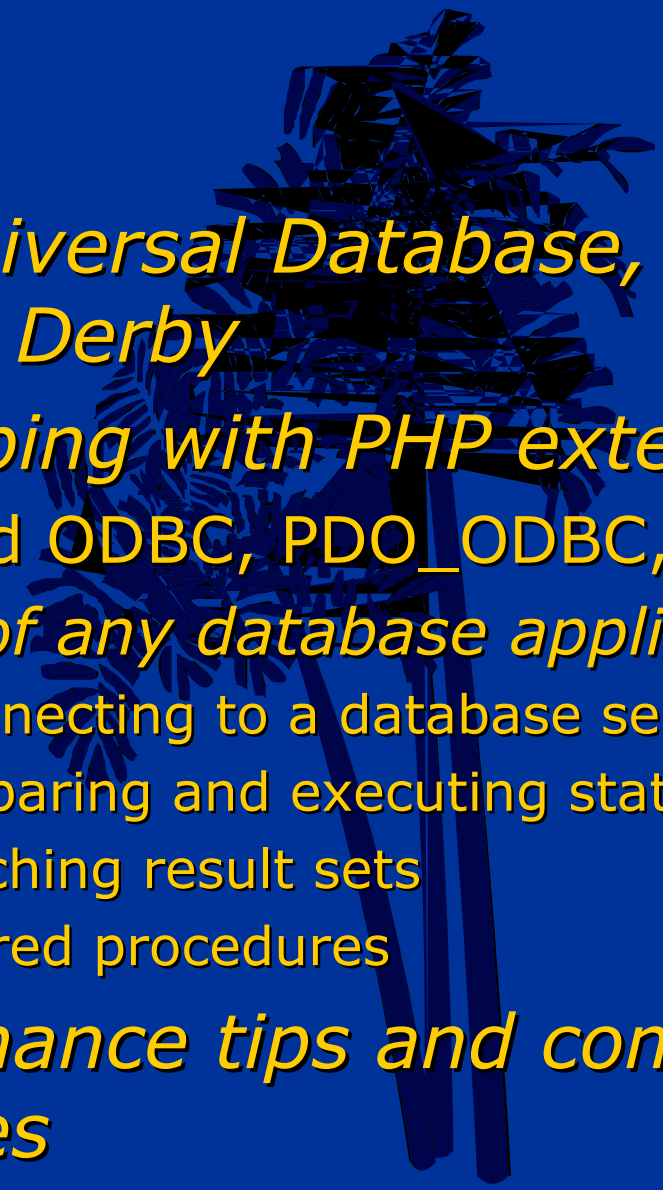
IBM DB2 Universal Database, Cloudscape, and Apache Derby

Dan Scott

dbs@php.net,
dan.scott@ca.ibm.com

Agenda



- *DB2 Universal Database, Cloudscape, Apache Derby*
- *Developing with PHP extensions*:
  - Unified ODBC, PDO_ODBC, ibm_db2
  - *90% of any database application:*
    - Connecting to a database server
    - Preparing and executing statements
    - Fetching result sets
    - Stored procedures
- *Performance tips and common mistakes*

Why IBM and PHP?

- "Radical simplicity"
  - Traditional infrastructure for creating applications has encouraged layers of abstraction (J2EE: entity beans)
  - One of IBM's "simple Web app" attempts resulted in Net.Data:

Net.Data, a full-featured and easy to learn scripting language, allows you to create powerful Web applications... Net.Data reached its end of support date on September 30, 2004 for Windows and UNIX... The successor product for Net.Data is IBM WebSphere Application server.

Why IBM and PHP?

