NAME

DBI::FAQ -- The Frequently Asked Questions for the Perl5 Database Interface

SYNOPSIS

```
perldoc DBI::FAQ
```

VERSION

This document is currently at version *0.38*, as of *February 8th, 2000*.

That's **very** old. A newer FAQ can be found at <http://faq.dbi-support.com/>

Neither this document nor that web site are actively maintained. Volunteers are welcome.

DESCRIPTION

This document serves to answer the most frequently asked questions on both the DBI Mailing Lists and personally to members of the DBI development team.

Basic Information & Information Sources

1.1 What is DBI, DBperl, Oraperl and *perl?

To quote Tim Bunce, the architect and author of DBI:

```
DBI is a database access Application Programming Interface (API)
for the Perl Language. The DBI API Specification defines a set
of functions, variables and conventions that provide a consistent
database interface independent of the actual database being used.
```

In simple language, the DBI interface allows users to access multiple database types transparently. So, if you connecting to an Oracle, Informix, mSQL, Sybase or whatever database, you don't need to know the underlying mechanics of the 3GL layer. The API defined by DBI will work on *all* these database types.

A similar benefit is gained by the ability to connect to two *different* databases of different vendor within the one perl script, *ie*, I want to read data from an Oracle database and insert it back into an Informix database all within one program. The DBI layer allows you to do this simply and powerfully.

DBperl is the old name for the interface specification. It's usually now used to denote perl*4* modules on database interfacing, such as, *oraperl*, *isqlperl*, *ingperl* and so on. These interfaces didn't have a standard API and are generally *not* supported.

Here's a list of DBperl modules, their corresponding DBI counterparts and support information. *Please note*, the author's listed here generally *do not* maintain the DBI module for the same database. These email addresses are unverified and should only be used for queries concerning the perl4 modules listed below. DBI driver queries should be directed to the *dbi-users* mailing list.

```
Module Name Database Required Author DBI
----------- ----------------- ------ ---
Sybperl Sybase Michael Peppler DBD::Sybase
<mpeppler@itf.ch>
Oraperl Oracle 6 & 7 Kevin Stock DBD::Oracle
<dbi-users@perl.org>
Ingperl Ingres Tim Bunce & DBD::Ingres
Ted Lemon
<dbi-users@perl.org>
Interperl Interbase Buzz Moschetti DBD::Interbase
<buzz@bear.com>
Uniperl Unify 5.0 Rick Wargo None
<rickers@coe.drexel.edu>
Pgperl Postgres Igor Metz DBD::Pg
<metz@iam.unibe.ch>
Btreeperl NDBM John Conover SDBM?
<john@johncon.com>
```

```
Ctreeperl C-Tree John Conover None
<john@johncon.com>
Cisamperl Informix C-ISAM Mathias Koerber None
<mathias@unicorn.swi.com.sg>
Duaperl X.500 Directory Eric Douglas None
User Agent
```

However, some DBI modules have DBperl emulation layers, so, *DBD::Oracle* comes with an Oraperl emulation layer, which allows you to run legacy oraperl scripts without modification. The emulation layer translates the oraperl API calls into DBI calls and executes them through the DBI switch.

Here's a table of emulation layer information:

```
Module Emulation Layer Status
------ --------------- ------
DBD::Oracle Oraperl Complete
DBD::Informix Isqlperl Under development
DBD::Ingres Ingperl Complete?
DBD::Sybase Sybperl Working? ( Needs verification )
DBD::mSQL Msqlperl Experimentally released with
DBD::mSQL-0.61
```

The *Msqlperl* emulation is a special case. *Msqlperl* is a perl5(1) driver for *mSQL* databases, but does not conform to the DBI Specification. It's use is being deprecated in favour of *DBD::mSQL* *Msqlperl* may be downloaded from CPAN *via*:

http://www.perl.com/cgi-bin/cpan_mod?module=Msqlperl

1.2. Where can I get it from?

The Comprehensive Perl Archive Network resources should be used for retrieving up-to-date versions of the DBI and drivers. CPAN may be accessed *via* Tom Christiansen's splendid *CPAN multiplexer* program located at:

http://www.perl.com/CPAN/

For more specific version information and exact URLs of drivers, please see the DBI drivers list and the DBI module pages which can be found on:

http://dbi.perl.org/

This list is automatically generated on a nightly basis from CPAN and should be up-to-date.

