

Notes from Well House Consultants

These notes are written by Well House Consultants and distributed under their Open Training Notes License. If a copy of this license is not supplied at the end of these notes, please visit

*<http://www.wellho.net/net/whcotnl.html>
for details.*

1.1 Well House Consultants

Well House Consultants provides niche training, primarily but not exclusively in Open Source programming languages. We offer public courses at our training centre and private courses at your offices. We also make some of our training notes available under our "Open Training Notes" license, such as we're doing in this document here.

1.2 Open Training Notes License

With an "Open Training Notes License", for which we make no charge, you're allowed to print, use and distribute these notes provided that you retain the complete and unaltered license agreement with them, including our copyright statement. This means that you can learn from the notes, and have others learn from them too.

You are NOT allowed to charge (directly or indirectly) for the copying or distribution of these notes, nor are you allowed to charge for presentations making any use of them.

1.3 Courses presented by the author

If you would like us to attend a course (Java, Perl, Python, PHP, Tcl/Tk, MySQL or Linux) presented by the author of these notes, please see our public course schedule at

<http://www.wellho.net/course/index.html>

If you have a group of 4 or more trainees who require the same course at the same time, it will cost you less to have us run a private course for you. Please visit our onsite training page at

<http://www.wellho.net/course/otc.html>

which will give you details and costing information

1.4 Contact Details

Well House Consultants may be found online at

<http://www.wellho.net>

graham@wellho.net

technical contact

lisa@wellho.net

administration contact

Our full postal address is

404 The Spa

Melksham

Wiltshire

UK SN12 6QL

Phone +44 (0) 1225 708225

Fax +44 (0) 1225 707126

Interfacing Applications to MySQL Databases

Structured Query Language is, as its name implies, a programming language. Application programs send commands in SQL to the relational database and also provide the application's user interface. In other words, they provide the glue between the application user and the database. Such programs can be written in a variety of languages including C, PHP, Java and Perl.

<i>Interfacing MySQL to Perl</i>	4
<i>Interfacing MySQL to PHP</i>	9
<i>Interfacing MySQL to Java</i>	9
<i>Interfacing MySQL to Tcl/Tk</i>	11
<i>Interfacing MySQL to C</i>	11
<i>When should I choose MySQL?</i>	13

At the core of MySQL is "mysqld", the database engine daemon which handles connections and requests via a TCP port – typically port number 3306. Provided with the MySQL distribution is the "mysql" program, a small client which allows for manual examination, amendment and administration of the database, but which is not a tool for daily production work.

Typically, your MySQL database will be accessed from applications written in other programming languages, such as PHP or Perl; this module goes on to cover (in varying depths) how a number of such languages interface to MySQL.

In order interface an SQL based database (such as MySQL) to a programming language (such as Perl), you need to have some prior understanding of both SQL and the programming language that you'll be using. This prior knowledge is assumed in this module.

2.1 Interfacing MySQL to Perl

The Perl DBI module (available from the CPAN) provides a database independent access tool for Perl to a wide range of SQL based databases, including commercial products such as Oracle, Sybase, Informix, and open source databases such as mSQL, PostGRESql and MySQL.

DBI is just an access tool; it wouldn't be practical to support and maintain a single package with support for such a wide range of databases. Additionally, you will need the appropriate DBD module from the CPAN.

DBI Data Base Independent

+DBD Data Base Dependant

In your Perl program, you'll

use DBI;

to draw in the DBI module, then when you run a connect command, Perl will also draw in the appropriate DBD module(s).

Here's a sample Perl program that selects a specific table from a database, and displays its contents:

```
#!/usr/bin/perl

# Perl program to display a table contents from a MySQL database

use DBI;

$db_handle = DBI -> connect('DBI:mysql:entertainment', 'root', $ARGV[0])
    or die ("connection: $DBI::errstr\n");

$getkey = $db_handle -> prepare("SELECT * FROM dvd");
$getkey -> execute;

while (@row = $getkey->fetchrow) {
    print (join (" | ", @row), "\n");
}
```

Let's see the table "dvd" from the "entertainment" database displayed through the mysql utility, then through the Perl program.

