Using Perl to access DB2 for Linux

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of Contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. Tutorial tips

Should I take this tutorial?

In this tutorial you'll learn how to install and use a Perl interface to the IBM DB2 Universal Database, Personal Developer's Edition. You'll also learn by example how to query the sample database provided with the DB2 Personal Developer's Edition.

This tutorial is for a beginning or intermediate Perl programmer who wants to work with the Perl DB2 interface using DB2 for Linux.

Navigation

Navigating through the tutorial is easy:

* Use the Next and Previous buttons to move forward and backward through the tutorial.
* Use the Main menu button to return to the tutorial menu.
* If you'd like to tell us what you think, use the Feedback button.

Getting help

For technical questions about the content of this tutorial, contact the author, Thomas Burger, at *twburger@bigfoot.com* .

Thomas W. Burger is a programmer, writer, teacher, and systems analyst with twenty three years of industry experience.

Section 2. Required hardware and software

What hardware is required?

You'll need an x86 200 MHz (400 Mhz recommended), 256 MB of RAM, and 800 MB of free disk space.

The computer must have 256 MB of RAM to compile the DB2 interface. The DB2 interface itself will take about 400 MB of drive space. The DBI and DB2 interfaces will take about the same amount.

Because some of the software files (which I'll describe next) that you'll need to download from the Web are large, they will download more quickly if you have a high-speed (cable or DSL) Internet connection, but it is not required.

What software is required?

All of the software used in this tutorial is free. You'll need a standard installation of any current Linux distribution with developers components installed. Specific requirements are:

* Any current version of Linux. This tutorial was tested with Red Hat Linux 7.0. A Linux kernel 2.12 or later will work.
* Perl. This tutorial was tested with Perl 5.6.0. IBM officially supports Perl 5.004_04 or later.
* glibc 2.1.2 or later
* libncurses 4.x
* libstdc++ 2.9.0
* The rpm (at least version 3.0), gzip and tar utilities.

The IBM Developer Kit for Java, 1.1.8 or 1.3 is an optional component, but it is required if you want to use the DB2 Control Center to administer your databases using a graphical user interface, or if you want to create or run Java applications, including stored procedures and user-defined functions.

What required software will be installed?

Using Perl with DB2 for Linux requires the following additional software to be installed:

DBI

DBI is a database interface module for Perl. It defines a set of methods, variables, and conventions that provide a consistent database interface independent of the actual database being used. The DBI version required is 0.93 or newer.

http://cpan.valueclick.com/modules/by-category/07_Database_Interfaces/DBI/

or

http://www.symbolstone.org/technology/perl/DBI/index.html

What required software will be installed? (continued)

DBD::DB2

You'll need Release 0.74 of the DB2 UDB driver (DBD::DB2) for the Perl Database Interface. The DBD::DB2 driver works in conjunction with the DBI to access DB2 UDB.

http://www.perl.com/CPAN-local/modules/by-module/DBD/

What required software will be installed? (continued)

The public domain korn shell

The korn shell is required to install DB2 in Linux. This shell may not be part of some standard installations and must be added before DB2 installation can occur. The file pdksh-5.2.14-x.i386.rpm (x is for release number, which is 2 for Red Hat 6.2 distribution and 8 for the latest; either is fine) will install the public domain korn shell pdksh and it can be run from the bash shell simply by invoking it as pdksh. The DB2 installation can then be run from the install directory s000510.personal with the command ./db2setup.

http://www.redhat.com/swr/i386/pdksh-5.2.14-8.i386_dl.html

or

ftp://ftp.redhat.com/pub/redhat/redhat-7.0/i386/RedHat/RPMS/pdksh-5.2.14-8.i386.rpm

or the Red Hat 6.2 distribution CD (the 7.0 release does not have it).