1.3. Where can I get more information?

There are a few information sources on DBI.

"Programming the Perl DBI"

"Programming the Perl DBI" is the *official* book on the DBI written by Alligator Descartes and Tim Bunce and published by O'Reilly & Associates. The book was released on February 9th, 2000.

The table of contents is:

```
C. ASLaN Sacred Site Charter
Index
```

The book should be available from all good bookshops and can be ordered online either <I>via</I> O'Reilly & Associates

http://www.oreilly.com/catalog/perldbi

or Amazon

http://www.amazon.com/exec/obidos/ASIN/1565926994/dbi

POD documentation

*POD*s are chunks of documentation usually embedded within perl programs that document the code "*in place*", providing a useful resource for programmers and users of modules. POD for DBI and drivers is beginning to become more commonplace, and documentation for these modules can be read with the `perldoc(1)` program included with Perl.

The DBI Specification

The POD for the DBI Specification can be read with the:

```
perldoc DBI
```

command. The Specification also forms Appendix A of "Programming the Perl DBI".

Oraperl

Users of the Oraperl emulation layer bundled with *DBD::Oracle* may read up on how to program with the Oraperl interface by typing:

```
perldoc Oraperl
```

This will produce an updated copy of the original oraperl man page written by Kevin Stock for perl4. The oraperl API is fully listed and described there.

Drivers

Users of the DBD modules may read about some of the private functions and quirks of that driver by typing:

```
perldoc <driver>
```

For example, the *DBD::mSQL* driver is bundled with driver-specific documentation that can be accessed by typing

```
perldoc DBD::mSQL
```

Frequently Asked Questions

This document, the *Frequently Asked Questions*is also a vailable as POD documentation! You can read this on your own system by typing:

```
perldoc DBI::FAQ
```

This may be more convenient to persons not permanently, or conveniently, connected to the Internet. The *DBI::FAQ* module should be downloaded and installed for the more up-to-date version.

The version of *DBI::FAQ* shipped with the `DBI` module may be slightly out of date.

POD in general

Information on writing POD, and on the philosophy of POD in general, can be read by typing:

```
perldoc perlpod
```

Users with the Tk module installed may be interested to learn there is a Tk-based POD reader available called `tkpod`, which formats POD in a convenient and readable way. This is available *via* CPAN as the module called *Tk::POD* and is highly recommended.

Driver and Database Characteristics

The driver summaries that were produced for Appendix B of "Programming the Perl DBI" are available online at:

http://dbi.perl.org/

in the driver information table. These summaries contain standardised information on each driver and database which should aid you in selecting a database to use. It will also inform you quickly of any issues within drivers or whether a driver is not fully compliant with the DBI Specification.

Rambles, Tidbits and Observations

http://dbi.perl.org/tidbits

There are a series of occasional rambles from various people on the DBI mailing lists who, in an attempt to clear up a simple point, end up drafting fairly comprehensive documents. These are quite often varying in quality, but do provide some insights into the workings of the interfaces.

Articles

A list of articles discussing the DBI can be found on the DBI WWW page at:

http://dbi.perl.org/

These articles are of varying quality and age, from the original Perl Journal article written by Alligator and Tim, to more recent debacles published online from about.com.

README files

The *README* files included with each driver occasionally contains some useful information (no, really!) that may be pertinent to the user. Please read them. It makes our worthless existences more bearable. These can all be read from the main DBI WWW page at:

http://dbi.perl.org/

Mailing Lists

There are three mailing lists for DBI:

```
dbi-announce@perl.org -- for announcements, very low traffic
dbi-users@perl.org -- general user support
dbi-dev@perl.org -- for driver developers (no user support)
```

For information on how to subscribe, set digest mode etc, and unsubscribe, send an email message (the content will be ignored) to:

```
dbi-announce-help@perl.org
dbi-users-help@perl.org
dbi-dev-help@perl.org
```

Mailing List Archives

US Mailing List Archives

http://outside.organic.com/mail-archives/dbi-users/

Searchable hypermail archives of the three mailing lists, and some of the much older traffic have been set up for users to browse.

European Mailing List Archives

http://www.rosat.mpe-garching.mpg.de/mailing-lists/PerlDB-Interest

As per the US archive above.

