PERL SCRIPTING FOR SQLITE

Slides by Laura Liparulo

Http://lauraliparulo.altervista.org

Overview

- Introduction to SQLite
- Introduction to Perl scripting
- Comprehensive Perl Archive Network (CPAN)
- Perl Database Interface Module (DBI)
- Perl scripting for SQLite

INTRODUCTION TO SQLITE

What is SQLite? 1/2

- ***embedded*** SQL database engine;
- the most widely distributed database in history;
- used throughout the majority of mobile devices including iPods, iPads etc.;
- If you use Firefox, Google Chrome, Skype you are a consumer of SQLite!

What is SQLite? 2/2

•an ***ACID***-compliant embedded relational database management system contained in a relatively small (~275 kB) C programming library;

• it stores the entire database as a single cross-platform file on a host machine, which has the ***file extension .db****;*

•SQLite implements SQL functions using ***callbacks*** to C-language routines. Even the built-in SQL functions are implemented this way.

History of SQLite

SQLite was designed by **Richard Hipp** in the spring of 2000 while working as a software engineer for *General Dynamics* on contract with the *United States Navy*.
Hipp was designing software used on board guided missile destroyer ships.
The design goals of SQLite were to allow the program to be operated without installing a database management system or administration.

Who uses SQLite?

SQLite is a popular choice for local/client SQL storage within a web browser and within a rich internet application framework.
Most notably the leaders in this area embed SQLite.

Well-Known Users of SQLite

SQLite Smartphones adoption

- Google's Android
- Apple iOS
- RIM's BlackBerry
- Linux Foundation's MeeGo
- Palm's webOS
- Symbian OS
- Nokia's Maemo

SQLite Apple adoption

Apple adopted it as an option in Mac OS X's Core Data API from the original implementation in Mac OS X 10.4 onwards (Apple Mail, Safari, and in Aperture), and also for administration of videos and songs on the iPhone and iPod.
(where it is used for the SMS/MMS, Calendar, Call history and Contacts storage).
Also in iTunes software.

SQLite Web browsers adoption



- Mozilla Firefox and Mozilla Thunderbird store a variety of configuration data (bookmarks, cookies, contacts etc.)
- Opera browser
- Google's Chrome browser

Adobe and Ruby on Rails adoption

- Adobe Systems uses SQLite as its file format in Adobe Photoshop Lightroom, a standard database in Adobe AIR, and internally within Adobe Reader.

- Ruby on Rails' default database management system is also SQLite.

SQLite features 1/3

- Small. (< 300 kB). No dependencies;
- Simple: single data file with a portable format;
- True relational database transactions. Rich subset of SQL92;
- ACID;
- High performance;
- Safer than UNIX tools for related tables.

SQLite feature 2/3

- APIs in C, C++, Tcl, Python, ODBC, Java, PHP...
- Portable. Widely used, stable, solid;
- Large databases;
- Open source (public domain);
- significantly faster (as much as 10 or 20×) than the default PostgreSQL, MySQL for most common operations.

SQLite feature 3/3

- ***backup*** as simple as copying one file***;***

- ***dynamically typed*****:** instead of assigning a type to a column as in most SQL database systems, types are assigned to individual values;

- ***Weakly typed:*** a string can be inserted into an integer column. This adds flexibility to columns, especially when bound to a dynamically typed scripting language.

SQLite limitations

- No access control (i.e., embeddable);
- Foreign key constants must be implemented with triggers;
- No type checking;
- No scaled integers. No date and time types;
- No nested transactions;
- Other minor SQL limitations.

Why a Relational Database?

- Normalized data structures are stable, minimize redundancy and bugs;
- Simple data integrity protects from GUI bugs;
- Ad hoc queries are a powerful diagnostic tool.

Why embedding a Database?

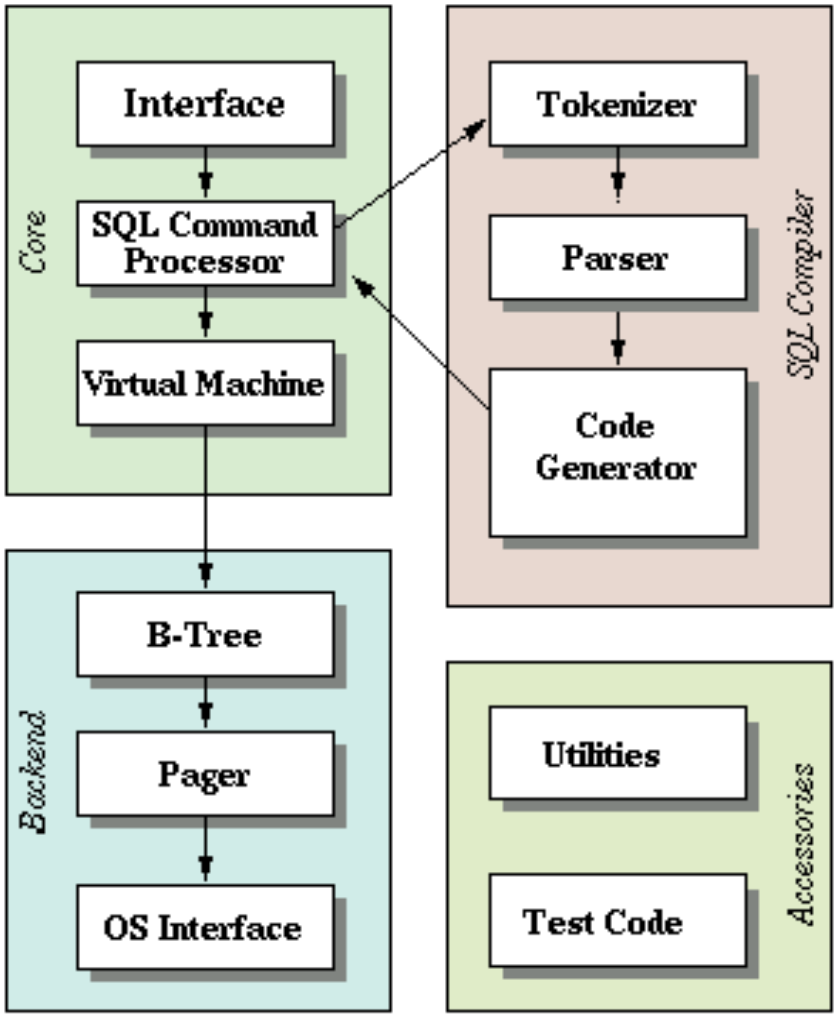
- A simple back end for a web server;
- Improve performance for client-server applications;
- Persistent data. No need to “save.”;
- Transactions protect data from corruption.