What required software will be installed? (continued)

IBM DB2 Universal Database Personal Developer's Edition, version 7.1

You'll need the DB2 Application Development Client v6 or later that is included with the DB2 Personal Developer's Edition and the DB2 Universal Developer's Edition:

http://www-4.ibm.com/software/data/db2/udb/

Section 3. The Perl DB2 components, explained

What is Perl?

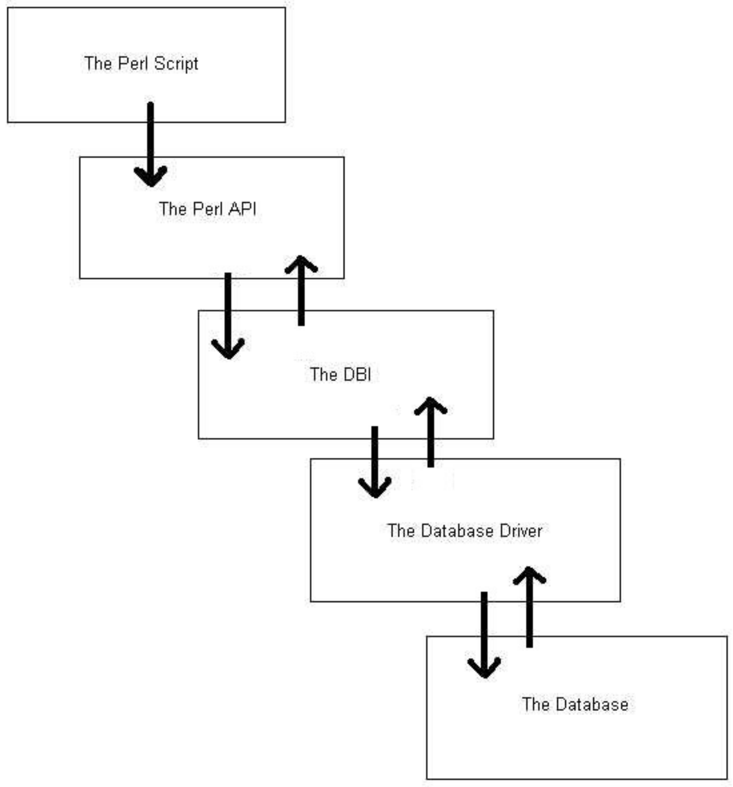
Perl is a scripting language. This means it is not compiled into an executable program but run as a set of instructions (a script) that are interpreted when they are run.

Perl is an acronym that stands for "practical extraction and report language." Perl is referred to as the "Swiss Army Knife" of languages. It is very powerful and flexible. Larry Wall, a linguist, while working as a systems administrator at NASA in the late 1980s, developed Perl. He created it to make the task of building reports easier.

Perl is now used to perform many tasks. It is perhaps the most popular language for CGI (common gateway interface) programming on the Web. The reason for this is because Perl is a very powerful text processor, and Web programming is mostly about text processing. Also, Perl is generally considered easier to use than C/C++, Java, or Tcl and was available earlier than Python.

The DBI

The DBI is a database interface module for Perl. It defines a set of methods, variables, and conventions that provide a consistent database interface independent of the actual database being used. The DBI gives the API a consistent interface to any database that the programmer wishes to use.



The DBD::DB2

The DBD::DB2 is the DB2 database driver for the DBI. This driver is not specifically designed to work with Perl. It is designed to work with the DBI. The DBI itself is specifically designed to interface with the Perl API.

The IBM DB2 Universal Database Personal Developer's Edition, version 7.1

The DB2 database being used in this tutorial is the free Personal Developer's Edition.

DB2 is a product family that runs on IBM and non-IBM machines. DB2 is supported on systems such as Sun and Hewlett-Packard as well as operating systems such as Windows, Linux, Sun's Solaris Operating Environment, HP-UX, NUMA-Q, AIX, OS/2. It is also supported on handheld device operating systems such as the Windows CE and the Palm Computing platforms.