Compilation Problems

2.1. Compilation problems or "It fails the test!"

First off, consult the README for that driver in case there is useful information about the problem. It may be a known problem for your given architecture and operating system or

database. You can check the README files for each driver in advance online at:

http://dbi.perl.org/

If it's a known problem, you'll probably have to wait till it gets fixed. If you're *really* needing it fixed, try the following:

Attempt to fix it yourself
This technique is generally *not* recommended to the faint-hearted. If you do think you have managed to fix it, then, send a patch file (context diff) to the author with an explanation of:

- What the problem was, and test cases, if possible.
- What you needed to do to fix it. Please make sure you mention everything.
- Platform information, database version, perl version, module version and DBI version.

*Email the author*Do *NOT* whinge!
Please email the address listed in the WWW pages for whichever driver you are having problems with. Do *not* directly email the author at a known address unless it corresponds with the one listed.

We tend to have real jobs to do, and we do read the mailing lists for problems. Besides, we may not have access to <*insert your favourite brain-damaged platform here*> and couldn't be of any assistance anyway! Apologies for sounding harsh, but that's the way of it!

However, you might catch one of these creative genii at 3am when we're doing this sort of stuff anyway, and get a patch within 5 minutes. The atmosphere in the DBI circle is that we *do* appreciate the users' problems, since we work in similar environments.

If you are planning to email the author, please furnish as much information as possible, *ie*:

- *ALL* the information asked for in the README file in the problematic module. And we mean *ALL* of it. We don't put lines like that in documentation for the good of our health, or to meet obscure README file standards of length.
- If you have a core dump, try the *Devel::CoreStack* module for generating a stack trace from the core dump. Send us that too. *Devel::CoreStack* can be found on CPAN at:

  http://www.perl.com/cgi-bin/cpan_mod?module=Devel::CoreStack
- Module versions, perl version, test cases, operating system versions and *any other pertinent information*.

Remember, the more information you send us, the quicker we can track problems down. If you send us no useful information, expect nothing back.

Finally, please be aware that some authors, including Tim Bunce, specifically request that you do *not* mail them directly. Please respect their wishes and use the email addresses listed in the appropriate module `README` file.

Email the dbi-users Mailing List
It's usually a fairly intelligent idea to *cc* the mailing list anyway with problems. The authors all read the lists, so you lose nothing by mailing there.

Platform and Driver Issues

3.1 What's the difference between ODBC and DBI?

In terms of architecture - not much: Both define programming interfaces. Both allow multiple drivers to be loaded to do the actual work.

In terms of ease of use - much: The DBI is a 'high level' interface that, like Perl itself, strives to make the simple things easy while still making the hard things possible. The ODBC is a 'low level' interface. All nuts-bolts-knobs-and-dials.

Now there's an ODBC driver for the DBI (DBD::ODBC) the ''What's the difference'' question is

more usefully rephrased as:

Chapter 7 of "Programming the Perl DBI" covers this topic in far more detail and should be consulted.

3.2 What's the difference between Win32::ODBC and DBD::ODBC?

The DBI, and thus DBD::ODBC, has a different philosophy from the Win32::ODBC module:

The Win32::ODBC module is a 'thin' layer over the low-level ODBC API. The DBI defines a simpler 'higher level' interface.

The Win32::ODBC module gives you access to more of the ODBC API. The DBI and DBD::ODBC give you access to only the essentials. (But, unlike Win32::ODBC, the DBI and DBD::ODBC do support parameter binding and multiple prepared statements which reduces the load on the database server and can dramatically increase performance.)

The Win32::ODBC module only works on Win32 systems. The DBI and DBD::ODBC are very portable and work on Win32 and Unix.

The DBI and DBD::ODBC modules are supplied as a standard part of the Perl 5.004 binary distribution for Win32 (they don't work with the older, non-standard, ActiveState port).

Scripts written with the DBI and DBD::ODBC are faster than Win32::ODBC on Win32 and are trivially portable to other supported database types.

The DBI offers optional automatic printing or *die()*ing on errors which makes applications simpler and more robust.

The current DBD::ODBC driver version 0.16 is new and not yet fully stable. A new release is due soon [relative to the date of the next TPJ issue :-] and will be much improved and offer more ODBC functionality.

To summarise: The Win32::ODBC module is your best choice if you need access to more of the ODBC API than the DBI gives you. Otherwise, the DBI and DBD::ODBC combination may be your best bet.

Chapter 7 of "Programming the Perl DBI" covers this topic in far more detail and should be consulted.