SQLite 3

A standalone program that can be used to:

- create a database;
- define tables;
- insert and change rows;
- run queries;
- manage a SQLite database file.

SQLite 3.0 library architecture 1/3



- **Interface**

implemented by functions found in the ***main.c***, ***legacy.c***, and ***vdbeapi.c*** source files.

The **sqlite3_get_table()** routine is implemented in **table.c**. **sqlite3_mprintf()** is found in **printf.c**. **sqlite3_complete()** is in **tokenize.c**. The Tcl interface is implemented by **tclsqlite.c**.

- **Tokenizer**

breaks the original string from the interface up into tokens and pass those tokens one by one to the parser. Source file: ***tokenize.c.***

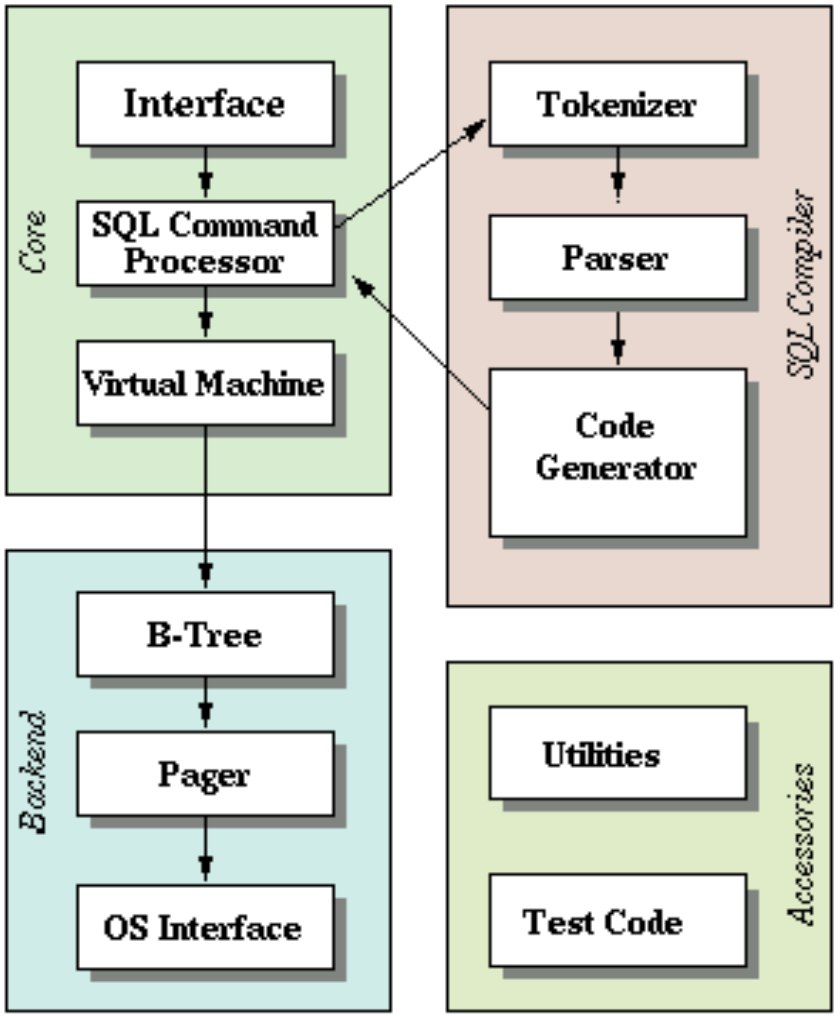
- **Parser**

using the Lemon LALR(1) parser generator, which is concurrent and thread-safe.

- **Code Generator**

producing virtual machine code that will do the work that the SQL statements request. Files in the code generator: **attach.c**, **auth.c**, **build.c**, **delete.c**, **expr.c**, **insert.c**, **pragma.c**, **select.c**, **trigger.c**, **update.c**, **vacuum.c** and **where.c**.

SQLite 3.0 library architecture 2/3



- **Virtual machine**

implements an abstract computing engine specifically designed to manipulate database files. The machine has a stack which is used for intermediate storage.

entirely contained in a single source file *vdbe.c*i with its own header files: *vdbe.h* that defines an interface between the virtual machine and the rest of the SQLite library and **vdbeInt.h** which defines structure private the virtual machine

- **B-tree**

found in the *btree.c* source file. A separate B-tree is used to maintain each table and index of the database on disk. All B-trees are stored in the same disk file.

The interface to the B-tree subsystem is defined by the header file *btree.h*.