Section 4. Installing the Perl DB2 components

Step 1: Install the public domain korn shell

Introduction to pdksh

The public domain korn shell pdksh is required to install DB2. The installation script is written specifically to use this shell and no other. If your system has the korn shell, skip to Step 2: Install the IBM DB2 Universal Database Personal Developer's Edition, version 7.1 on page 11.

Installation is easy. Check for the file pdksh-5.2.14-x.i386.rpm (x for release number, 2 for Red Hat 6.2 distribution and 8 for the latest) on your Linux CD. If you do not have it, you can download it as a tar.gz file or as an RPM file. Either file will install the public domain korn shell pdksh, and you can run it from the bash shell simply by invoking it as pdksh on the command line.

TAR file installation of pdksh

The pdksh-5.2.14.tar.gz file is about 469 KB and contains all of the source files and build scripts to create the pdksh shell. It is designed for a gcc compiler and a POSIX environment.

Download and uncompress pdksh

Download the pdksh tar.gz file from *ftp://ftp.cs.mun.ca/pub/pdksh/pdksh-5.2.14.tar.gz* .

A file that ends in the suffixes .tar.gz indicates that the files are combined into a single file that is a tape archive built with the tar utility (.tar) and it has been compressed with the gzip utility (.gz).

To uncompress and separate the original files type at the command prompt:

```
# gzip -dc pdksh-5.2.14.tar.gz | tar -xvf-
```

This will uncompress the file and send the uncompressed result as an input (using the pipe |) to the tar utility.

The command options used in the gzip utility are -d (decompress) and -c (use standard output and keep the original files unchanged). The tar utility options used are -x (extract), -v (verbose, detailed output of activities), and -f- (use the file being supplied by the pipe from gzip).

The tar utility can be used by itself. The command tar -xvfz pdksh-5.2.14.tar.gz will both decompress and extract the installation files. The z option tells tar to uncompress the file using gzip.

Compiling and installing pdksh

Read the README file for details about installation and configuration.

Before installing any new software into Linux it is usually required to be logged on as root. Do this by issuing the su (super user) command at the command prompt and typing the root password, or by logging out and back in as root.

The quick way to get pdksh up and running is to execute the following commands:

```
./configure  # this sets up the compilation
make         # this compiles pdksh
make check   # this is optional, it tests the make and is a good idea
make install # will install /usr/local/bin/ksh
             # and /usr/local/man/man1/ksh.1
```

RPM file installation of pdksh

The pdksh-5.2.14-8.i386.rpm file is about 180 KB and is a Red Hat Package Manager file. Using RPM file to modify and update a Linux distribution is easier and has some advantages over using tar.gz files.

For more details on the use of RPM files, see *http://www.redhat.com/support/manuals/RHL-7-Manual/ref-guide/ch-rpm.html* .

Download and install the pdksh RPM file

First get the pdksh-5.2.14-8.i386.rpm file from *ftp://rpmfind.net/linux/redhat/redhat-7.0/i386/en/RedHat/RPMS/pdksh-5.2.14-8.i386.rpm* .

Installation of any RPM file is easy. If you are working in a graphical environment you can simply double click the file in the file browser, or from the command prompt, issue the RPM command: rpm -Uvh pdksh-5.2.14-8.i386.rpm.

The command options used are -U (upgrade) and -v (display the rpm status or current actions line), and -h (places fifty hash marks (#) between the rpm output lines).

Step 2: Install the IBM DB2 Universal Database Personal Developer's Edition, version 7.1

Introduction

Now that the public domain korn shell clone pdksh is installed, DB2 can be installed. DB2 is installed from a tar.gz file like the korn shell can be. If you have pdksh already installed and skipped the section on installing it you may want to review the section for details on setting up a software installation from a tar.gz file.

The Personal Developer's Edition of the IBM DB2 Universal Database allows you to program a DB2 interface without having to purchase the full product. Although a limited version, it has the same interface as the commercial versions. It comes with a ready-made database for testing and instruction purposes that will be used in this tutorial.