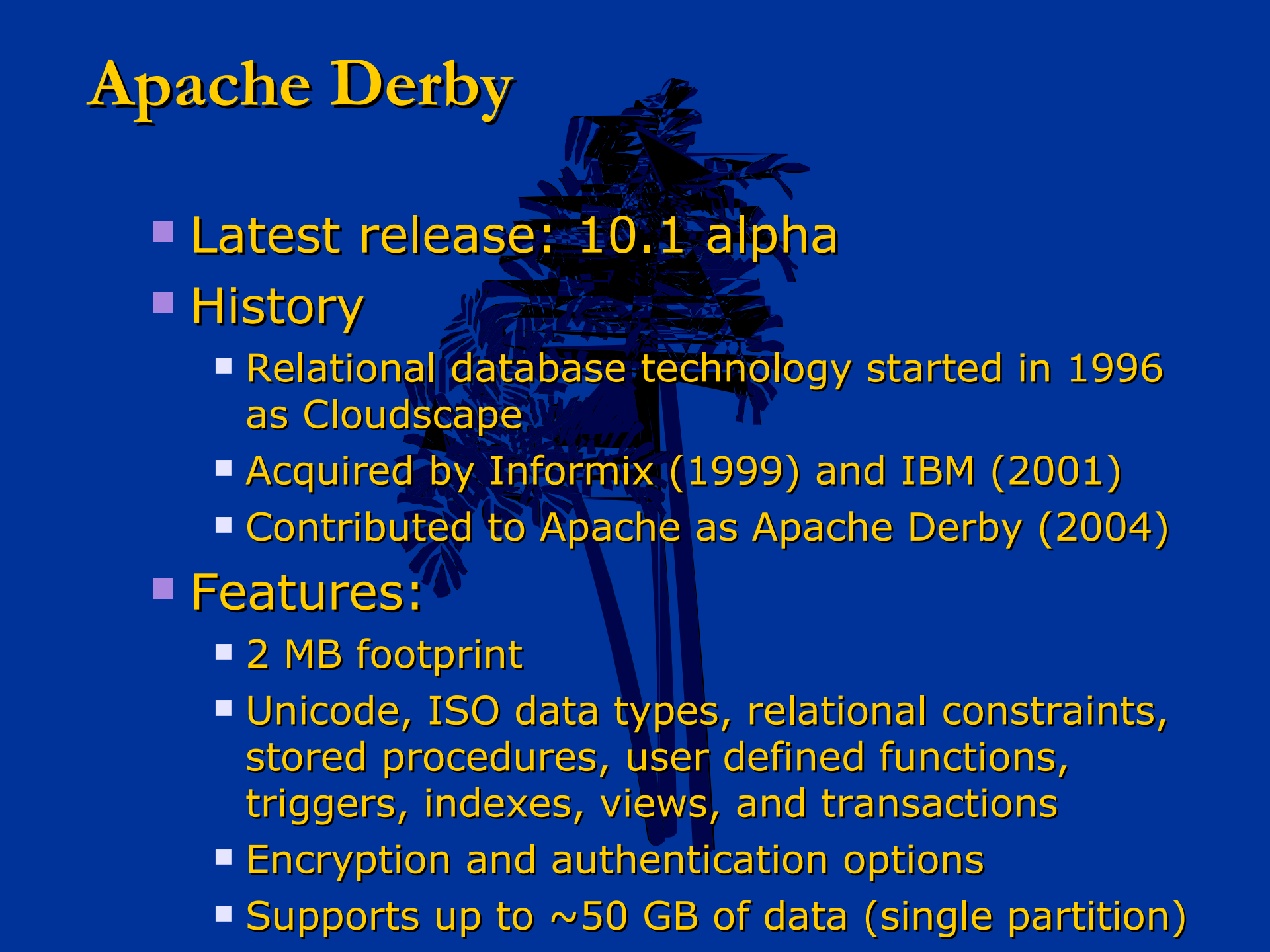
- PHP is a barrier-free programming language, supporting the full continuum from direct to abstract
- PHP itself is open-source, with a broad community contributing extensions, documentation, and help
- PHP is the language of choice for many OSS applications: blogs, wikis, CMS, Web mail, CRM

IBM DB2 Universal Database

- *The* name in enterprise databases
  - Available for z/OS, iSeries, Linux, UNIX, and Windows
  - IBM -- originator and leader of relational database technology
- Fast, powerful, reasonably priced
  - Start with DB2 UDB Express (maximum 2 CPUs, 4GB RAM) and then scale as your business grows:
    - Scale up with SMP (64-way or beyond)
    - Scale out with a cluster of up to 1,024 servers
  - DB2 on Linux holds top positions in TPC-H* performance *and* price/performance
  - DB2 on AIX holds top position in TPC-C* performance

*See Appendix for TPC publication information. Results referenced are current as of April 30, 2005.For the latest TPC results, visit www.tpc.org.