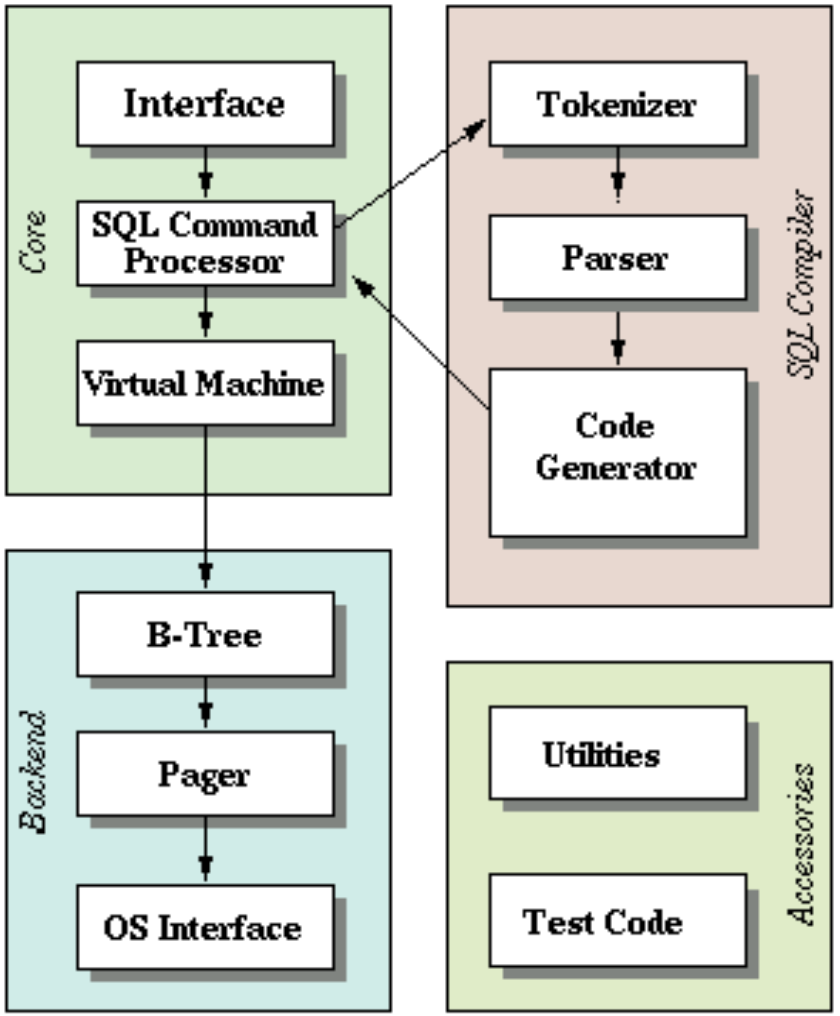
- **Page cache**

responsible for reading, writing, and caching fixed-size chunks, the B-tree module requests information. The default chunk size is 1024 bytes but can vary between 512 and 65536 bytes.

The page cache also provides the rollback and atomic commit abstraction and takes care of locking of the database file.

The code to implement the page cache is contained in the single C source file *pager.c*. The interface to the page cache subsystem is defined by the header file ***pager.h***.

SQLite 3.0 library architecture 3/3



- **Os interface**

abstraction layer to interface with the operating system, defined in ***os_unix.c*** for Unix, ***os_win.c*** for Windows, etc. Each of these operating-specific implements typically has its own header file: ***os_unix.h, os_win.h***, etc.

- **Utilities**

Memory allocation and caseless string comparison routines are located in ***util.c***. Symbol tables used by the parser are maintained by hash tables found in ***hash.c***. The ***utf.c*** source file contains Unicode conversion subroutines. SQLite has its own private implementation of **printf()** (with some extensions) in ***printf.c*** and its own random number generator in ***random.c***.

- **Test code**

more than half the total code base of SQLite is devoted to testing. There are many **assert()** statements in the main code files. In additional, the source files ***test1.c*** through ***test5.c*** together with ***md5.c*** implement extensions used for testing purposes only. The ***os_test.c*** backend interface is used to simulate power failures to verify the crash-recovery mechanism in the pager.

Installing SQLite on Linux

On UBUNTU, DEBIAN, etc:

:~$ sudo apt-get install sqlite3

On OPEN-SUSE:

:~$ sudo zypper install sqlite3

On REDHAT, CentOS, or FEDORA:

:~$ yum install SQLite3

Alternatively you can download it and install manually at:

http://www.sqlite.org/download.html

SQLite: creating a new database

Let's create a new database by shell:

:~$ sqlite3 test.db create table user
(id integer primary key, name text, surname text);

Let’s fill it:

:~$ sqlite3 test.db “insert into user(name, surname)
values (‘linus’,'torvalds’);”
:~$ sqlite3 test.db “insert into user(name,surname)
values(‘richard’, ‘stallman’)”;

SQLite query by bash

To check it out:

*:~$sqlite3 test.db “select * from n”;*

The result is:

1|linus|torvalds

2|richard|stallman

Sqlite3 enviroment

Alternatively you can create a database entering the sqlite3 enviroment:

:~$ sqlite3 test.db
SQLite version 3.0.8
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>

Anyway it's better to work in the shell prompt directly, that allows you to run scripts.

SQLite backup: „dump“ command

The ***.dump*** command shows information about all the changes performed onto the database.

$ sqlite3 test.db “.dump”

The result is:

PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE n(id INTEGER PRIMARY KEY, f TEXT, l TEXT);
INSERT INTO “n” VALUES(1,’linus’,'torvalds’);
INSERT INTO “n” VALUES(2,’richard’,'stallman’);
COMMIT

If you want to backup the database in a new file, you can specify a name (ex. “dbbackup”):

$ $ sqlite3 test.db ‘.dump’ > dbbackup

SQLite backup: „sed“ command

The contents of the backup can be filtered and piped to another database.

Below, table t1 is changed to t2 with the sed command, and it is piped into the test2.db database.