Download and uncompress the DB2 TAR file

The DB2 download file is found at *http://www6.software.ibm.com/dl/db2pde/db2pde-p* . You will have to register with IBM to get the file. This is free and involves supplying a user name and password and answering a marketing questionnaire.

The file that will be downloaded is linuxpecmn.tar and it is about 78 MB.

Recreate the installation files with the tar -xvf linuxpecmn.tar command. This will place the files into an install directory s000510.personal that is created in the current directory.

Install DB2

The installation of the Personal Developer's Edition of the IBM DB2 Universal Database requires the use of the public domain korn shell clone pdksh. If pdksh is not installed, please go to Step 1: Install the public domain korn shell on page 9 .

At the command prompt, change the shell being used to the korn shell by simply typing in pdksh. Use cd s000510.personal to make the current directory the DB2 installation directory. Run the installation with the command ./db2setup. The ./ placed before the name of the korn shell script name is required to tell the shell that the file is found in the current directory.

The installation asks a lot of questions. To be safe, install everything. At a minimum you will have to install the following optional components:

* DB2 client
* The sample database
* Application development tools
* Run time environment

Testing the DB2 installation

The DB2 installation will set up a user account db2inst1 with password ibmdb2. This account can run DB2. To run DB2 type db2start at the command prompt. This will begin the DB2 session. You can prepend any queries with DB2 like this:

```
db2start
db2 CONNECT TO sample
db2 select * from staff where dept = 20
db2stop
```

or simply type db2 and work from the db2 prompt:

```
db2start
db2
db2=> CONNECT TO sample
db2 => select * from staff where dept = 20

ID     NAME      DEPT   JOB   YEARS  SALARY    COMM
------ --------- ------ ----- ------ --------- ---------
10     Sanders   20     Mgr   7      18357.50  -
20     Pernal    20     Sales 8      18171.25  612.45
80     James     20     Clerk -      13504.60  128.20
190    Sneider   20     Clerk 8      14252.75  126.50
4 record(s) selected.
db2 => quit
DB20000I The QUIT command completed successfully.
db2stop
```

Step 3: Install the DBI

Introduction

The Perl database interface DBI is an abstraction level interface for the Perl API. It does not know what database it is talking to, and it acts as a standard interface between the API and the database driver. This way the programming of the interface code can remain the same no matter what the database is, and the database driver can remain the same no matter what the API is.

Download and uncompress the DBI

Download the file
http://cpan.valueclick.com/modules/by-category/07_Database_Interfaces/DBI/DBI-1.14.tar.gz
.

Then uncompress and create the installation files with the tar utility:

```
tar -xvfz DBI-1.14.tar.gz
```

Install the DBI

Correct installation requires that Perl 5 (5.004_04 or later) is installed first.

Installation is done in three steps. The first is the build that creates the software from source code. The second step is the test of the build. The last step is the installation of the software. This method of installation allows the software to be custom built for your system.

Installation commands (remember you must be the super user root to install new software and the commands and file names are case sensitive) are as follows:

```
perl Makefile.PL
make
make test
```

and finally if the test looks okay:

```
make install
```

Testing the DBI

Testing must wait until the DBD::DB2 DB2 UDB driver is installed.

Step 4: Install the DBD::DB2 driver

Introduction

Installation of the DBD::DB2 driver failed on a workstation with 128 MB of memory. After increasing the memory to 256 MB of RAM the compilation worked correctly.

The installation takes quite a long time. There are a large number of large files to compile and link. The estimated build time is 10 minutes.

Download and uncompress the DBD::DB2 driver

Download the file:

http://www.perl.com/CPAN-local/modules/by-module/DBD/DBD-DB2-0.74.tar.gz .