First, through mysql:

```
$ mysql -uroot -pabc123
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 12 to server version: 4.0.13

Type 'help' for help.
```

```
mysql> use entertainment;
```

Reading table information for completion of table and column names

You can turn off this feature to get a quicker startup with -A

Database changed

```
mysql> show tables;
```

```
+-----+
| Tables_in_entertainment |
+-----+
| dvd                      |
+-----+
```

1 row in set (0.01 sec)

```
mysql> select * from dvd;
```

```
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | title | length | cert | videotype | lang | bought | changed | timeseen | company |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 1 | Dinosaur | 79.5 | PG | PAL | English | 2001-03-15 | 20010701133113 | 1 | NULL |
| 2 | Chicken Run | 81 | U | PAL | English | 2001-02-07 | 20010701133113 | 1 | NULL |
| 3 | Groundhog Day | 97 | PG | PAL | English,French,German,Italian,Spanish | 2000-12-08 | 20010701133113 | 1 | NULL |
| 4 | Erin Brockovic | 126 | 15 | NULL | English,German | NULL | 20010701133113 | 1 | NULL |
| 5 | The Piano | 115 | 15 | NULL | NULL | NULL | 20010701133113 | 1 | NULL |
| 6 | The Color Purple | 148 | 15 | NULL | NULL | NULL | 20010701133113 | 1 | NULL |
| 7 | Billy Elliott | 106 | 15 | NULL | English | NULL | 20010701133113 | 1 | NULL |
| 8 | Patch Adams | 111 | 15 | NULL | English | NULL | 20010701133113 | 1 | NULL |
| 9 | Toy Story 2 | 89 | U | NULL | English | NULL | 20010701133113 | 1 | NULL |
| 10 | The Perfect Storm | 125 | 12 | NULL | English | NULL | 20010701135612 | 0 | Warner Bros. |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

10 rows in set (0.00 sec)

```
mysql> exit
```

Bye

\$

And through the Perl program:

```
$ ./sql1.pl abc123
```

```
1 | Dinosaur | 79.5 | PG | PAL | English | 2001-03-15 | 20010701133113 | 1 |
2 | Chicken Run | 81 | U | PAL | English | 2001-02-07 | 20010701133113 | 1 |
3 | Groundhog Day | 97 | PG | PAL | English,French,German,Italian,Spanish | 2000-12-08 | 20010701133113 | 1 |
4 | Erin Brockovich | 126 | 15 | | English,German | | 20010701133113 | 1 |
5 | The Piano | 115 | 15 | | | | 20010701133113 | 1 |
6 | The Color Purple | 148 | 15 | | | | 20010701133113 | 1 |
7 | Billy Elliott | 106 | 15 | | English | | 20010701133113 | 1 |
8 | Patch Adams | 111 | 15 | | English | | 20010701133113 | 1 |
9 | Toy Story 2 | 89 | U | | English | | 20010701133113 | 1 |
10 | The Perfect Storm | 125 | 12 | | English | | 20010701135612 | 0 |
Warner Bros.
$
```

You'll notice that we still require the user of the Perl program to type in the database password, which we collect from the command line within the SQL connection. Technically, it would be quite possible to build the password into the Perl program, but then

- anyone who got a copy of the script would learn the password
 - and • a password change would result in a script change
- which are major security and maintenance issues.

The Perl program in detail