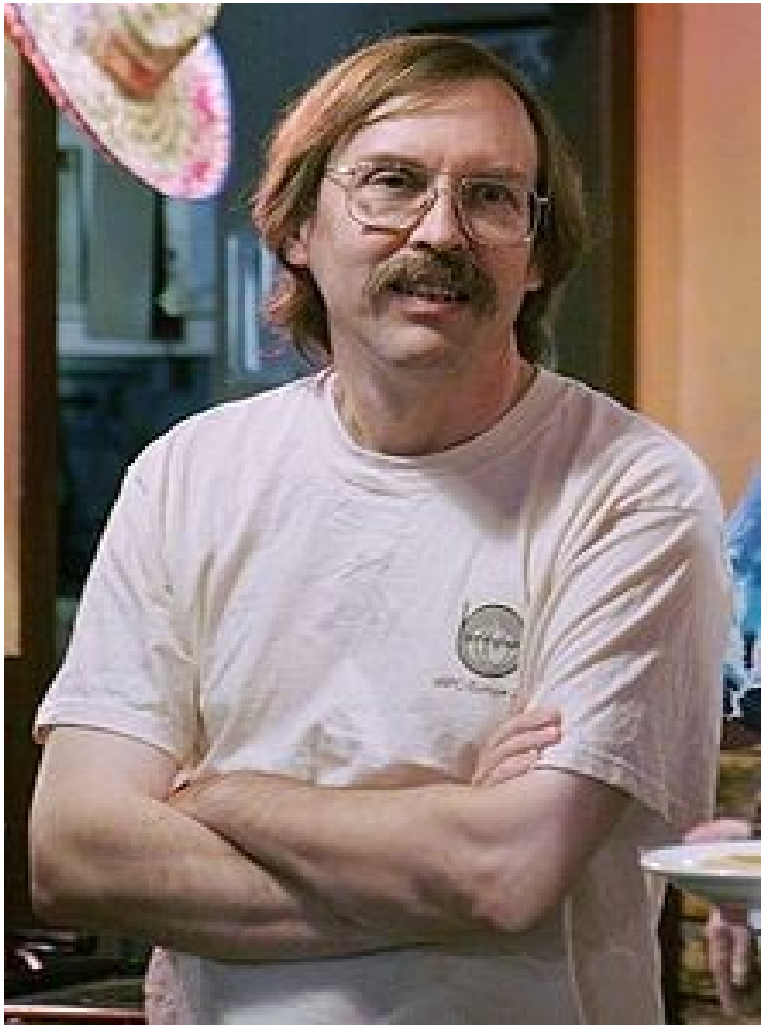
$ sqlite3 test.db ".dump"|sed -e s/t1/t2/|sqlite3 test2.db

INTRODUCTION TO PERL SCRIPTING

What is Perl?

- "Practical Extraction and Report Language";
- **interpreted** programming language;
- used for a wide range of tasks: system administration, web development, network programming, GUI development, and more;
- language for open source code.

History of Perl



- Originally created in 1987 by **Larry Wall,** while working at als Unisys.
- Wall was unhappy by the functionality that sed, C, awk and the Bourne Shell offered him. He looked for a language that will combine all of their best features, while having as few disadvantages of its own.
- Perl became especially popular as a language for writing ***server-side scripts*** for web-servers.
- commonly used for system administration tasks, managing database data, as well as writing GUI applications.

Perl features

- supports both procedural and object-oriented (OO) programming;
- a small amount of code goes a long way
- easy to use (plain text)
- runs much faster than bash scripting (file.sh)
- many built-in features (ex. text processing)
- ***Type friendly*** (no need for explicit casting)
- Basic syntax is C and PHP like
- very high ***portability*** across Web servers.

Perl versus C

- Perl program may be around 1/3 to three-quarters as long as the corresponding program in C;

•Perl faster to write, faster to read, faster to maintain...

•number of bugs in a program proportional to the length of the source code: fewer bugs on average

•optimized for problems which are about 90% working with text(regular expressions)

Creating a simple Perl script

Make a file called „helloWorld.pl“ :

:~$ touch web_code.pl

Then make it executable:

:~$ sudo chmod +x web_code.pl

„Web code“ Perl script

```
#!/usr/bin/perl -w
# script to download the .html page of the website passed as argument by shell

use LWP::UserAgent;
use HTTP::Request;
use HTTP::Response;
use URI;


my $uris = shift || "http://www.google.com";
my $uri = URI->new($uris)->canonical;
$uris = $uri->as_string;
print "$uris -- HTTP RESPONSE\n";

#userAgent: to send and retrieve messages
my $ua = LWP::UserAgent->new;
my $request = HTTP::Request->new(GET => $uri);

my $scheme = $uri->scheme;

my $authority = $uri->authority;
my $path = $uri->path;
my $response = $ua->request($request);
print $response->content;
print "\n protocol: $scheme, path: $path, website: $authority \n";
```

Running „web_code.pl“

To run a Perl program from the Unix command line:

$ perl web_code.pl http://ubuntu.com

Alternatively, put the path to your perl executable in the ***she-bang*** as the first line of your script:

#!/usr/bin/perl

Then run the perl script simply:

$ ./web_code.pl http://ubuntu.com

The result is the html code printed on the terminal.

COMPREHENSIVE PERL ARCHIVE NETWORK

Comprehensive Perl Archive Network 1/2

- CPAN, archive of nearly 100,000 ***modules*** of software written in Perl, as well as documentation for it;
- Online-Repository for Perl-Module;
- About 300 sites that all replicate the same contents around the globe;
- ***Module:*** one file (which extension is **.pm*) contain common routines used by several programs (class library)

Comprehensive Perl Archive Network 2/2

- resources easily accessible with the CPAN.pm module;
- to help programmers locate modules and programs not included in the Perl standard distribution.
- Modules Network: Perl programming Authors Upload Server (PAUSE)
- a Perl ***core module*** which is used to download and install Perl software from the CPAN archive.