And create the installation with:

```
tar -xvfz DBD-DB2-0.74.tar.gz
```

Install the DBD::DB2 driver

Installation involves running the build Makefile.PL and this requires that a DB2_HOME environment system variable be set to the location of the DB2 directory. This is typically /usr/IBMdb2/V7.1.

To set this variable's value enter, at the shell prompt (bash):

```
export DB2_HOME="/usr/IBMdb2/V7.1"
```

Make sure you are root.

The procedure is the same as with the DBI:

```
perl Makefile.PL
make
make test
make install
```

The build and installation of DBD::DB2 can take quite a while. It is large. The estimated build time is 20 minutes.

Testing the DBD::DB2 driver

You can now test the interface by writing a Perl script and running it. A script is presented and analyzed in the next section.

Section 5. Perl DB2 programming

The Perl DB2 example program

The programming example being used (and more documentation) is available at:

http://search.cpan.org/doc/IBMTORDB2/DBD-DB2-0.74/DB2.pod

The panels in this section will describe and analyze the Perl DB2 example program.

Summary of the Perl example program

This program accesses the sample database "sample" provided with db2. The program uses the employee photo table in the sample database to produce a set of files in the current directory with the employee number as the name and .GIF as the extension. Each file is a photo of an employee in GIF format.

The Perl program uses the DB2 interface by issuing two SELECT SQL statements. The first retrieves a set of rows, or a result table, that is used to provide input for the second statement. This statement is then executed in a while loop using the parameter markers ? to be replaced with the two elements of the @row array $row[0] and $row[1], which correspond to the empno and photo_format columns in the first SQL statement.

The use statements

The start of the program makes the DBI and DBD::DB2 subroutines ready for use in the Perl code.

```
# This is a note about the Perl being used.
#!/usr/local/bin/perl
use DBI; # load the DBI
use DBD::DB2::Constants; #load the DB2 constant v
use DBD::DB2 qw($attrib_int $attrib_char $attrib_
```

Subroutines are both loaded and imported into your namespace with a use statement. Packages can be nested inside other packages as they are in the "use DBD::DB2::Constants;" line. Any double colons in the module name are translated into your system's directory separator, usually / in Linux.

The use statements

Read the README file for details about installation and configuration. The third use statement:

```
use DBD::DB2 qw($attrib_int $attrib_char $attrib_
```

imports variables in a list. The qw() is a macro that allows the programmer to avoid having to type in quotes and commas. So the array names can be created and set in the standard way like this:

```
@names = ("Tom", "Dick", "Harry");
```

which is the same as

```
@names = qw(Tom Dick Harry);
```

Creating new DB2 attribute types

New, custom DB2 attributes can be created by the programmer:

```
# An example of the syntax for creating a new att
# $attrib_int is a reference to the original "ter
$attrib_dec = { %$attrib_int,
'db2_type' => SQL_DECIMAL,
'SCALE' => 2,
'PRECISION' => 31 };
```

The first line in the assignment %$attrib_int, (the % explicitly references it as a hash) gives the new anonymous hash reference (this is a reference to a value that did not previously exist much like void * p = malloc(100); in C programming) the contents of the existing hash. The SCALE and PRECISION are then changed for the new attribute hash.

More about hash values

A hash literal contains pairs of values to be interpreted as a key and a value:

```
($colormap{'red'}, $colormap{'blue'}, $colormap{
# same as colormap assignment above
%colormap = ('red',0x00f,'blue',0x0f0,'green',0xf
```

An often more readable variant of the above is to use the => operator between key/value pairs. The => operator is more visually distinctive synonym for a comma and it allows the left-hand operand to be interpreted as a string, if it qualifies as a legal identifier and a bareword (a set of characters outside a string that Perl does not recognize). For example:

```
%colormap = (
red => 0x00f,
blue => 0x0f0,
green => 0xf00,
);
```

Debugging options

The next line of code:

```
#$DBI::dbi_debug=9; # increase the debug output i
```

lets the programmer set the DBI debug level if the comment symbol # is removed. Debug levels can be 0 through 9. The default is 0 and 3 is usually detailed enough.