```
$db_handle = DBI -> connect('DBI:mysql:entertainment', 'root',
$ARGV[0])
    or die ("connection: $DBI::errstr\n");
```

The connection is made though the **connect** method in the DBI module, and the **connect** method returns an object reference. By returning an object reference in this way, Perl allows you to connect to more than one database engine at the same time within the same Perl application, which makes Perl an ideal tool for transferring information between different database products.

Parameters to connect are:

- The database to which to connect, in the format
DBI:[databasetype]:[databasename]:[hostcomputer]:[portnumber]
- The latter two parameters default to localhost and 3306.
- The SQL account name (not the system account name)
- The SQL account password

Optionally, a reference to a hash of additional parameters may also be given if you want to modify the default behaviour of the DBD.

If **connect** fails, it will return a null string – a false value which we've trapped in our example using a lazy **or** operator. If we want to know why the failure occurred, we can look within the DBI module [package] at the **errstr** variable which contains the descriptive text.

Having connected to the "entertainment" database, we want to print out the entire contents of the "dvd" table. The SQL for this is

```
SELECT * FROM dvd
```

and so we embed this command into our Perl:

```
$getkey = $db_handle -> prepare("SELECT * FROM dvd");
```

The **prepare** method, though, doesn't actually do the work. It simply tells the DBD module that we're going to run a command, and it returns a further reference to us, this time a reference to the command. There are two reasons why it doesn't just run the SQL command:

1. It might be that we want to pass in other SQL as a part of the same command (for example, "WHERE ...")
2. The result set returned might be vast, and if that's the case we'll want to get it back record-by-record rather than setting up any huge lists or hashes in our Perl.

Once we've completed our SQL instruction, we tell the DBD module to actually execute it on the Database:

```
$getkey -> execute;
```

and then we fetch back the result set in a loop. The **fetchrow** method iterates through the result set, returning a list of the fields in each line in turn (and a null when the response has been exhausted) which, for the purpose of our example, we simple turn into a series of "|" delimited lines.

```
while (@row = $getkey->fetchrow) {
    print (join (" | ", @row), "\n");
}
```

There, job done!

A Further Example

This example includes SQL commands to create and drop a table, and to select specific records too. The program:

```
#!/usr/bin/perl
# Example of handling

use DBI;

"@ARGV" or die ("Usage: $0 [populate|delete|whereclause] password\n");

$db_handle = DBI -> connect('DBI:mysql:test:localhost:3306',
                            'root', $ARGV[-1] , {RaiseError => 1})
or die ("connection: $DBI::errstr\n");

pop @ARGV;

$field_defs = <<"FDS";
code          CHAR(20),
latitude      FLOAT,
longitude     FLOAT,
description   CHAR(40),
journeyflags  CHAR(6)
FDS

# Special cases

($ARGV[0] =~ /^populate$/i) and populate() and exit;
($ARGV[0] =~ /^delete$/i) and remove() and exit;

# Other cases - enquiry

$whereclause = (@ARGV > 0)?"WHERE @ARGV":"";

display($whereclause);

#####

sub populate {

    # create and fill the database table

    open (D,"distances") or die ("No distances text file
available\n");
    $db_handle -> do("CREATE TABLE dist ($field_defs)");

    while (<D>) {
        ($code,$lat,$long,@txt) = split;
        if ($txt[-1] =~ /[+*0%]/) {
            $flags = pop @txt;
        } else {
            $flags = "";
        }
        $recin = ''.join(' ',
        '$code,$lat,$long,@txt',$flags).'

```

```

        1;
    }

#####

sub remove {
    # Wow - this is dangerous .....
    $db_handle -> do ("DROP TABLE dist");
    print "Table DROPPED\n";
    1;
}

#####

sub display {
    $what = $_[0];
    print "Listing records $what\n";
    $getkey = $db_handle -> prepare("SELECT * FROM dist $what");
    $getkey -> execute;

    while (@row = $getkey->fetchrow) {
        print "@row\n";
    }
}

```

And some examples of the program in operation:

```

$ sql2.pl
Usage: ./sql2.pl [populate|delete|whereclause] password
$ sql2.pl abc123
Listing records
DBD::mysql::st execute failed: Table 'test.dist' doesn't exist at
./sql2.pl line 71.
$ sql2.pl populate abc123
Table populated from text file
$ sql2.pl code = \"LL\" abc123
Listing records WHERE code = "LL"
LL 3.8 53.3 North Wales 0
$ sql2.pl abc123
Listing records
AB 3 57.8 Aberdeen +
AL 0.4 51.7 St. Albans -
B 2 52.5 Birmingham -
BA 2.4 51.4 Bath
BB 2.6 53.8 Blackburn 0
(etc)
Wexford 6.6 52.4 County *0%
Carlow 6.9 52.6 County *0%
Waterford 7.5 52.2 County *0%
Kilkenny 7.2 52.5 County *0%
Cork 8.9 52.2 County *+%
Kerry 9.8 52.3 County *+%
(etc)
Norway 10.5 59.8 Oslo *+!
Sweden 18 59 Stockholm *+%
Austria 18 47 Vienna *+%
Italy 14 42 Rome *+%
Italia 14 42 Roma *+%

```

```

$ sql2.pl delete abc123
Table DROPPED
$ sql2.pl abc123
Listing records
DBD::mysql::st execute failed: Table 'test.dist' doesn't exist at
./sql2.pl line 71.
$

```

For further details of interfacing Perl to any relational database, see "Programming the Perl DBI" by Alligator Descartes and Tim Bunce, published by O'Reilly.

2.2 Interfacing MySQL to PHP

PHP is an HTML embedded scripting language which has rocketed in popularity recently. It includes functions to make direct calls to the MySQL database engine, and like MySQL, it's open source. This means that it's "a natural" to use MySQL with PHP if you're going to be using a browser (i.e. a web front end) to access your relational database. So much is MySQL used in association with PHP that a number of books cover both subjects (plus the integration of the two). At the time of writing, we already have four such volumes in our library.

Here's an example of a PHP script which reports on the contents of a MySQL database within a web page:

```

<head>
<title>Report contents of table</title>
</head>
<body bgcolor=white>
This page reports contents of an SQL table ....<BR>
<font color=red size=5>
<?php
$dbid = mysql_connect('localhost','root','abc123');
mysql_select_db("test",$dbid);

$query = "SELECT * FROM demo";
$result = mysql_query($query,$dbid);
print("<TABLE BORDER=1>");
while ($row = mysql_fetch_row($result)) {
    print("<tr><td>$row[0]</td><td>$row[1]</td></tr>");
}
print("</table>");
?>
</font><P>
OK. Done!
</body>

```

2.3 Interfacing MySQL to Java

Java interfaces to databases through JDBC (Java Database Connectivity). Your Java Runtime Environment should include the *java.sql* package by default, which is basically a driver manager class and does not include any drivers for any specific databases. You then source appropriate drivers from elsewhere; there's a list of suitable drivers on Sun's site:

<http://industry.java.sun.com/products/jdbc/drivers>

which currently lists 155 drivers, including several for MySQL.

In addition, you could interface MySQL to Java using the JDBC to ODBC bridge but ... don't; you'll be adding in an extra layer of conversions, and the drivers listed

on the site above are all "type 4" drivers which means that they're written in native Java code.

Here's a complete working example, using drivers sourced via this web site:

```
public class jdbc1 {

public static void main(String [] args) {

    java.sql.Connection conn = null;

    System.out.println("SQL Test");

    try {
        Class.forName("org.gjt.mm.mysql.Driver").newInstance();
        conn = java.sql.DriverManager.getConnection(

"jdbc:mysql://bhajee/test?user=jtest&password=");

    }
    catch (Exception e) {
        System.out.println(e);
        System.exit(0);
    }

    System.out.println("Connection established");

    try {
        java.sql.Statement s = conn.createStatement();
        java.sql.ResultSet r = s.executeQuery ("SELECT code,
latitude, longitude FROM dist");
        while(r.next()) {
            System.out.println (
                r.getString("code") + " " +
                r.getString("latitude") + " " +
                r.getString("longitude") );
        }
    }
    catch (Exception e) {
        System.out.println(e);
        System.exit(0);
    }
}
}
```

And in operation:

```
$ java jdbc1
SQL Test
Connection established
java.sql.SQLException: General error: Table 'test.dist' doesn't exist
$ sql2.pl populate abc123
Table populated from text file
$ java jdbc1
SQL Test
Connection established
AB 3 57.8
AL 0.4 51.7
```

```

B 2 52.5
BA 2.4 51.4
BB 2.6 53.8
BD 1.9 53.8
BH 2.9 50.8
BL 2.5 53.6
(etc)
Portugal -8.5 37
Denmark 14 55.5
Germany 9 50
Norway 10.5 59.8
Sweden 18 59
Austria 18 47
Italy 14 42
Italia 14 42
$

```

2.4 Interfacing MySQL to Tcl/Tk

A Tcl/Tk 8.0 interface to MySQL is available from
<http://www.xdobry.de/mysqltcl>

Download from that site ... you get full instructions, plus test programs. The following code snippet connects (as user root, no password, localhost) to a MySQL database that's already been created and populated, and reads back all records in the Student table in the uni database.

```

load ../libs/libmysqltcl.so

set handle [mysqlconnect -user root]

set rows [mysqlsel $handle {select * from uni.Student}]
for {set x 0} {$x<$rows} {incr x} {
    set res [mysqlnext $handle]
    set nr [lindex $res 0]
    set name [lindex $res 1]
    set sem [lindex $res 2]
    puts "$nr, $name, $sem"
}

mysqlclose

```

2.5 Interfacing MySQL to C

We've looked at interfacing to Perl, Java, PHP and Tcl in this section, and in each case the library involved has been provided by a third party or by the provider of the language. MySQL actually ships with a C language API, and that's what is used within each of the interfaces we've already looked at.

By the nature of the language, C code to interface to a MySQL database will be longer and slower to develop than code in one of the other languages that we have covered, but if you want to you can use it to produce a very fast-running client and you'll have every possible facility offered to you through MySQL there at your fingertips. This is a bit of a specialist subject; in our experience, few people are writing directly in C these days so we'll just include one short example in this module.

Let's say we want a table of users, flagging up those to whom our administrator has given **create** privilege. We want the C application that generates this list to log in automatically to the SQL database on a computer with a fixed name so that the application can be run very easily:

```
$ ./sqlusers
```

```

      User          Host
      ----          -
root    localhost   HAS CREATE PRIVILEGE
              localhost
content localhost
graham   %
reader  localhost
trainee  %          HAS CREATE PRIVILEGE

```

```
$
```

Our application is "just a piece of C" which includes headers from the header file `mysql.h`, and calls to library functions in the `mysql` library:

```

#include <stdio.h>
#include <mysql.h>

/* Simple example by Well House Consultants (enquiries@wellho.net) showing
MySQL C language API. Shows the mechanism only; error checking, etc, must
be added to a live application */

#define HOST          "dhansak"
#define USER          "trainee"
#define PASS          "abc123"
#define DBNAME        "mysql"

MYSQL *connector;
MYSQL_RES *result;
MYSQL_ROW row;

main (int argc, char **argv) {
    connector = mysql_init(NULL);
    mysql_real_connect( connector,
                        HOST, USER, PASS, DBNAME,
                        0, NULL, 0);

    mysql_query(connector, "SELECT User, Host, Create_priv from user");
    result = mysql_store_result(connector);
    printf("%12s %12s\n", "User", "Host");
    printf("%12s %12s\n", "----", "----");

    while (( row = mysql_fetch_row (result)) != NULL) {
        if (strcmp(row[2], "Y") == 0) {
            printf("%12s %12s   HAS CREATE PRIVILEGE \n",
                  row[0], row[1]);
        } else {
            printf("%12s %12s\n",
                  row[0], row[1]);
        }
    }
}

```

```

    }

}
mysql_free_result(result);
mysql_close(connector);
exit (0);
}

```

To save a long compile line command, we've built the compile command for our example into a Makefile:

```

SQL = -I/usr/include/mysql -L/usr/lib/mysql

sqlusers:    sqlusers.c
    gcc -o sqlusers sqlusers.c $(SQL) -lmysqlclient -lm

```

and really that's all there is to it. Many other functions are available in the mysql to C API, and for a realistic program you would add a great deal of error checking (almost all the functions above return a NULL if they fail), structure, command-line parameters, etc.

2.6 When should I choose MySQL?

There are numerous relational databases available, and decision makers have quite a task on their hand deciding which they should choose. In this section we'll look at some of the strengths and weaknesses of MySQL to help you understand better any decision you have to take.

MySQL is characterised as a free, fast, reliable open source relational database. It does lack some sophistication and facilities, but it has an active development team, and as it goes from release to release, more capabilities are added. At certain times there will be a trade-off between speed and capabilities, and the MySQL team intend to keep their database engine fast and reliable.

Brief comparisons

PostgreSQL is another open source relational database management system. It conforms to the SQL standards much better than MySQL and runs on a wide variety of platforms, but the extra feature set slows and complicates its use.

Oracle is the leading commercial RDMS; it's highly flexible, runs on many platforms and has a full and sophisticated feature set. Because of its highly tunable nature, an Oracle database administrator needs to be well and heavily trained.

Microsoft **SQL Server** runs only on Windows platforms, which excludes its use in applications where there is a need to run on Linux or Unix, or potentially such a need in the future. Since the majority of big web servers are running two "nix" operating systems, there's a tendency for SQL servers to turn up on smaller or non-web applications where it is easy to use and has a relatively low cost and a good performance. Concerns were expressed on a previous MySQL course about the robustness of SQL Server. It was said to have a tendency to lock up and crash, and sometimes only return partial queries. A discussion amongst the group (some of whom were SQL Server gurus) concluded that these problems are indeed real. The group also commented that Microsoft recommend that SQL Server run on a dedicated computer, as the problems are caused by the interference between applications. In other words, it should be reliable if run as directed.

Limitations of MySQL, and work-arounds

There are variations and inconsistencies in the more sophisticated commands and structures of SQL from database engine to database engine. MySQL uses a clean,

tight set of SQL commands. The following functionality is present in many database systems, but not in MySQL

subselects

A subselect is where you use a `select` command within the `WHERE` clause of another command in order to choose your rows for access based on data contained elsewhere in the database. You can work around the lack of subselects from your application; simply run the internal `select` first, and pass the results in to the main command.

Subselects are on the MySQL to-do list and this limitation will be removed at some point.

transactions and commit/rollback

A transaction is a series of database updates which form a group that will result in a loss of data integrity if it is not completed. For example, if you are adding a new record to one table with relational elements in another and only one table actually gets updated, you have a problem.

More sophisticated (potentially costly, potentially slower running) RDMS systems have a transaction capability, where a transaction is defined as such, with `COMMIT` commands to indicate the end of a transaction and a rollback capability to cleanly abort and unravel a transaction that cannot complete.

In MySQL, you do not have a built-in transaction capability, but you can lock and unlock tables to ensure that updates are grouped together, and use error checking to spot problems and rollback if necessary yourself. This turns out to be perfectly adequate for many applications.

Foreign keys and referential Integrity

A foreign key lets you declare a key in one table that relates to another table, and referential integrity is the maintenance of multiple tables via that one key. Were this sort of facility available, it would allow for the deletion of a row from one table to do an automatic delete from another table in order to maintain the integrity.

For example, if you had a table of bus services and another of bus places and times, then you would want the deletion of a bus service to delete all the places and times for that service from the other table. In MySQL you could of course do this, it's just that you have to program the multiple transactions yourself.

Foreign key support would have considerable speed and complexity effects on MySQL and there are no plans to implement it. Read this section in conjunction with the "stored procedures" and "transaction processing" notes which are also in this section.

Stored procedures and triggers