PERL DATABASE INTERFACE MODULE (DBI)

PERL DBI

- Database Interface Module in Perl;

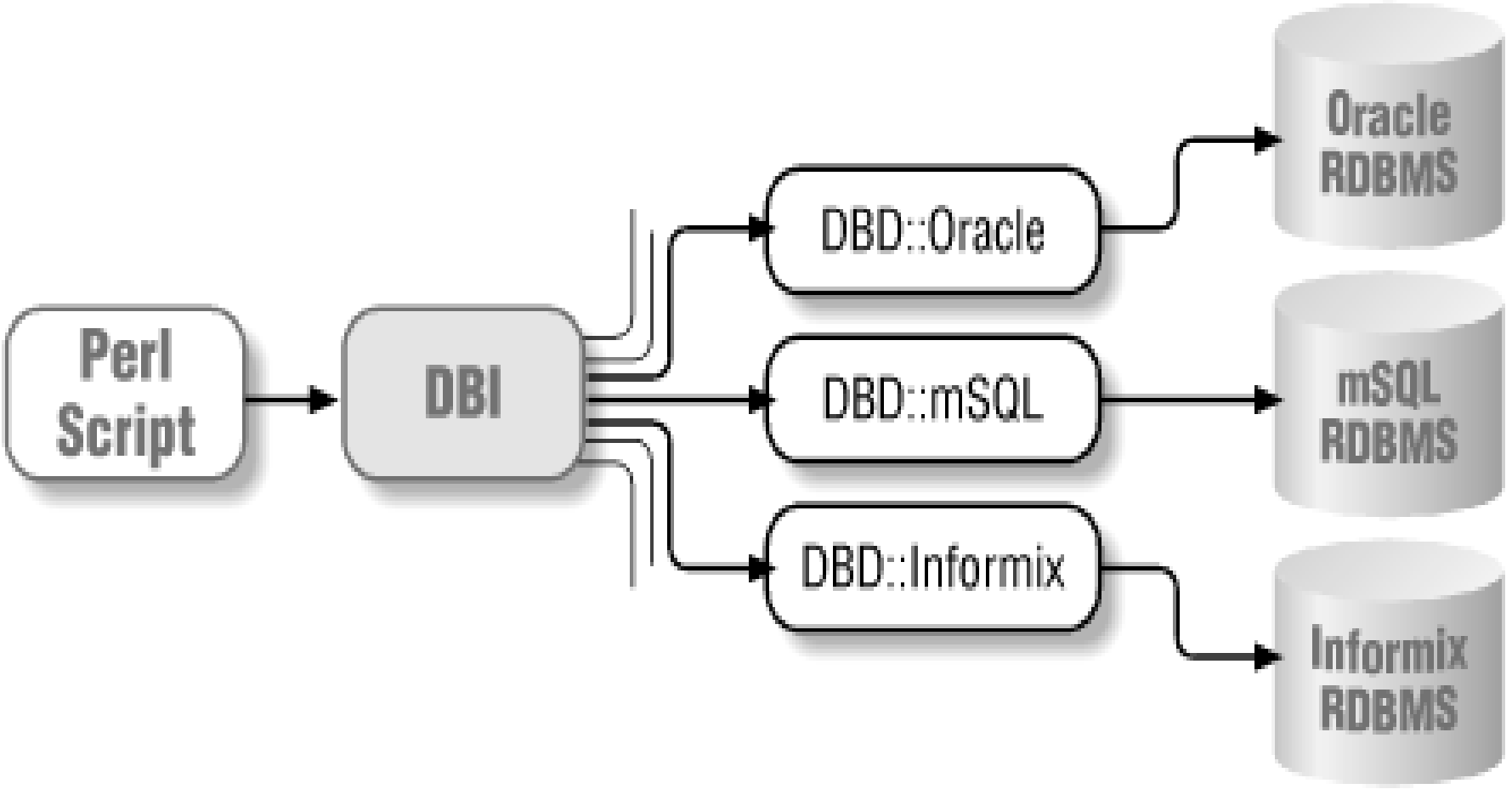
- included as core module;

- DBI knows how to locate and load in DBD (`Database Driver') modules.

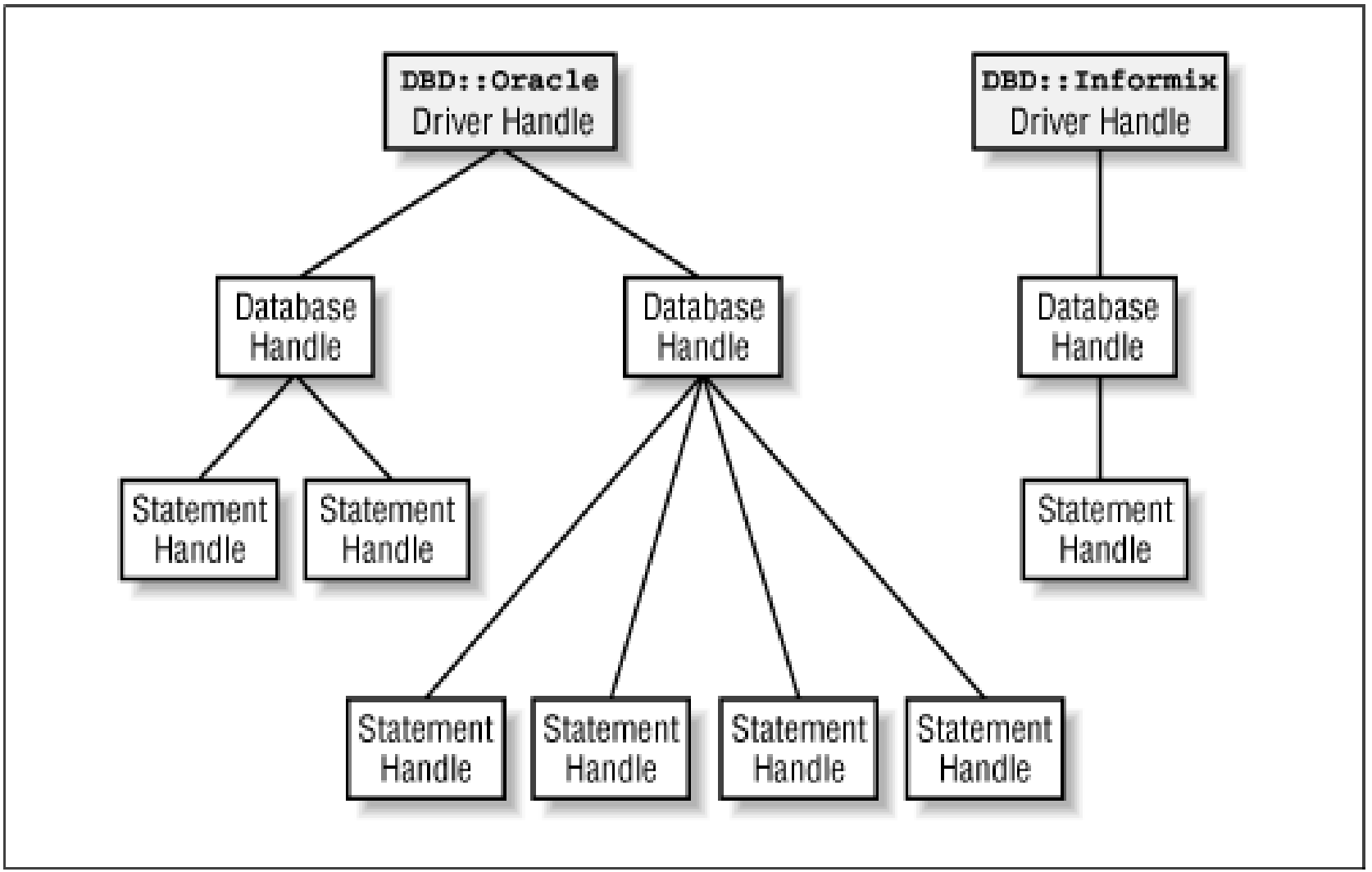
- DBD modules have the vendor libraries (es. ***dbi:SQLite***) in them and know how to talk to the real databases;

- one DBD module for every different database.

Perl DBI architecture I



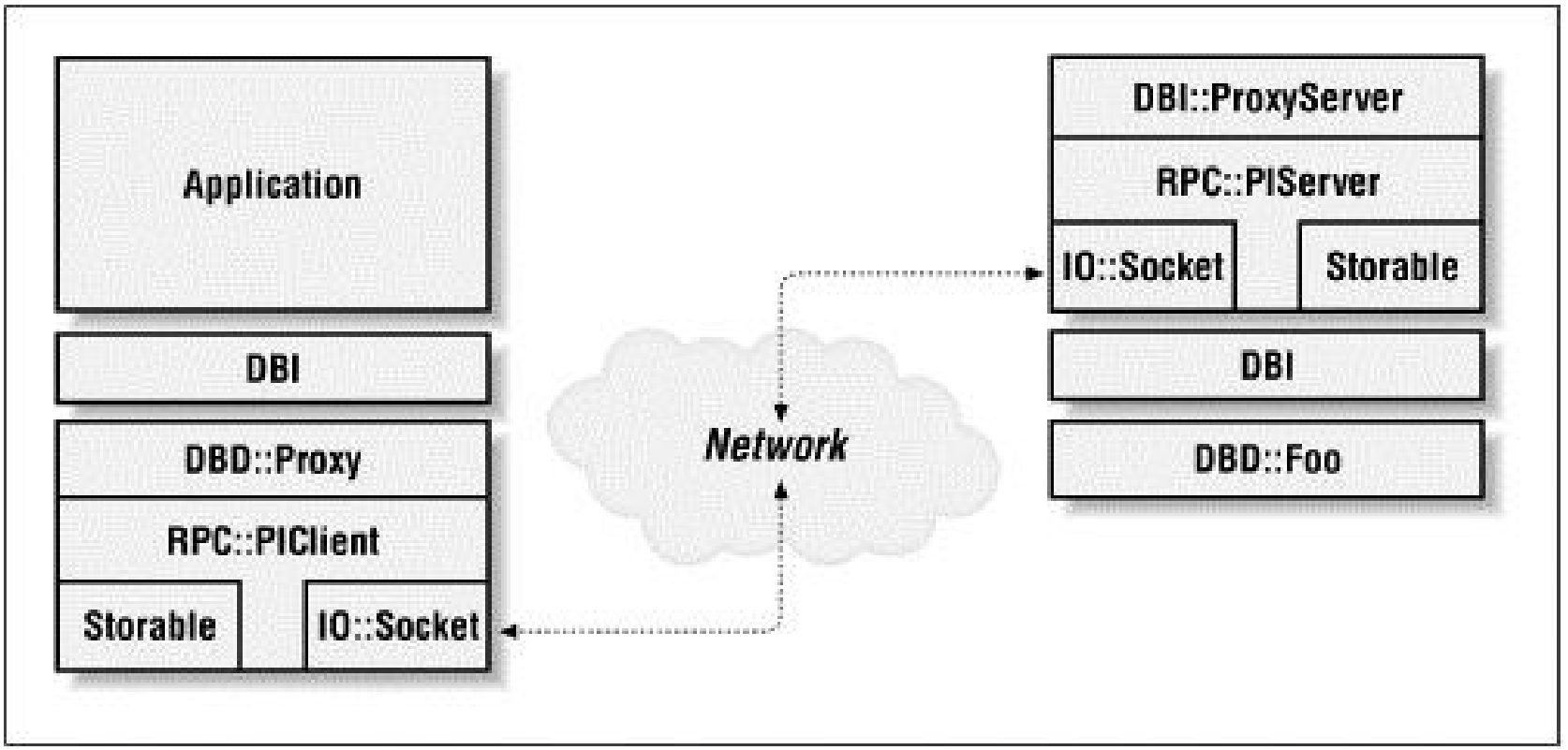
Perl DBI architecture II



DBI proxying

Perl DBI supports database proxying through two modules:

- ***DBD::Proxy***, used by client programs to talk to a proxy server;
- ***DBI::ProxyServer***, implementing the server.



Perl DBI script overview

```
#!/usr/bin/perl

use strict;
use DBI; # dbi module

my $database = "test";
my $hostname = "localhost";
my $dsn  = "DBI:mysql:database=$database;host=$hostname";
my $user = "testuser"; my $pass = "userpass";

# dbh = database handler
# connecting to the database by DBI
my $dbh = DBI::->connect( $dsn, $user, $pass,
                { 'RaiseError' => 1, 'AutoCommit' => 1 } )
                    or die DBI::errstr;

          # handling the database.

# to disconnect fromthe database
$dbh->disconnect; .
```

Driver handles

- ***$drh*** : driver handle
- ***@drh = DBI->available_drivers();***

  list of the available DB-drivers (ex. copy of a Microsoft Access database(DBD::ODBC) to a Oracle database (DBD::Oracle).
- ***@drh = DBI->datasource();***

DBI Driver handles script

```
#!/usr/bin/perl -w

use DBI;
### Probe DBI for the installed drivers
my @drh = DBI->available_drivers();
die "No drivers found!\n" unless @drivers; # should never happen

### Iterate through the drivers and list the data sources for each one
foreach my $driver ( @drh) {
print "Driver: $driver\n";
my @dataSources = DBI->data_sources( $driver );
foreach my $dataSource ( @dataSources ) {
print "\tData Source is $dataSource\n"; }
print "\n";
}
exit;
```

DBI Database handles

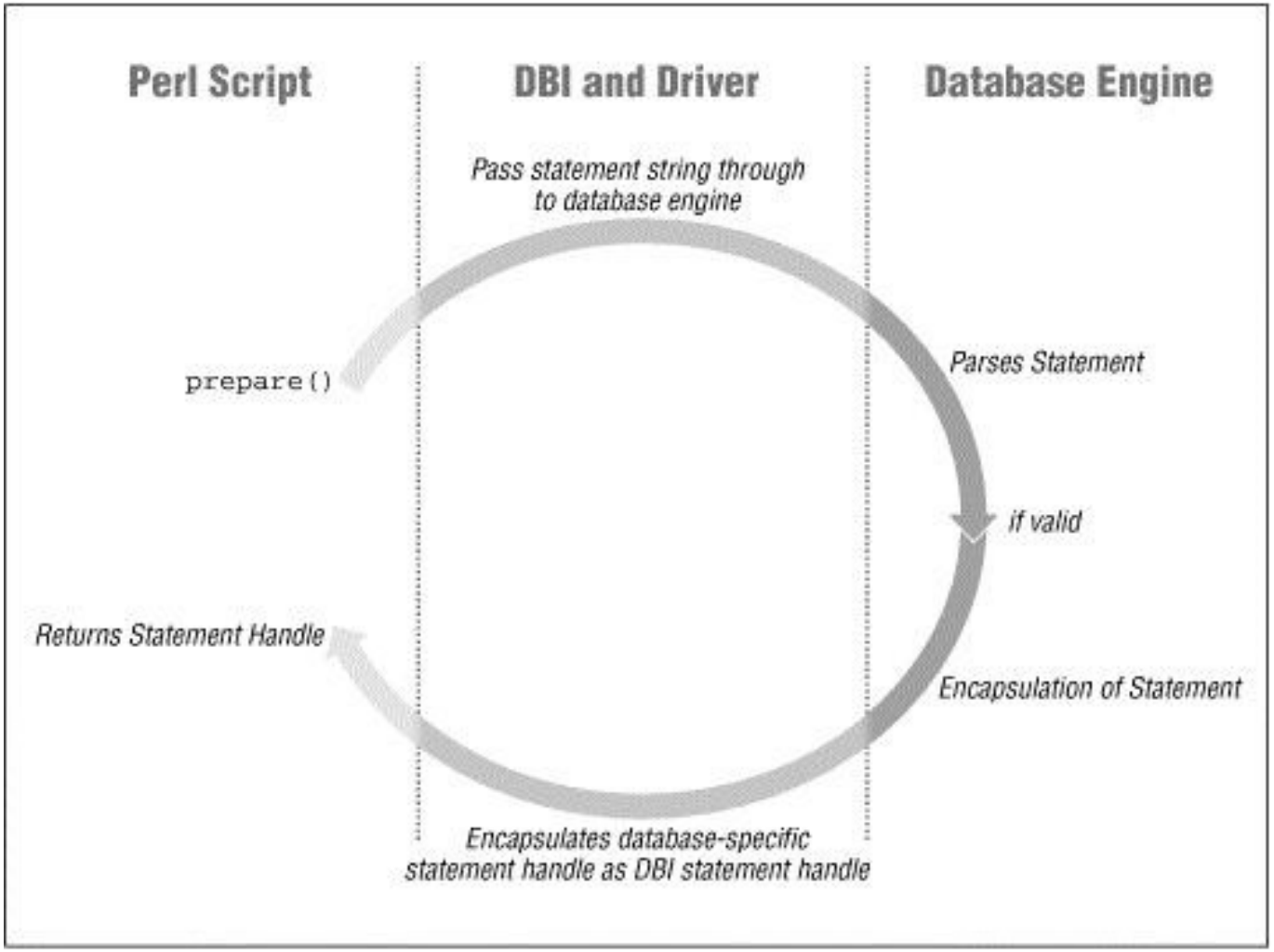
- ***$dbh*** : database handle
- ***$dbh = DBI->connect($data_source, $username, $auth,***

  \%attr);
- ***dbh->disconnect();***

DBI Statement handle

- ***$sth = $dbh->prepare($query)***
- ***$sth->execute($query)***
- ***$sth->fetchrow_array()***
- ***$sth->finish();***
- Unlimited amount of statements pro database.

Statement handles DBI data flow



Statements handling 1/2

```
# PREPARE THE CREATE STATEMENT
my $query = "CREATE TABLE if not exists student (
        id INTEGER PRIMARY KEY,
        first_name TEXT,
        last_name TEXT,
        email TEXT) ";

my $sth = $dbh->prepare($query) or die("Cannot prepare: " .
DBI::errstr() );

# EXECUTE THE CREATE STATEMENT
my $ret = $sth->execute() or die("Cannot execute: " . DBI::errstr() );
```

Statements handling 2/2

Perl DBI allows to perform statements directly with the ***do()*** function, simply wrapping the prepare and execute stages in one.

```
$dbh->do("CREATE TABLE if not exists student (
        id INTEGER PRIMARY KEY, first_name TEXT,
         last_name TEXT, email TEXT) ");
```

Invoking do( ) repeatedly to insert a uge number of rows into a table, you could be preparing a statement handle many times more than is required, especially if the statement contained placeholder variables.

PERL SCRIPTING FOR SQLITE

DBI: Creating/accessing a Sqlite .db

```
#!/usr/bin/perl -w

use DBI;
use strict;

# CONFIG VARIABLES
my $platform = "SQLite";
my $database = "hollywood.db";
my $host = "localhost";
my $port = "3306";
my $user = "username";
my $pw = "password";

# DATA SOURCE NAME
my $dsn = "dbi:$platform:$database:$host:$port";

# PERL DBI CONNECT
my $dbh = DBI->connect($dsn, $user, $pw) or die "Cannot connect:
$DBI::errstr";
```

DBI: Creating and populating tables

```
# creating the "hollywood" database
$dbh->do("CREATE TABLE actors(aid integer primary key, name text)");

$dbh->do("CREATE TABLE movies(mid integer primary key, title text)");

$dbh->do("CREATE TABLE actors_movies(id integer primary key, mid integer,
aid integer)");


#populating "actors" table
$dbh->do("INSERT INTO actors(name) VALUES('Philip Seymour Hofman')");
$dbh->do("INSERT INTO actors(name) VALUES('Kate Shindle')");
$dbh->do("INSERT INTO actors(name) VALUES ('Kelci Stephenson')");
$dbh->do("INSERT INTO actors(name) VALUES('Al Pacino')");
$dbh->do("INSERT INTO actors(name) VALUES('Gabrielle Anwar')");
$dbh->do("INSERT INTO actors(name) VALUES('Patricia Arquette')");
$dbh->do("INSERT INTO actors(name) VALUES('Gabriel Byrne')");
$dbh->do("INSERT INTO actors(name) VALUES('Max von Sydow')");
$dbh->do("INSERT INTO actors(name) VALUES('Ellen Burstyn')");
```

DBI: Querying a Sqlite .db Ex.1

```
my $sql = „SELECT id, name, title, phone FROM employees „

my $sth = $dbh->prepare( $sql );

$sth->execute();

#bind colums to scalar references
my( $id, $name, $title, $phone );

$sth->bind_columns( \$id, \$name, \$title, \$phone );

while( $sth->fetch() ) {

  print "$name, $title, $phone\n";

}
```

DBI: Querying a Sqlite .db Ex.2

```
# EXECUTE THE QUERY
my $query = "SELECT actors.name , movies.title  FROM
actors,movies,actors_movies WHERE actors.aid=actors_movies.aid and
movies.mid=actors_movies.mid";

my $sth=$dbh->selectall_arrayref($query);

print "Actor                          Movie \n" ;
print "======================  ====================\n";

foreach my $row (@$sth) {
my ($name, $title) = @$row;

### Print out the table metadata...
printf "%-23s %-23s \n", $name, $title;

}
```

DBI: Querying a Sqlite .db Ex.3

```
my @names = ( "Larry%", "Tim%", "Randal%", "Doug%" );

my $sql = „SELECT id, name, title, phone FROM employees
            WHERE name LIKE ?“;

my $sth = $dbh->prepare( $sql );

for( @names ) {
# bind_param to prepare an SQL statement one time and execute it very quickly.
  $sth->bind_param( 1, $_, );
  $sth->execute();
  my( $id, $name, $title, $phone );
  $sth->bind_columns( undef, \$id, \$name, \$title, \$phone );

     while( $sth->fetch() ) {
      print "$name, $title, $phone\n";
      }
}
$sth->finish();
```

Perl DBI: Transactions

- Transactions: UPDATE, DELETE, etc.
- ***eval {…}*** blocks to trap errors
- ***$dbh->commit();***
- ***$dbh->rollback();***

Perl DBI: Commit

```
foreach my $country_code ( qw(US CA GB IE FR) ) {
print "Processing $country_code\n";
### Do all the work for one country inside an eval
eval {
### Read, parse and sanity check the data file (e.g., using DBD::CSV)
my $data = load_sales_data_file( "$country_file.csv" );
### Add data from the Web (e.g., using the LWP modules)
add_exchange_rates( $data, $country_code,"http://exchange-rate.com" );
### Perform database loading steps (e.g., using DBD::Oracle)
insert_sales_data( $dbh, $data );
update_country_summary_data( $dbh, $data );
insert_processed_files( $dbh, $country_code );
### Everything done okay for this file, so commit the database changes
$dbh->commit();
}; #end eval
```

Perl DBI: Rollback

```
### If something went wrong...
if ($@) {
### Tell the user that something went wrong, and what went wrong
warn "Unable to process $country_code: $@\n";
### Undo any database changes made before the error occured
$dbh->rollback();
### Keep track of failures
push @failed, $country_code;
}
} #end foreach
```

Http://perl.org

Http://sqlite.org

http://cpan.perl.org

http://dbi.perl.org/

Check my blog:

Http://lauraliparulo.altervista.org

Also available on:

as „Laura Liparulo“