Opening the DB2 table

Here the handle to the sample database "sample" is assigned a value:

```
                                 # Open a connection and set LongReadLen to maximu
                                 $dbh =
connect("dbi:DB2:sample","","", { LongReadLen => 102400 } );
                                 if (!defined($dbh)) { exit; }
```

If the value is undefined, the program aborts. The value LongReadLen sets the number of bytes that can be read from the database into a table column at one time.

Getting information from DB2

Here a simple SQL statement is created, prepared and executed:

```
                                 # Note in the following sequence, that the stater
                                 # no parameter markers, which makes the executior
                                 # just prepare and execute.
                                 $stmt = "SELECT empno, photo_format FROM emp_phot
                                 photo_format = 'gif';";
                                 $sth = $dbh->prepare($stmt);
                                 $sth->execute();
```

The comment about not using parameter markers is with regard to the more complex statement to follow.

SQL statements

This SQL statement tells DB2 to do create a table in memory that has two columns containing the values from the fields empno and photo_format in the sample database table emp_photo where the field photo_format has the value 'gif' in the record:

```
                                 $stmt = "SELECT empno, photo_format FROM emp_phot
```

SQL statement preparation and execution

The prepare method gives the SQL statement to DB2 for evaluation and prepares DB2 to receive the execute command. The execute method, if all is successful, will create, in memory, a result table of the values found:

```
$sth = $dbh->prepare($stmt);
$sth->execute();
```

Using parameter markers in an SQL statement

This new SQL statement uses the parameter marker (or placeholder) value ?. Note that there are no quotes around the marker question marks. This means that DB2 will expect parameters to be bound to these values prior to an execute method being called. These values will be provided from the table created by the first execute method call.

```
# $row[0] is the empno from the database and $row
# image type. In this case, the type will always
$stmt = "SELECT picture FROM emp_photo WHERE empn
photo_format = ? ;" ;
# prepare statement, which contains two parameter
$pict_sth =
prepare($stmt);
```

SQL loops

A while loop is controlled by the retrieval of information from the table created by the first SQL statement. The array @row is set to the two row values using the method fetchrow:

```
while( @row = $sth->fetchrow ) {
```

Creating an output file

The program will produce output of files containing the employee pictures in GIF format. The files are named using the employee number and the picture type extension:

```
# create an output file named empno.type in the c
open(OUTPUT,">$row[0].$row[1]") || die "Can't ope
```

This will create a file handle OUTPUT that is a file that will be written to. The > character indicates that the file will be created or overwritten if it exists. The empno and photo_format field values stored in the array @row are used to name the output file. The next line causes the file to write the data assigned to it in binary mode. There will be no translations or insertions that would occur if the file was in text mode.

```
binmode OUTPUT;
```

Binding parameters to an SQL statement

Perl only has string and number scalar data types. All database types that aren't numbers are bound to markers as strings and must be in a format the database will understand.

```
$pict_sth->bind_param(1,$row[0]);
$pict_sth->bind_param(2,$row[1]);
```

These lines bind the values in the @row array to the parameters in the markers in the second SQL statement as strings by default. DBD::DB2 supports the following methods of binding parameters:

For input-only parameters:

```
$rc = $sth->bind_param($p_num, $bind_value);
$rc = $sth->bind_param($p_num, $bind_value, $binc
$rc = $sth->bind_param($p_num, $bind_value, \%att
```

Binding parameters to an SQL statement (continued)

For input/output, output or input by reference:

```
                                 $rc =
bind_param_inout($p_num, \$bind_value, $max_len);
                                 $rc =
bind_param_inout($p_num, \$bind_value, $max_len, $bind_type);
                                 $rc =
bind_param_inout($p_num, \$bind_value, $max_len, \%attr)
```

The \%attr attribute hash reference parameter can be used to hint at the data type the markers should have. Usually the driver is only interested in knowing if the marker/placeholder should be a number or a string:

```
                                 $sth->bind_param(1, $value, { TYPE
> SQL_INTEGER });
```

A shortcut for this is to pass the data type directly:

```
                                 $sth->bind_param(1, $value, SQL_INTEGER);
```

The data type for a placeholder cannot be changed after the first bind_param call, but it can be left unspecified, and in that case it defaults to the previous value. There must be one bind_param call per parameter per execution of the SQL statement.