A stored procedure is a set of instructions that can be stored within a database or database engine, and can be invoked by specific request ("go run this procedure") or by a trigger (automatically when some other event, such as an application making a particular change or enquiry) happens. A stored procedure is a program (in a programming language) in its own right, for example, Oracle's PL/SQL.

The 4.0 release of MySQL is an enabling release to make underlying changes that will enable future enhancements. A stored procedure capability is planned for the 4.1 development release, which will in due course transform into the 4.2 production release.

No doubt you'll be accessing your database through applications written in Perl or PHP or Java or Tcl or some other language. Facilities exist in all these languages for you to share procedures (or subroutines, methods or procs) between a number of

applications via files of common code, so you can easily implement common procedures to perform certain database activity within your code today!

Views

A view is sometimes known as a "virtual table". It behaves like a table, although in reality the data that it presents is a combination of several tables. This is planned for a future MySQL release.

Record level privileges and locking

MySQL does not support locking at individual row (record) level. However, you can implement co-operative locking in your software.

Why are the limitations not always limitations?

The features we have just described, which are not present within MySQL, can be reproduced or emulated in your application(s). Whilst having them present might save a certain amount of development and maintenance work, none of them is vital.

It is of course up to you to decide whether you want to do this extra work, or whether you would be better to buy and learn Oracle. Remember that with good application code design and the use of OO or structured shared code principles, the extra work can be minimised and done just once, then shared between all your applications.

Portability of applications is increasingly important, and if you write code that relies on facilities that are offered only by certain products, you are tying yourself in much more tightly than you might wish. Paradoxically, the restriction of MySQL to a fundamental set of capabilities and commands can help you here, as code you develop can be made to run on other engines with relative ease. Just imagine the problems you would have if you relied heavily on stored procedures, then had to port to a system that had a different stored procedure language or none at all.

Perl and Java have taken the issue of database portability to heart, and they provide a common wrapper for any database engine so that the application programmer can write easily transferrable code ... all of which means the application programmer won't be encouraged to write code for a specific database engine anyhow, so why pay the cost and performance of having those features there? PHP takes a somewhat different approach by providing a separate function set for interfacing to each database; if you're writing PHP to use a database, we recommend that you use a wrapper to put in the same commonality as used in Perl and Java, as described in Chapter 16 of [PHP Developers Cookbook](#).



Exercise

License

*These notes are distributed under the **Well House Consultants Open Training Notes License**. Basically, if you distribute it and use it for free, we'll let you have it for free. If you charge for its distribution of use, we'll charge.*

3.1 Open Training Notes License

Training notes distributed under the **Well House Consultants Open Training Notes License** (WHCOTNL) may be reproduced for any purpose PROVIDE THAT:

- This License statement is retained, unaltered (save for additions to the change log) and complete.
- No charge is made for the distribution, nor for the use or application thereof. This means that you can use them to run training sessions or as support material for those sessions, but you cannot then make a charge for those training sessions.
- Alterations to the content of the document are clearly marked as being such, and a log of amendments is added below this notice.
- These notes are provided "as is" with no warranty of fitness for purpose. Whilst every attempt has been made to ensure their accuracy, no liability can be accepted for any errors of the consequences thereof.

Copyright is retained by Well House Consultants Ltd, of 404, The Spa, Melksham, Wiltshire, UK, SN12 6QL - phone number +44 (1) 1225 708225. Email contact - Graham Ellis (graham@wellho.net).

Please send any amendments and corrections to these notes to the Copyright holder - under the spirit of the Open Distribution license, we will incorporate suitable changes into future releases for the use of the community.

If you are charged for this material, or for presentation of a course (Other than by Well House Consultants) using this material, please let us know. It is a violation of the license under which this notes are distributed for such a charge to be made, except by the Copyright Holder.

If you would like Well House Consultants to use this material to present a training course for your organisation, or if you wish to attend a public course is one is available, please contact us or see our web site - <http://www.wellho.net> - for further details.

Change log
Original Version, Well House Consultants, 2004

Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____

License Ends.