Apache Derby



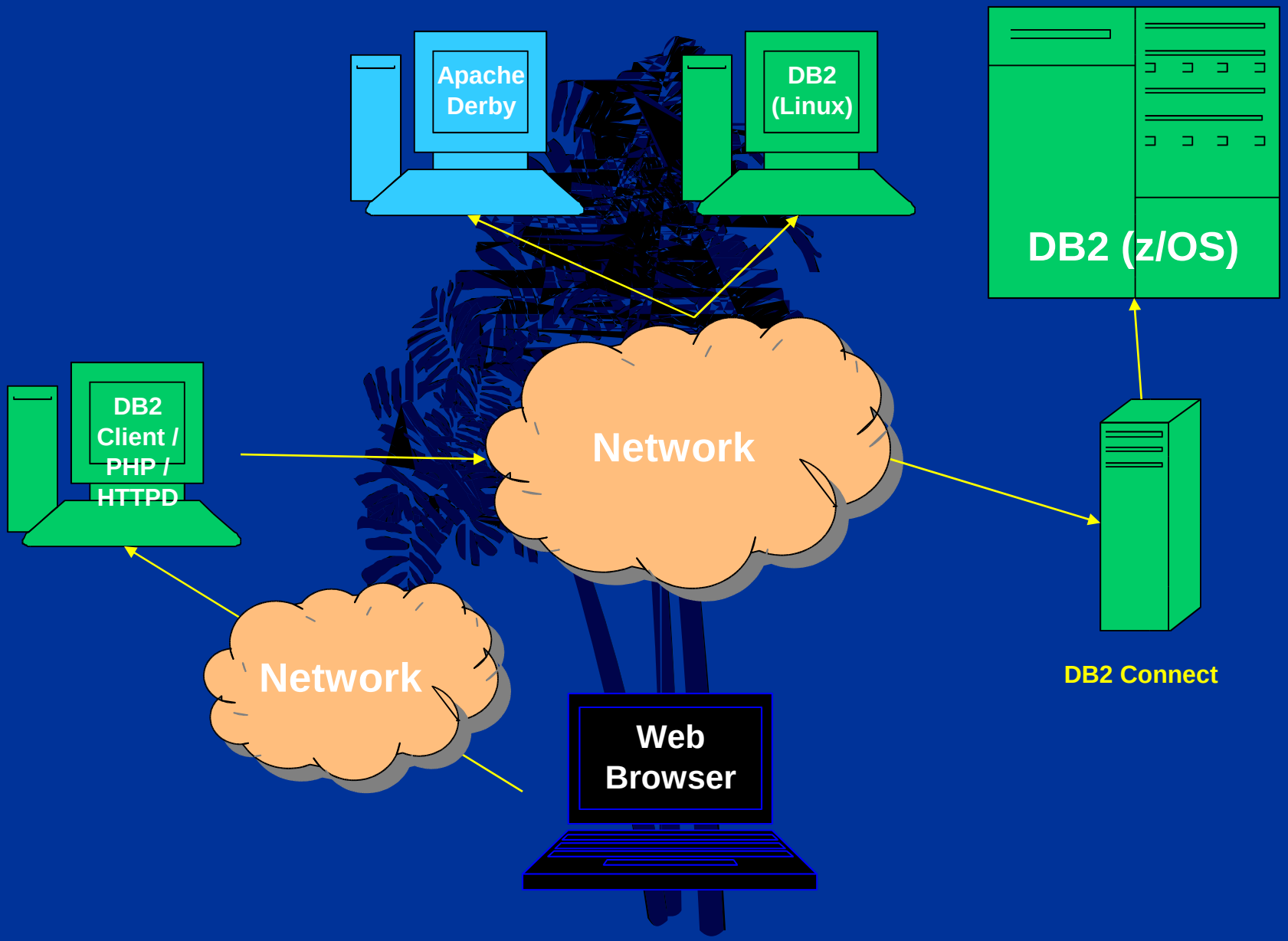
- Latest release: 10.1 alpha
- History
  - Relational database technology started in 1996 as Cloudscape
  - Acquired by Informix (1999) and IBM (2001)
  - Contributed to Apache as Apache Derby (2004)
- Features:
  - 2 MB footprint
  - Unicode, ISO data types, relational constraints, stored procedures, user defined functions, triggers, indexes, views, and transactions
  - Encryption and authentication options
  - Supports up to ~50 GB of data (single partition)

IBM Cloudscape

- IBM Cloudscape =
  - Stable snapshot of Apache Derby
  - + Java & native installers
  - + Java Runtime Environment
  - + Documentation
  - + Optional IBM support

Many servers, a single client

- DB2 and Apache Derby database servers speak DRDA protocol
  - DB2 client contains CLI (and ODBC, and Java) libraries that talk DRDA
  - Therefore, one PHP extension built on the DB2 client can talk to many different database servers
  - DB2 Runtime Client is a free download from http://www.ibm.com/db2/udb/support/downloadv8.html


Apache Derby
DB2 (Linux)
DB2 (z/OS)
DB2 Client / PHP / HTTPD
Network
Network
DB2 Connect
Web Browser

Cataloging a database

- Start your DB2 command line processor
- (Unix only) - Inherit the DB2 instance profile
  - `bash$ source /home/db2inst1/sqllib/db2profile`
- Catalog the database "node" (server)
  - `bash$ db2 CATALOG TCPIP NODE warehouse REMOTE warehouse.db2geek.com SERVER 50000`
- Catalog the database (DB2)
  - `bash$ db2 CATALOG DATABASE db2mart AT NODE warehouse`
- Catalog the database (Derby)
  - `bash$ db2 CATALOG DATABASE db2mart AT NODE warehouse AUTHENTICATION SERVER`

Test your connection

- Connect to your database
  - `bash$ db2 CONNECT TO db2mart [USER username [USING PASSWORD password]]`
  - DB2 defaults to using your current, logged-in user ID
  - Derby *always* requires a user ID and password
- If the connection fails, ensure:
  - The database server has been started:
    - **(DB2)** `bash$ db2start`
    - **(Derby)** `bash$ ./setNetworkServerCP.ksh && ./startNetworkServer.ksh`
  - The DB2 server setting **DB2COMM** includes **TCPIP**:
    - `bash$ db2set DB2COMM=TCPIP`

A wealth of PHP options

- Three PHP drivers support DB2:
  - Unified ODBC (ext/odbc)
  - Extension for DB2 and Cloudscape (ibm_db2)
  - PHP Data Objects: PDO_ODBC
- All three PHP drivers are offered under open-source licenses and available from http://www.php.net

Unified ODBC (ext/odbc)

- Built in to PHP core
  - Normally compiled against a generic ODBC driver manager
  - Can be compiled against DB2 libraries (./configure --with-ibm-db2) for native access
- Drawbacks:
  - Scrollable cursors: slow & warning-prone
  - No support for OUT/INOUT stored procedures

Extension for DB2 and Cloudscape (ibm_db2)

- Available from the PECL repository under the Apache 2.0 License
  - Developed and supported by IBM
  - Full featured support for stored procedures and LOBs*
  - Fast
- Drawbacks:
  - New extension = undoubtedly some bugs, somewhere

PHP Data Objects: PDO_ODBC

- Fast, light, pure-C standardized data access interface for PHP
  - PDO_ODBC compiles directly against DB2 libraries for native access to DB2
  - Standard database API simplifies application development
  - Fast, light weight OO interface
- Drawbacks:
  - PHP 5 only; still under development

Compiling the ibm_db2 extension

- Prerequisites:
  - DB2 client with development headers (/opt/IBM/db2/V8.1/include/sqlcli.h)
- Through PEAR installer:
  - `bash$ pear install ibm_db2-beta`
- Through CVS:
  - `bash$ phpize`
  - `bash$ ./configure --with-IBM_DB2`
  - `bash$ make && su -c 'make install'`

Configuring PHP for ibm_db2

- Ensure extension dir is set in php.ini
- Add `extension=ibm_db2.so` to php.ini
- Source the DB2 instance environment
  - `. /home/db2inst1/sqllib/db2profile`
- Adjust Web server environment:
  - Modify /etc/init.d/httpd or /etc/init.d/apache[2] to source db2profile

Or just use Zend Core for IBM

- Free download from http://ibm.com/software/data/info/zendcore/
- Installs DB2 client, PHP, ibm_db2 extension, and Unified ODBC extension
- Includes Web-based configuration UI
- Available for Linux x86, x86-64, Linux on POWER, and AIX

Your first ibm_db2 application

```
<?php
$sql = "SELECT name, breed FROM ANIMALS
    WHERE weight < ?";
$conn = db2_connect($database, $user, $password);
$stmt = db2_prepare($conn, $sql);
$res = db2_execute($stmt, array(10));
while ($row = db2_fetch_assoc($res)) {
    print "{$row['NAME']} is a {$row['BREED']}.\n";
}
?>
```

```
Pook is a cat.
Bubbles is a gold fish.
```

Cataloged connections

- Assuming:

```
$database = 'db2mart';
$user = 'db2inst1';
$password = 'ibm_db2';
```

- Unified ODBC

```
$conn = odbc_connect($database, $user, $password);
```

- ibm_db2

```
$conn = db2_connect($database, $user, $password);
```

Uncataloged connections

- Assuming:

```
$host = 'localhost'; $port = 50000;
$DSN = "DRIVER={IBM DB2 ODBC DRIVER};PORT=$port;
   HOSTNAME=$host;DATABASE=$database;PROTOCOL=TCPIP;";
```

- Unified ODBC

```
$uconn = odbc_connect($DSN, null, null);
```

- ibm_db2

```
$uconn = db2_connect($DSN, null, null);
```

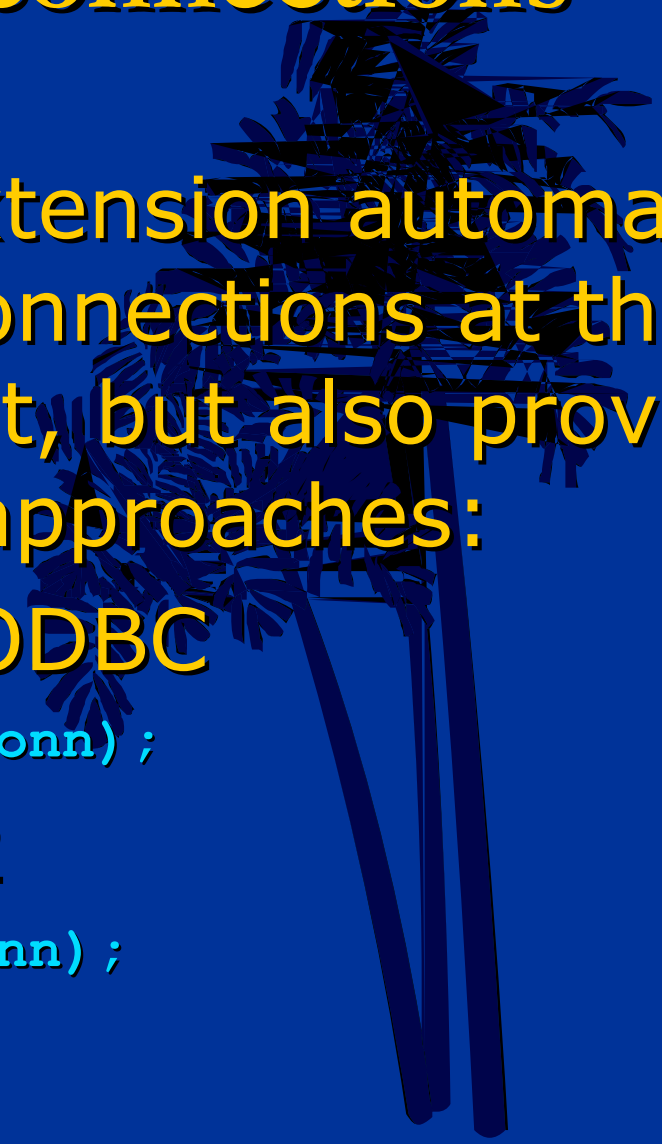
Connection tips

- Why does it take so long to make the first connection to a DB2 database?
  - DB2 automatically releases system resources after a period of inactivity; the first connection makes it allocate system resources.
  - You can avoid this by keeping DB2 vigilant:

```
bash$ db2 ACTIVATE DATABASE db2mart
```

- What about persistent connections?
  - Dangerous, but can be useful for read-only operations
  - Unified ODBC and ibm_db2 offer *_pconnect()

Closing connections



- Every extension automatically closes connections at the end of the script, but also provides explicit approaches:
- Unified ODBC

```
odbc_close($conn);
```

- ibm_db2

```
db2_close($conn);
```

Error handling

- Unified ODBC and ibm_db2 offer procedural interfaces:
  - To return an SQLSTATE value:
    - `odbc_error()`
    - `db2_conn_error()`,
    - `db2_stmt_error()`
  - To return an error message:
    - `odbc_errormsg()`
    - `db2_conn_errormsg()`
    - `db2_stmt_errormsg()`

Issuing SQL statements

- DB2 and Derby support prepared statements for superior performance and security
- Use ? placeholders to bind values by position:

```
$sql = 'SELECT name FROM animals WHERE breed = ?';
```

  - Unified ODBC*:

```
$stmt = odbc_prepare($conn, $sql);
$rc = odbc_execute($stmt, array('cat'));
```

  - ibm_db2

```
$stmt = db2_prepare($conn, $sql);
$rc = db2_execute($stmt, array('cat'));
```

Unified ODBC: cursor issues

- By default, Unified ODBC uses a dynamic scrollable cursor, which DB2 and Derby servers don't support
  - Every fetch and prepare request results in an SQL warning as the cursor downgrades automatically to a dynamic keyset-driven cursor
  - Also causes severe performance problems
- With a cataloged connection, you can automatically downgrade the cursor to forward-only by setting a CLI configuration parameter:

```
bash$ db2 UPDATE CLI CFG FOR SECTION db2mart USING
   PATCH2 6
```

Fetching data by row

- Unified ODBC:

```
// Indexed by column number
odbc_fetch_into($stmt, $row);
// Indexed by column name
$row = odbc_fetch_assoc($stmt);
// PHP object; column names = properties
$row = odbc_fetch_object($stmt);
```

Fetching data by row

- ibm_db2

```
// Indexed by column number
$row = db2_fetch_into($stmt);
// Indexed by column name
$row = db2_fetch_assoc($stmt);
// Indexed by column number and name
$row = db2_fetch_both($stmt);
// PHP object; column names = properties
$row = db2_fetch_object($stmt);
```

Calling stored procedures

- Unified ODBC
  - Provides no means of explicitly binding parameters
  - You can only call stored procedures that accept input (IN) parameters*
- PDO and ibm_db2
  - Provide methods for binding output (OUT) and input/output (INOUT) parameters for stored procedures

Calling stored procedures: ibm_db2

- Rather than dynamically binding an array of input parameters in **`db2_execute()`**, call **`db2_bind_param()`**
  - Passing DB2_PARAM_INOUT tells ibm_db2 to bind the parameter as an INOUT parameter
  - Be careful: instead of passing in a variable, you pass in a string containing the name of a variable

```
$colour = 'red';
$stmt = db2_prepare('CALL puree_fruit(?)');
db2_bind_param($stmt, 1, 'colour',
    DB2_PARAM_INOUT);
db2_execute();
print("After CALL the colour is: $colour");
```

Performance tips: COUNT

- Counting rows returned by SELECT
  - Common (bad) pattern – does not work for forward-only cursors, very slow for scrollable cursors

```
$stmt = db2_exec('SELECT * FROM animals', array(CURSOR =>
   DB2_SCROLLABLE));
$num = db2_num_rows($stmt);
if ($num) { ... }
```

  - Better pattern:

```
$stmt = db2_exec('SELECT COUNT(*) FROM animals');
list($num) = db2_fetch_into($stmt);
if ($num) {
  $stmt = db2_exec('SELECT col, col FROM animals');
}
```

  - Best pattern:

```
$stmt = db2_exec('SELECT col, col FROM animals');
while ($row = db2_fetch_into($stmt)) { ... }
```

Performance tips: cursors

- Scrollable cursors can be s-l-o-w and create lock escalation problems
  - Also, Derby currently does not support scrollable cursors
- Forward-only cursors meet the needs of 99% of all applications
  - Unified ODBC defaults to dynamic scrollable cursors, which DB2 server has to downgrade to keyset-driven
    - Major networking overhead in client / server mode
    - Use UPDATE CLI CFG workaround to force downgrade at client and avoid overhead
  - PDO and ibm_db2 default to fast "forward-only" cursors, but give you the option of forcing a keyset-driven cursor for a given statement
    - If you just want a dirty snapshot of a small set of data, PDO offers `PDOStatement::fetchAll();` you can iterate through the returned array of rows at will

Performance tips: transactions

- DB2, Cloudscape, and Derby default to AUTOCOMMIT on
- This is a good default -- avoids locking escalation
- However, if a single script performs a number of INSERT / UPDATE / DELETE operations, consider using a transaction + db2_commit() or db2_rollback() instead

Five fast migration tips: #1 -- odbc -> ibm_db2

- ibm_db2 was designed to map easily to Unified ODBC:
  - Function names:
    - s/odbc_/db2_/g
  - db2_fetch_into() returns a row, rather than accepting a reference
    - `$row = db2_fetch_into($stmt)` vs. `odbc_fetch_into($stmt, $row)`
  - Column functions are 0-indexed
  - Row functions are 1-indexed

Five fast migration tips: #2 -- Counting rows

- db2_num_rows() does *not* return the number of rows affected by a SELECT
  - Issue the SELECT and short-circuit if first fetch returns FALSE
  - Issue SELECT COUNT(*) with the same SELECT predicates
  - Issue the SELECT using a scrollable cursor (supported by DB2 only, and slow)

Five fast migration tips: #3 -- Numeric values

- DB2 client is strictly SQL-compliant: character values are delimited, numeric values are not
  - For example, this fails if `id` is an INTEGER column:

```
INSERT INTO animals (id, name) VALUES
  ('1', 'Pook')
```

  - Use prepared statements with placeholders and let PHP convert for you

```
$stmt = db2_prepare($conn, 'INSERT INTO
  animals (id, name) VALUES (?, ?)');
db2_execute($stmt, array('1', 'Pook'));
```

Five fast migration tips: #4 -- Replacing LIMIT

- The LIMIT clause is a non-standard extension to SQL popularized by MySQL
  - DB2 servers (1): use the ANSI SQL ROW_NUMBER OVER() syntax
  - DB2 servers (2): use scrollable cursors
  - DB2, Cloudscape, and Derby all support sequences and auto-generated columns; use a simple WHERE clause

Five fast migration tips: #5 -- Metadata functions

- Both Unified ODBC and ibm_db2 offer database metadata functions
  - Unified ODBC uses the PHP 4 smashedtogetherfunctionnames()
  - ibm_db2 uses PHP 5 separated_function_names(), but also defines aliases for the deprecated naming style

Two performance pointers: #1 -- ACTIVATE DATABASE

- By default, DB2 is a good system citizen and uses as few resources as possible.
  - However, in this state it takes time to allocate resources for the first connection.
  - To prime the connection pump for a specific database:
  - `bash$ db2 ACTIVATE DATABASE dbname`

Two performance pointers: #2 -- Prepared statements

- Issuing SQL with **`db2_exec()`** allows you to execute 'one-shot' statements
- However, preparing and executing statements with **`db2_prepare()`** / **`db2_execute()`** enables the database to cache its access plan
- This can even improve performance for statements that are only issued once

Common problems: TCP/IP connections

- If your first attempt to connect to a database via TCP/IP fails:
  - Is the database running?
    - `(DB2) bash$ db2start`
    - `(Derby) bash$ ./startNetworkServer.sh`
  - Can you connect from the command line?
    - `bash$ db2 CONNECT TO db USER user`
  - Does the server know it should talk TCP/IP?
    - `(DB2 server) bash$ db2set DB2COMM=TCPIP`

Common problems: LIMIT

- DB2 and Derby do not support the non-standard LIMIT clause
- DB2 implements the SQL standard ROW_NUMBER() OVER clause:

```
SELECT * FROM (
  SELECT name, breed, ROW_NUMBER()
  OVER (ORDER BY id)
  AS rows FROM animals
) AS myanimals
WHERE rows > skip AND rows <= (n+skip)
```

- With Derby, you need to implement unique row identifiers as part of your database schema

Common problem: Case-sensitivity

- Standard states that all non-delimited SQL object identifiers are folded to upper case
- This causes all kinds of confusion with metadata methods or fetching rows into associative arrays
- User names versus schema names:

```
CREATE TABLE db2inst1.animals (name VARCHAR(128))
    db2_tables(... 'DB2INST1' ...) returns results
    db2_tables(... 'db2inst1' ...) returns no results
```