Getting file data

The second SQL statement is executed and the @row array is reassigned the value of the fetchrow method of the pict_sth handle. The Perl print operator sends the array content to the file and the file is closed. Then the finish method is called to destroy the binary data stored in memory that has been written to the file.

```
$pict_sth->execute();
@row = $pict_sth->fetchrow;
print OUTPUT $row[0];
@row = "";
close(OUTPUT);
$pict_sth->finish();
} # end of the while loop
```

Finishing the program

The blob cursor refers to the binary data and the SQL table pointer to it. A blob is slang for a large piece of binary data. The cursor is the standard name of the SQL data pointer.

As the comment states, the closing of the pict_sth- >execute method after the loop is probably not required, but is there in case the while loop terminates before the finish is called. The next finish method call closes and frees the memory allocated for the result table of the first SQL statement execute method, and the disconnect method closes the database interface.

```
# redundantly close the blob cursor -- should be
$pict_sth->finish();
# close selection criteria cursor
$sth->finish();
$dbh->disconnect();
```

Section 6. Wrapup

Summary

With Perl, DBI, DBD::DB2, and the IBM DB2 UDB installed on your Linux system together with a basic understanding of Perl DB2 programming, achieved by the analysis of the example program, you can now continue to explore the use of DB2 with Perl.

The information learned in this tutorial can be applied to other databases and to other programming languages. You could now install Java and a Java interface to DB2. You could install MySQL and the DBD::mysql to access the MySQL database with the Perl DBI.

Perl resources

Perl DB2 program example

http://search.cpan.org/doc/IBMTORDB2/DBD-DB2-0.74/DB2.pod

ActivePerl

http://www.activestate.com/Products/ActivePerl/index.html

CPAN: Comprehensive Perl Archive Network

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

Introduction to Perl

MU Information & Access Technology Services Short Course, University of Missouri - Columbia, 20 April 1999:
http://www.cclabs.missouri.edu/things/instruction/perl/perlcourse.html

O'Reilly, The Source for Perl

http://www.perl.com/pub

Web Developer's Virtual Library - Perl

http://www.wdvl.com/Authoring/Languages/Perl

Reference books

Beginning Perl, Simon Cozens, Peter Wainwright, Wrox Press Ltd., 2000.

Advanced Perl Programming, Sriram Srinivasan, O'Reilly & Associates Inc., 1997.

DB2 resources

DB2 Version 7.1 for Linux HOWTO

http://www.linuxdoc.org/HOWTO/DB2-HOWTO/index.html

DB2 Magazine

http://www.db2mag.com/

IBM's DB2 Universal Database 7.1 for Linux shines

http://www.linuxworld.com/linuxworld/lw-2000-09/lw-09-db2_p.html#resources

The IBM DB2 Product Family

http://search.cpan.org/doc/IBMTORDB2/DBD-DB2-0.74/DB2.pod

The DB2 Self Study Course

http://www-4.ibm.com/software/data/db2/selfstudy/

Your feedback

Please let us know whether this tutorial was helpful to you and how we could make it better. We'd also like to hear about other tutorial topics you'd like to see covered. Thanks!

For technical questions about the content of this tutorial, contact the author, Thomas Burger, at *twburger@bigfoot.com* .

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The Toot-O-Matic tool is a short Java program that uses XSLT stylesheets to convert the XML source into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML.