SQLITE PERL TUTORIAL

Installation

The SQLite3 can be integrated with Perl using Perl DBI module, which is a database access module for the Perl programming language. It defines a set of methods, variables and conventions that provide a standard database interface.

Here are simple steps to install DBI module on your Linux/UNIX machine:

```
$ wget http://search.cpan.org/CPAN/authors/id/T/TI/TIMB/DBI-1.625.tar.gz
$ tar xvfz DBI-1.625.tar.gz
$ cd DBI-1.625
$ perl Makefile.PL
$ make
$ make install
```

If you need to install SQLite driver for DBI, then it can be installed as follows:

```
$ wget http://search.cpan.org/CPAN/authors/id/M/MS/MSERGEANT/DBD-SQLite-1.11.tar.gz
$ tar xvfz DBD-SQLite-1.11.tar.gz
$ cd DBD-SQLite-1.11
$ perl Makefile.PL
$ make
$ make install
```

DBI Interface APIs

Following are important DBI routines which can suffice your requirement to work with SQLite database from your Perl program. If you are looking for a more sophisticated application, then you can look into Perl DBI official documentation.

S.N.	API & Description
1	**DBI->connect***$data_source, "" , "" , %attr* Establishes a database connection, or session, to the requested $data_source. Returns a database handle object if the connection succeeds. Datasource has the form like : **DBI:SQLite:dbname='test.db'** SQLite is SQLite driver name and test.db is the name of SQLite database file. If the filename is given as **':memory:'**, it will create an in-memory database in RAM that lasts only for the duration of the session. If filename is actual device file name, then it attempts to open the database file by using its value. If no file by that name exists then a new database file by that name gets created. You keep second and third paramter as blank strings and last parameter is to pass various attributes as shown below in the example.
2	*dbh* − > *do*(**sql**) This routine prepares and executes a single SQL statement. Returns the number of rows affected or undef on error. A return value of -1 means the number of rows is not known, not applicable, or not available. Here $dbh is a handle returned by DBI->connect() call.

3

dbh − > _prepare_(sql)

This routine prepares a statement for later execution by the database engine and returns a reference to a statement handle object.

4

$sth->execute()

This routine performs whatever processing is necessary to execute the prepared statement. An undef is returned if an error occurs. A successful execute always returns true regardless of the number of rows affected. Here, *sthisastatementhandlereturnedby*dbh->prepare*$sql* call.

5

$sth->fetchrow_array()

This routine fetches the next row of data and returns it as a list containing the field values. Null fields are returned as undef values in the list.

6

$DBI::err

This is equivalent to *h* − > *err*, *where*h is any of the handle types like *dbh*, sth, or $drh. This returns native database engine error code from the last driver method called.

7

$DBI::errstr

This is equivalent to *h* − > *errstr*, *where*h is any of the handle types like *dbh*, sth, or $drh. This returns the native database engine error message from the last DBI method called.

8

$dbh->disconnect()

This routine closes a database connection previously opened by a call to DBI->connect.

Connecting To Database

Following Perl code shows how to connect to an existing database. If database does not exist, then it will be created and finally a database object will be returned.

```perl
#!/usr/bin/perl

use DBI;
use strict;

my $driver   = "SQLite";
my $database = "test.db";
my $dsn = "DBI:$driver:dbname=$database";
my $userid = "";
my $password = "";
my $dbh = DBI->connect($dsn, $userid, $password, { RaiseError => 1 })
                      or die $DBI::errstr;

print "Opened database successfully\n";
```

Now, let's run above program to create our database test.db in the current directory. You can change your path as per your requirement. Keep above code in sqlite.pl file and execute it as shown below. If database is successfully created, then it will give the following message:

```
$ chmod +x sqlite.pl
$ ./sqlite.pl
Open database successfully
```

Create a Table

Following Perl program will be used to create a table in previously created database:

```
#!/usr/bin/perl

use DBI;
use strict;

my $driver   = "SQLite";
my $database = "test.db";
my $dsn = "DBI:$driver:dbname=$database";
my $userid = "";
my $password = "";
my $dbh = DBI->connect($dsn, $userid, $password, { RaiseError => 1 })
                      or die $DBI::errstr;
print "Opened database successfully\n";

my $stmt = qq(CREATE TABLE COMPANY
      (ID INT PRIMARY KEY     NOT NULL,
       NAME           TEXT    NOT NULL,
       AGE            INT     NOT NULL,
       ADDRESS        CHAR(50),
       SALARY         REAL););
my $rv = $dbh->do($stmt);
if($rv < 0){
   print $DBI::errstr;
} else {
   print "Table created successfully\n";
}
$dbh->disconnect();
```

When above program is executed, it will create COMPANY table in your test.db and it will display the following messages:

```
Opened database successfully
Table created successfully
```

NOTE: in case you see following error in any of the operation:

```
DBD::SQLite::st execute failed: not an error(21) at dbdimp.c line 398
```

In this case you will have open dbdimp.c file available in DBD-SQLite installation and find out **sqlite3_prepare** function and change its third argument to -1 instead of 0. Finally install DBD::SQLite using **make** and do **make install** to resolve the problem.

INSERT Operation

Following Perl program shows how we can create records in our COMPANY table created in above example:

```
#!/usr/bin/perl

use DBI;
use strict;

my $driver   = "SQLite";
my $database = "test.db";
my $dsn = "DBI:$driver:dbname=$database";
my $userid = "";
my $password = "";
my $dbh = DBI->connect($dsn, $userid, $password, { RaiseError => 1 })
```

```
                        or die $DBI::errstr;
print "Opened database successfully\n";

my $stmt = qq(INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
      VALUES (1, 'Paul', 32, 'California', 20000.00 ));
my $rv = $dbh->do($stmt) or die $DBI::errstr;

$stmt = qq(INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
      VALUES (2, 'Allen', 25, 'Texas', 15000.00 ));
$rv = $dbh->do($stmt) or die $DBI::errstr;

$stmt = qq(INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
      VALUES (3, 'Teddy', 23, 'Norway', 20000.00 ));
$rv = $dbh->do($stmt) or die $DBI::errstr;

$stmt = qq(INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY)
      VALUES (4, 'Mark', 25, 'Rich-Mond ', 65000.00 ););
$rv = $dbh->do($stmt) or die $DBI::errstr;

print "Records created successfully\n";
$dbh->disconnect();
```

When above program is executed, it will create given records in COMPANY table and will display the following two lines:

```
Opened database successfully
Records created successfully
```

SELECT Operation

Following Perl program shows how we can fetch and display records from our COMPANY table created in above example:

```
#!/usr/bin/perl

use DBI;
use strict;

my $driver   = "SQLite";
my $database = "test.db";
my $dsn = "DBI:$driver:dbname=$database";
my $userid = "";
my $password = "";
my $dbh = DBI->connect($dsn, $userid, $password, { RaiseError => 1 })
                        or die $DBI::errstr;
print "Opened database successfully\n";

my $stmt = qq(SELECT id, name, address, salary  from COMPANY;);
my $sth = $dbh->prepare( $stmt );
my $rv = $sth->execute() or die $DBI::errstr;
if($rv < 0){
   print $DBI::errstr;
}
while(my @row = $sth->fetchrow_array()) {
      print "ID = ". $row[0] . "\n";
      print "NAME = ". $row[1] ."\n";
      print "ADDRESS = ". $row[2] ."\n";
      print "SALARY =  ". $row[3] ."\n\n";
}
print "Operation done successfully\n";
$dbh->disconnect();
```

When above program is executed, it will produce the following result:

```
Opened database successfully
ID = 1
NAME = Paul
```

```
ADDRESS = California
SALARY =  20000

ID = 2
NAME = Allen
ADDRESS = Texas
SALARY =  15000

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY =  20000

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY =  65000

Operation done successfully
```

UPDATE Operation

Following Perl code shows how we can use UPDATE statement to update any record and then fetch and display updated records from our COMPANY table:

```
#!/usr/bin/perl

use DBI;
use strict;

my $driver   = "SQLite";
my $database = "test.db";
my $dsn = "DBI:$driver:dbname=$database";
my $userid = "";
my $password = "";
my $dbh = DBI->connect($dsn, $userid, $password, { RaiseError => 1 })
                      or die $DBI::errstr;
print "Opened database successfully\n";

my $stmt = qq(UPDATE COMPANY set SALARY = 25000.00 where ID=1;);
my $rv = $dbh->do($stmt) or die $DBI::errstr;
if( $rv < 0 ){
   print $DBI::errstr;
}else{
   print "Total number of rows updated : $rv\n";
}
$stmt = qq(SELECT id, name, address, salary  from COMPANY;);
my $sth = $dbh->prepare( $stmt );
$rv = $sth->execute() or die $DBI::errstr;
if($rv < 0){
   print $DBI::errstr;
}
while(my @row = $sth->fetchrow_array()) {
      print "ID = ". $row[0] . "\n";
      print "NAME = ". $row[1] ."\n";
      print "ADDRESS = ". $row[2] ."\n";
      print "SALARY =  ". $row[3] ."\n\n";
}
print "Operation done successfully\n";
$dbh->disconnect();
```

When above program is executed, it will produce the following result:

```
Opened database successfully
Total number of rows updated : 1
ID = 1
NAME = Paul
ADDRESS = California
```

```
SALARY =  25000

ID = 2
NAME = Allen
ADDRESS = Texas
SALARY =  15000

ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY =  20000

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY =  65000

Operation done successfully
```

DELETE Operation

Following Perl code shows how we can use DELETE statement to delete any record and then fetch and display remaining records from our COMPANY table:

```
#!/usr/bin/perl

use DBI;
use strict;

my $driver   = "SQLite";
my $database = "test.db";
my $dsn = "DBI:$driver:dbname=$database";
my $userid = "";
my $password = "";
my $dbh = DBI->connect($dsn, $userid, $password, { RaiseError => 1 })
                      or die $DBI::errstr;
print "Opened database successfully\n";

my $stmt = qq(DELETE from COMPANY where ID=2;);
my $rv = $dbh->do($stmt) or die $DBI::errstr;
if( $rv < 0 ){
   print $DBI::errstr;
}else{
   print "Total number of rows deleted : $rv\n";
}
$stmt = qq(SELECT id, name, address, salary  from COMPANY;);
my $sth = $dbh->prepare( $stmt );
$rv = $sth->execute() or die $DBI::errstr;
if($rv < 0){
   print $DBI::errstr;
}
while(my @row = $sth->fetchrow_array()) {
      print "ID = ". $row[0] . "\n";
      print "NAME = ". $row[1] ."\n";
      print "ADDRESS = ". $row[2] ."\n";
      print "SALARY =  ". $row[3] ."\n\n";
}
print "Operation done successfully\n";
$dbh->disconnect();
```

When above program is executed, it will produce the following result:

```
Opened database successfully
Total number of rows deleted : 1
ID = 1
NAME = Paul
ADDRESS = California
SALARY =  25000
```

```
ID = 3
NAME = Teddy
ADDRESS = Norway
SALARY =  20000

ID = 4
NAME = Mark
ADDRESS = Rich-Mond
SALARY =  65000

Operation done successfully
```

Processing math: 100%