CHECK, CLOB, CLOSE, COALESCE, COLLATE, COLLATION, COLUMN, COMMIT, CONDITION, CONNECT, CONNECTION, CONSTRAINT, CONSTRAINTS, CONSTRUCTOR, CONTAINS, CONTINUE, CONVERT, CORRESPONDING, COUNT, CREATE, CROSS, CUBE, CURRENT, CURRENT_DATE, CURRENT_DEFAULT_TRANSFORM_GROUP, CURRENT_PATH, CURRENT_ROLE, CURRENT_TIME, CURRENT_TIMESTAMP, CURRENT_TRANSFORM_GROUP_FOR_TYPE, CURRENT_USER, CURSOR, CYCLE

D

DATA, DATE, DAY, DEALLOCATE, DEC, DECIMAL, DECLARE, DEFAULT, DEFERRABLE, DEFERRED, DELETE, DEPTH, DEREF, DESC, DESCRIBE, DESCRIPTOR, DETERMINISTIC, DIAGNOSTICS, DISCONNECT, DISTINCT, DO, DOMAIN, DOUBLE, DROP, DYNAMIC

E

EACH, ELEMENT, ELSE, ELSEIF, END, EQUALS, ESCAPE, EXCEPT, EXCEPTION, EXEC, EXECUTE, EXISTS, EXIT, EXTERNAL, EXTRACT

F

FALSE, FETCH, FILTER, FIRST, FLOAT, FOR, FOREIGN, FOUND, FREE, FROM, FULL, FUNCTION

G

GENERAL, GET, GLOBAL, GO, GOTO, GRANT, GROUP, GROUPING

H

HANDLER, HAVING, HOLD, HOUR

I

IDENTITY, IF, IMMEDIATE, IN, INDICATOR, INITIALLY, INNER, INOUT, INPUT, INSENSITIVE, INSERT, INT, INTEGER, INTERSECT, INTERVAL, INTO, IS, ISOLATION, ITERATE

J

JOIN

K

KEY

L

LANGUAGE, LARGE, LAST, LATERAL, LEADING, LEAVE, LEFT, LEVEL, LIKE, LOCAL, LOCALTIME, LOCALTIMESTAMP, LOCATOR, LOOP, LOWER

M

MAP, MATCH, MAX, MEMBER, MERGE, METHOD, MIN, MINUTE, MODIFIES, MODULE, MONTH, MULTISET

N

NAMES, NATIONAL, NATURAL, NCHAR, NCLOB, NEW, NEXT, NO, NONE, NOT, NULL, NULLIF, NUMERIC

O

OBJECT, OCTET_LENGTH, OF, OLD, ON, ONLY, OPEN, OPTION, OR, ORDER, ORDINALITY, OUT, OUTER, OUTPUT, OVER, OVERLAPS

P

PAD, PARAMETER, PARTIAL, PARTITION, PATH, POSITION, PRECISION, PREPARE, PRESERVE, PRIMARY, PRIOR, PRIVILEGES, PROCEDURE, PUBLIC

R

RANGE, READ, READS, REAL, RECURSIVE, REF, REFERENCES, REFERENCING, RELATIVE, RELEASE, REPEAT, RESIGNAL, RESTRICT, RESULT, RETURN, RETURNS, REVOKE, RIGHT, ROLE, ROLLBACK, ROLLUP, ROUTINE, ROW, ROWS

S

SAVEPOINT, SCHEMA, SCOPE, SCROLL, SEARCH, SECOND, SECTION, SELECT, SENSITIVE, SESSION, SESSION_USER, SET, SETS, SIGNAL, SIMILAR, SIZE, SMALLINT, SOME, SPACE, SPECIFIC, SPECIFICTYPE, SQL, SQLCODE, SQLERROR, SQLEXCEPTION, SQLSTATE, SQLWARNING, START, STATE, STATIC, SUBMULTISET, SUBSTRING, SUM, SYMMETRIC, SYSTEM, SYSTEM_USER

T

TABLE, TABLESAMPLE, TEMPORARY, THEN, TIME, TIMESTAMP, TIMEZONE_HOUR, TIMEZONE_MINUTE, TO, TRAILING, TRANSACTION, TRANSLATE, TRANSLATION, TREAT, TRIGGER, TRIM, TRUE

U

UNDER, UNDO, UNION, UNIQUE, UNKNOWN, UNNEST, UNTIL, UPDATE, UPPER, USAGE, USER, USING

V

VALUE, VALUES, VARCHAR, VARYING, VIEW

W

WHEN, WHENEVER, WHERE, WHILE, WINDOW, WITH, WITHIN, WITHOUT, WORK, WRITE

Y

YEAR

Z

ZONE

Copyright © 2005-2015

Notes

Notes

Copyright © 2005-2015

Notes

Copyright © 2005-2015

Documenation

http://www.php.net/

http://www.sqlite.org/

http://phpbuilder.com/manual/en/ref.sqlite.php

http://www.w3schools.com/sql/

http://www.firstsql.com/

References

- Getting the Right Data with SQL Joins. http://www.devx.com/dbzone/Article/17403/0/page/1
- SQL Join. http://www.w3schools.com/sql/sql_join.asp
- SQLite home page, http://www.sqlite.org/
- SQL Features That SQLite Does Not Implement, http://www.sqlite.org/omitted.html
- A command-line program to administer SQLite databases, http://www.sqlite.org/sqlite.html
- SQL As Understood By SQLite, http://www.sqlite.org/lang.html
- Frequently Asked Questions, http://www.sqlite.org/faq.html
- SQLite Reference by Mike Chirico mchirico@users.sourceforge.net, http://souptonuts.sourceforge.net/readme_sqlite_reference.html