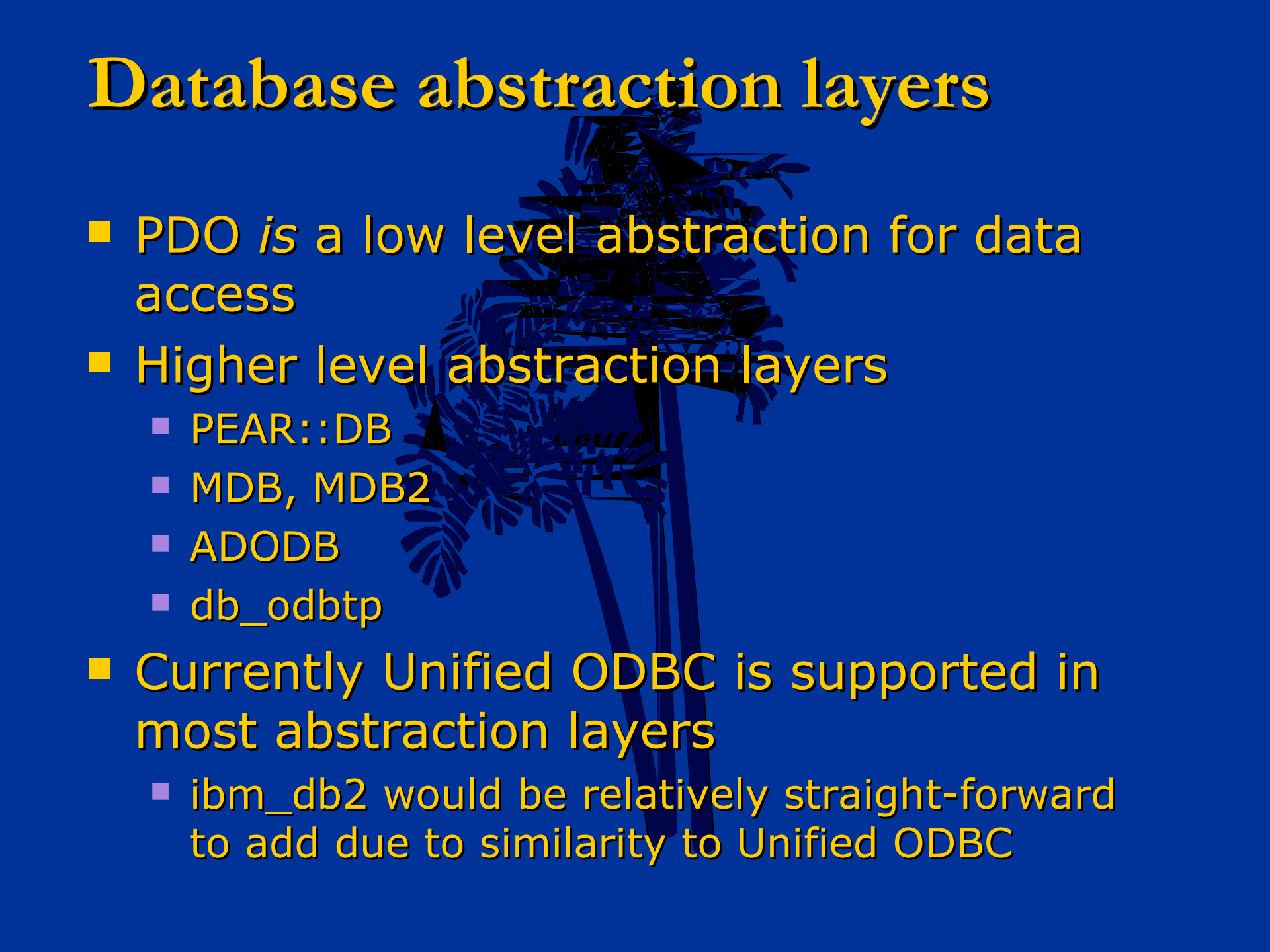
- Column names:

```
SELECT name, weight FROM animals;
$row = db2_fetch_assoc(...);
    $row['NAME'] exists
    $row['name'] does not exist
```

Common problem: SELinux

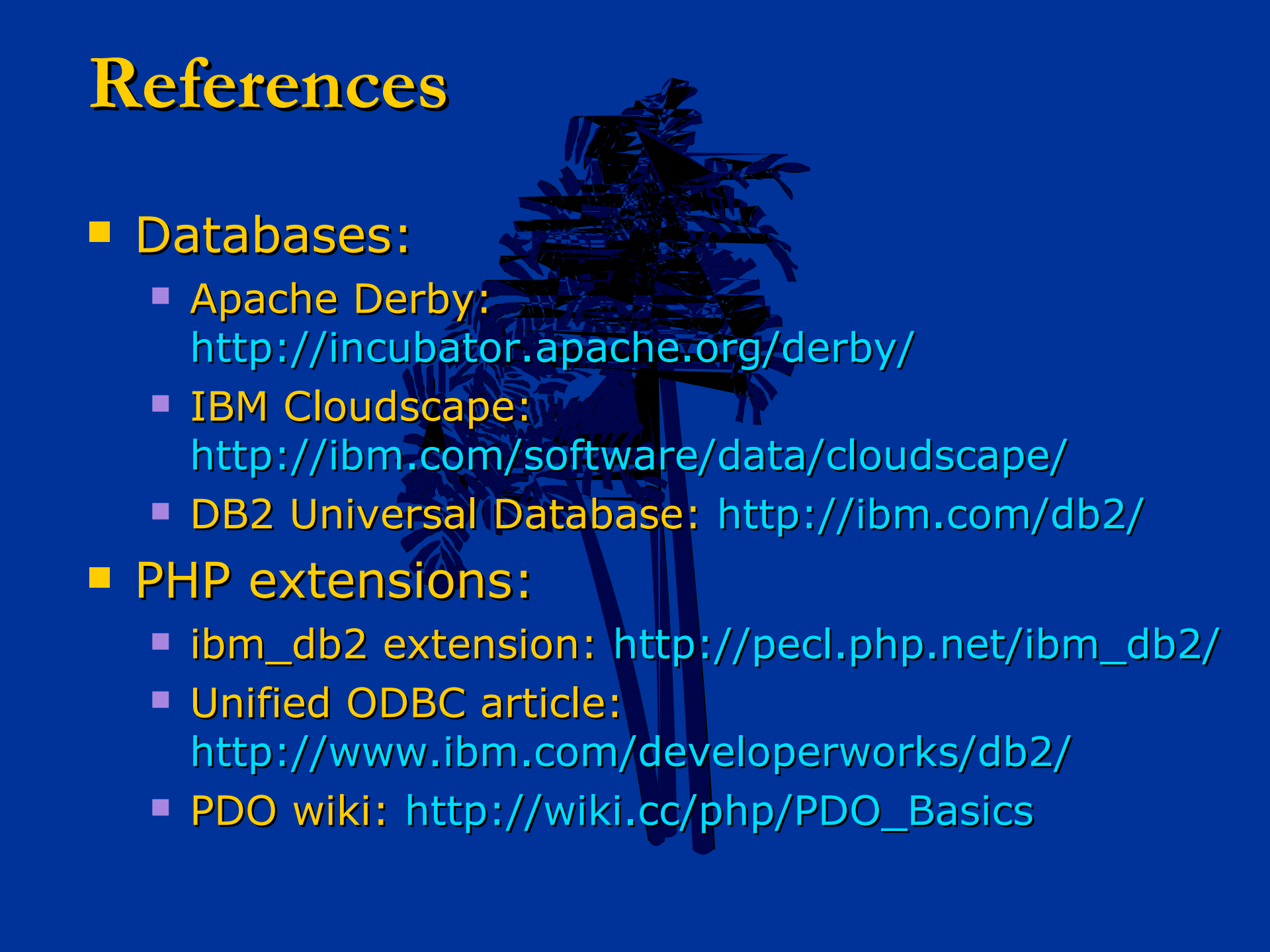
- Fedora Core 3, Red Hat Enterprise Linux 4.0 added SELinux policies for Apache, standard environment
- DB2 client libraries are located in a 'non-standard' location
- Result: default SELinux policy prevents Apache from loading DB2 libraries ☹
  - Solution #1: Disable SELinux (`/usr/sbin/setenforce 0` or modify `/etc/selinux/config`)
  - Solution #2: Modify SELinux policy (??)

Database abstraction layers



- PDO *is* a low level abstraction for data access
- Higher level abstraction layers
  - PEAR::DB
  - MDB, MDB2
  - ADODB
  - db_odbtp
- Currently Unified ODBC is supported in most abstraction layers
  - ibm_db2 would be relatively straight-forward to add due to similarity to Unified ODBC

References



- Databases:
  - Apache Derby: http://incubator.apache.org/derby/
  - IBM Cloudscape: http://ibm.com/software/data/cloudscape/
  - DB2 Universal Database: http://ibm.com/db2/
- PHP extensions:
  - ibm_db2 extension: http://pecl.php.net/ibm_db2/
  - Unified ODBC article: http://www.ibm.com/developerworks/db2/
  - PDO wiki: http://wiki.cc/php/PDO_Basics

TPC Publication Details

- TPC Benchmark, TPC-C, TPC-H, tpmC are trademarks of the Transaction Processing Performance Council. For further TPC-related information, please see http://www.tpc.org/
- Results as of May 10, 2005
  - TPC-H performance:
    - 100GB: IBM DB2 UDB Enterprise Server Edition 8.1 on IBM eServer 325 running Suse Linux Enterprise Server 8; Metrics: 12,216 QphH; $71.00/QphH; Available: 11/08/2003
    - 300GB: IBM DB2 UDB Enterprise Server Edition 8.1 on IBM eServer 325 running Suse Linux Enterprise Server 8; Metrics: 13,194 QphH; $65.00/QphH; Available: 11/08/2003
    - 1000GB: IBM DB2 UDB Enterprise Server Edition 8.2 on IBM eServer x346 running Suse Linux Enterprise Server 9; Metrics: 53,451 QphH; $33.00/QphH; Available: 02/14/2005
  - TPC-H price/performance:
    - 1000GB: IBM DB2 UDB Enterprise Server Edition 8.2 on IBM eServer x346 running Suse Linux Enterprise Server 9; Metrics: 53,451 QphH; $33.00/QphH; Available: 02/14/2005
  - TPC-C performance:
    - IBM DB2 UDB Enterprise Server Edition 8.2 on IBM eServer p5 595 64p running IBM AIX 5L 5.3; Metrics: 3,210,540 tpmC; $5.19/tpmC; Available: 05/14/2005