3.3 Is DBI supported under Windows 95 / NT platforms?

Finally, yes! Jeff Urlwin has been working diligently on building *DBI* and *DBD::ODBC* under these platforms, and, with the advent of a stabler perl and a port of *MakeMaker*, the project has come on by great leaps and bounds.

The *DBI* and *DBD::Oracle* Win32 ports are now a standard part of DBI, so, downloading *DBI* of version higher than *0.81* should work fine as should using the most recent *DBD::Oracle* version.

3.4 Can I access Microsoft Access or SQL-Server databases with DBI?

Yes, use the *DBD::ODBC* driver.

3.5 Is there a DBD for <*insert favourite database here*>?

First check if a driver is available on CPAN by searching for the name of the database (including common abbreviations and aliases).

Here's a general query that'll match all distributions:

```
http://search.cpan.org/search?query=DBD&mode=dist
```

If you can't find a driver that way, you could check if the database supports ODBC drivers. If so then you could probably use the DBD::ODBC driver:

```
http://search.cpan.org/dist/DBD-ODBC/
```

If not, then try asking on the dbi-users mailing list.

3.6 What's DBM? And why should I use DBI instead?

Extracted from "*DBI - The Database Interface for Perl 5*":

```
``UNIX was originally blessed with simple file-based ``databases'', namely
the dbm system. dbm lets you store data in files, and retrieve
that data quickly. However, it also has serious drawbacks.

File Locking

The dbm systems did not allow particularly robust file locking
capabilities, nor any capability for correcting problems arising through
simultaneous writes [ to the database ].

Arbitrary Data Structures

The dbm systems only allows a single fixed data structure:
key-value pairs. That value could be a complex object, such as a
[ C ] struct, but the key had to be unique. This was a large
limitation on the usefulness of dbm systems.

However, dbm systems still provide a useful function for users with
simple datasets and limited resources, since they are fast, robust and
extremely well-tested. Perl modules to access dbm systems have now
been integrated into the core Perl distribution via the
AnyDBM_File module.''
```

To sum up, DBM is a perfectly satisfactory solution for essentially read-only databases, or small and simple datasets. However, for more scalable dataset handling, not to mention robust transactional locking, users are recommended to use a more powerful database engine *via DBI*.

Chapter 2 of "Programming the Perl DBI" discusses DBM files in detail.

3.7 What database do you recommend me using?

This is a particularly thorny area in which an objective answer is difficult to come by, since each dataset, proposed usage and system configuration differs from person to person.

From the current author's point of view, if the dataset is relatively small, being tables of less than 1 million rows, and less than 1000 tables in a given database, then *mSQL* is a perfectly acceptable solution to your problem. This database is extremely cheap, is wonderfully robust and has excellent support. More information is available on the Hughes Technology WWW site at:

http://www.hughes.com.au

You may also wish to look at MySQL which is a more powerful database engine that has a similar feel to mSQL.

http://www.tcx.se

If the dataset is larger than 1 million row tables or 1000 tables, or if you have either more money, or larger machines, I would recommend *Oracle RDBMS*. Oracle's WWW site is an excellent source of more information.

http://www.oracle.com

Informix is another high-end RDBMS that is worth considering. There are several differences between Oracle and Informix which are too complex for this document to detail. Information on Informix can be found on their WWW site at:

http://www.informix.com

In the case of WWW fronted applications, *mSQL* may be a better option due to slow connection times between a CGI script and the Oracle RDBMS and also the amount of resource each Oracle

connection will consume. *mSQL* is lighter resource-wise and faster.

These views are not necessarily representative of anyone else's opinions, and do not reflect any corporate sponsorship or views. They are provided *as-is*.

3.8 Is <*insert feature here*> supported in DBI?

Given that we're making the assumption that the feature you have requested is a non-standard database-specific feature, then the answer will be *no*.

DBI reflects a *generic* API that will work for most databases, and has no database-specific functionality.

However, driver authors may, if they so desire, include hooks to database-specific functionality through the `func()` method defined in the DBI API. Script developers should note that use of functionality provided *via* the `func()` methods is very unlikely to be portable across databases.

Programming Questions

4.1 Is DBI any use for CGI programming?

In a word, yes! DBI is hugely useful for CGI programming! In fact, I would tentatively say that CGI programming is one of two top uses for DBI.

DBI confers the ability to CGI programmers to power WWW-fronted databases to their users, which provides users with vast quantities of ordered data to play with. DBI also provides the possibility that, if a site is receiving far too much traffic than their database server can cope with, they can upgrade the database server behind the scenes with no alterations to the CGI scripts.

4.2 How do I get faster connection times with DBD::Oracle and CGI?

`Contributed by John D. Groenveld`

The Apache `httpd` maintains a pool of `httpd` children to service client requests.

Using the Apache *mod_perl* module by *Doug MacEachern*, the perl interpreter is embedded with the `httpd` children. The CGI, DBI, and your other favorite modules can be loaded at the startup of each child. These modules will not be reloaded unless changed on disk.

For more information on Apache, see the Apache Project's WWW site:

`http://www.apache.org`

The *mod_perl* module can be downloaded from CPAN *via*:

`http://www.perl.com/cgi-bin/cpan_mod?module=Apache`

4.3 How do I get persistent connections with DBI and CGI?

`Contributed by John D. Groenveld`

Using Edmund Mergl's *Apache::DBI* module, database logins are stored in a hash with each of these `httpd` child. If your application is based on a single database user, this connection can be started with each child. Currently, database connections cannot be shared between `httpd` children.

Apache::DBI can be downloaded from CPAN *via*:

`http://www.perl.com/cgi-bin/cpan_mod?module=Apache::DBI`

4.4 "When I run a perl script from the command line, it works, but, when I run it under the `httpd`, it fails!" Why?

Basically, a good chance this is occurring is due to the fact that the user that you ran it from the command line as has a correctly configured set of environment variables, in the case of *DBD::Oracle* variables like `ORACLE_HOME`, `ORACLE_SID` or `TWO_TASK`.

The `httpd` process usually runs under the user id of `nobody`, which implies there is no configured environment. Any scripts attempting to execute in this situation will correctly fail.

One way to solve this problem is to set the environment for your database in a `BEGIN { }` block at the top of your script. Another technique is to configure your WWW server to pass-through

certain environment variables to your CGI scripts.

Similarly, you should check your `httpd` error logfile for any clues, as well as the ''Idiot's Guide To Solving Perl / CGI Problems'' and ''Perl CGI Programming FAQ'' for further information. It is unlikely the problem is DBI-related.

The ''Idiot's Guide To Solving Perl / CGI Problems'' can be located at:

http://www.perl.com/perl/faq/index.html

as can the ''Perl CGI Programming FAQ''. Read *BOTH* these documents carefully!

4.5 How do I get the number of rows returned from a SELECT statement?

Count them. Read the DBI docs for the `rows()` method.

Miscellaneous Questions

5.1 Can I do multi-threading with DBI?

Perl version 5.005 and later can be built to support multi-threading. The DBI, as of version 1.02, does not yet support multi-threading so it would be unsafe to let more than one thread enter the DBI at the same time.

It is expected that some future version of the DBI will at least be thread-safe (but not thread-hot) by automatically blocking threads entering the DBI while it's already in use.

5.2 How do I handle BLOB data with DBI?

Handling BLOB data with the DBI is very straight-forward. BLOB columns are specified in a SELECT statement as per normal columns. However, you also need to specify a maximum BLOB size that the <I>database handle</I> can fetch using the `LongReadLen` attribute.

For example:

```
### $dbh is a connected database handle
$sth = $dbh->prepare( "SELECT blob_column FROM blobby_table" );
$sth->execute;
```

would fail.

```
### $dbh is a connected database handle
### Set the maximum BLOB size...
$dbh->{LongReadLen} = 16384; ### 16Kb...Not much of a BLOB!

$sth = $dbh->prepare( "..." );
```

would succeed <I>provided no column values were larger than the specified value</I>.

If the BLOB data is longer than the value of `LongReadLen`, then an error will occur. However, the DBI provides an additional piece of functionality that will automatically truncate the fetched BLOB to the size of `LongReadLen` if it is longer. This does not cause an error to occur, but may make your fetched BLOB data useless.

This behaviour is regulated by the `LongTruncOk` attribute which is set to a false value by default (thus making overlong BLOB fetches fail).

```
### Set BLOB handling such that it's 16Kb and can be truncated
$dbh->{LongReadLen} = 16384;
$dbh->{LongTruncOk} = 1;
```

Truncation of BLOB data may not be a big deal in cases where the BLOB contains run-length encoded data, but data containing checksums at the end, for example, a ZIP file, would be rendered useless.

5.3 How can I invoke stored procedures with DBI?

The DBI does not define a database-independent way of calling stored procedures.

However, most database that support them also provide a way to call them from SQL statements - and the DBI certainly supports that.

So, assuming that you have created a stored procedure within the target database, *eg*, an Oracle database, you can use `$dbh->do()` to immediately execute the procedure. For example,

```
$dbh->do( "BEGIN someProcedure; END;" ); # Oracle-specific
```

You should also be able to `prepare` and `execute`, which is the recommended way if you'll be calling the procedure often.

5.4 How can I get return values from stored procedures with DBI?

Contributed by Jeff Urlwin

```
$sth = $dbh->prepare( "BEGIN foo(:1, :2, :3); END;" );
$sth->bind_param(1, $a);
$sth->bind_param_inout(2, \$path, 2000);
$sth->bind_param_inout(3, \$success, 2000);
$sth->execute;
```

Remember to perform error checking, though! (Or use the `RaiseError` attribute).

5.5 How can I create or drop a database with DBI?

Database creation and deletion are concepts that are entirely too abstract to be adequately supported by DBI. For example, Oracle does not support the concept of dropping a database at all! Also, in Oracle, the database *server* essentially *is* the database, whereas in mSQL, the server process runs happily without any databases created in it. The problem is too disparate to attack in a worthwhile way.

Some drivers, therefore, support database creation and deletion through the private `func()` methods. You should check the documentation for the drivers you are using to see if they support this mechanism.

5.6 How can I `commit` or `rollback` a statement with DBI?

See the `commit()` and `rollback()` methods in the DBI Specification.

Chapter 6 of "Programming the Perl DBI" discusses transaction handling within the context of DBI in more detail.

5.7 How are `NULL` values handled by DBI?

`NULL` values in DBI are specified to be treated as the value `undef`. `NULL`s can be inserted into databases as `NULL`, for example:

```
$rv = $dbh->do( "INSERT INTO table VALUES( NULL )" );
```

but when queried back, the `NULL`s should be tested against `undef`. This is standard across all drivers.

5.8 What are these `func()` methods all about?

The `func()` method is defined within DBI as being an entry point for database-specific functionality, *eg*, the ability to create or drop databases. Invoking these driver-specific methods is simple, for example, to invoke a `createDatabase` method that has one argument, we would write:

```
$rv =$dbh->func( 'argument', 'createDatabase' );
```

Software developers should note that the `func()` methods are non-portable between databases.

5.9 Is DBI Year 2000 Compliant?

DBI has no knowledge of understanding of what dates are. Therefore, DBI itself does not have a Year 2000 problem. Individual drivers may use date handling code internally and therefore be potentially susceptible to the Year 2000 problem, but this is unlikely.

You may also wish to read the "Does Perl have a Year 2000 problem?" section of the Perl FAQ at:

http://www.perl.com/CPAN/doc/FAQs/FAQ/PerlFAQ.html

Support and Training

The Perl5 Database Interface is *FREE* software. IT COMES WITHOUT WARRANTY OF ANY KIND. See the DBI README for more details.

However, some organizations are providing either technical support or training programs on DBI. The present author has no knowledge as to the quality of these services. The links are included for reference purposes only and should not be regarded as recommendations in any way. *Caveat emptor*.

Commercial Support

The Perl Clinic

The Perl Clinic provides commercial support for *Perl* and Perl related problems, including the *DBI* and its drivers. Support is provided by the company with whom Tim Bunce, author of *DBI* and *DBD::Oracle* works and ActiveState. For more information on their services, please see:

http://www.perlclinic.com

Training

Westlake Solutions

A hands-on class for experienced Perl CGI developers that teaches how to write database-connected CGI scripts using Perl and DBI.pm. This course, along with four other courses on CGI scripting with Perl, is taught in Washington, DC; Arlington, Virginia; and on-site worldwide upon request.

See:

http://www.westlake.com/training

for more details.

Other References

In this section, we present some miscellaneous WWW links that may be of some interest to DBI users. These are not verified and may result in unknown sites or missing documents.

http://www-ccs.cs.umass.edu/db.html
http://www.odmg.org/odmg93/updates_dbarry.html
http://www.jcc.com/sql_stnd.html

AUTHOR

Alligator Descartes. Portions are Copyright their original stated authors.

COPYRIGHT