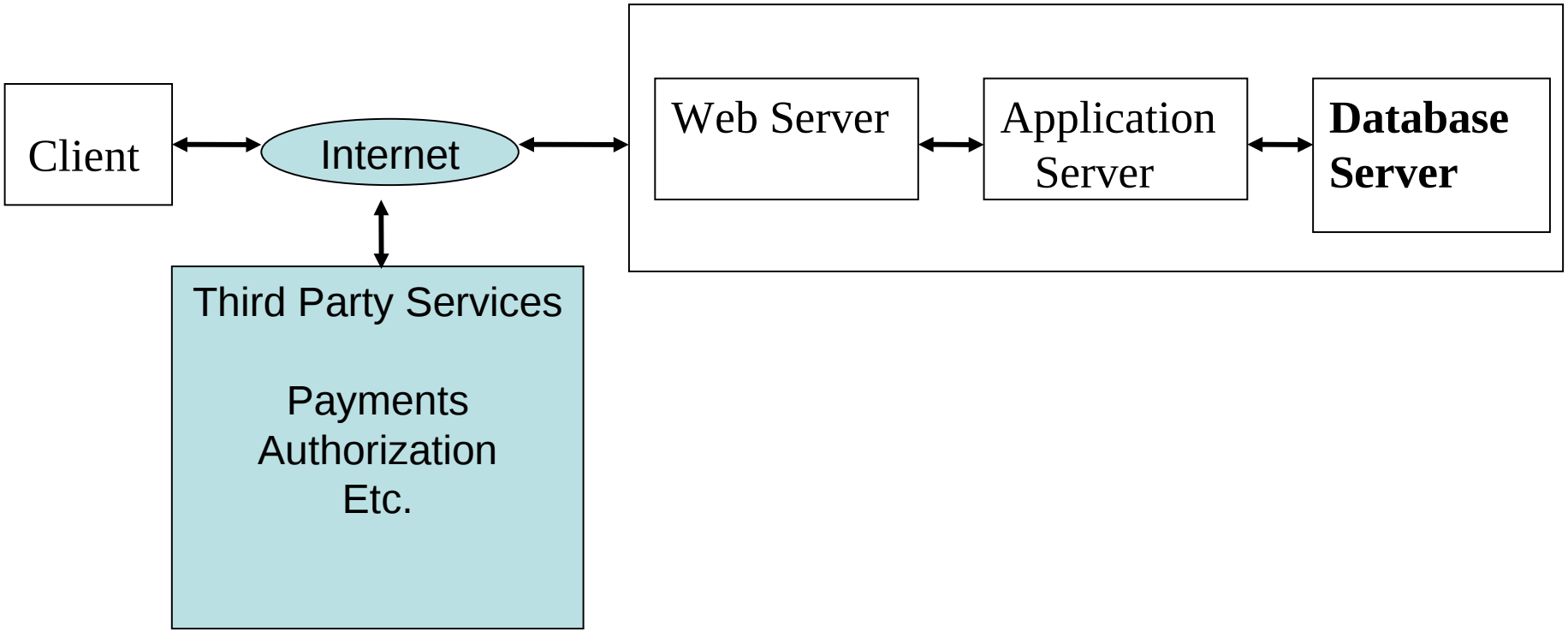
Systems Programming & Scripting

Lecture 19: Database Support

Typical Structure of a Web Application



PHP-based Design

- The PHP-based design of a web application is based on separating the application in several independent components.
- For specific tasks existing applications or libraries should be used, eg. Databases
- PHP is a scripting language, whose task is to connect these applications.
- For (small) web-specific tasks, PHP provides built-in libraries (session tracking)

PHP Database Support

- A database can be accessed in two ways using PHP.
  - Database-specific extension.
  - the *PEAR DB library*.

- The database-specific extension is a driver (interface) to a specific DataBase Management System (DBMS) with specific:
  - function names,
  - parameters and
  - error-handling.

PHP Database Support (cont'd)

- The other way is to access the required database using the *PEAR DB library* which comes with PHP.
  - PEAR DB is a *database-independent* library which hides database specifics allowing the same PHP script to access many DBMSs.
  - This enables the development of *portable code* that is not tied to one specific database.
  - One potential disadvantage of the PEAR DB approach is that it doesn't support certain features specific to a particular database.
  - Another potential disadvantage of the PEAR DB code is that it can be slower than code written in database-specific extensions.

PEAR DB Functions

$db = DB::connect *(string, option)*

- is used to open a connection to a database
- *string* is a data source name (DSN), encoding user, password and database to access
- *option* specifies whether the connection is persistent, optimised etc
- The result is a handle to the database or an error value

PEAR DB Functions

$res = $db->query($sql)

- issues an SQL query to the database, for which $db is a handle
- *$sql* is a string encoding the SQL query
- The full power of the SQL query language is available

PEAR DB Functions

$res->fetchRow()

- is used to get the next row from an SQL query, identified by $res

$res->fetchInto($row)

- gets the next row and stores the contents into the array $row
- $row is indexed by 0, 1...
- With the option DB_FETCHMODE_ASSOC, $row is indexed with the names of the columns in the table
- Both functions return NULL to indicate the end of the query results

PEAR DB Functions

$res->free()

- frees the resources held by the result of an SQL query

$db->disconnect()

- closes the connection to the database identified by $db

Placeholders in Queries

- Similar to printf, an SQL query can be constructed with unknowns, written as ? in the string, replaced by variables:

```
$books = array(array('Foundation', 1951),
               array('Foundation2', 1953));
foreach ($books as $book) {
  $db->query('INSERT INTO books (title,
    year) VALUES (?,?)', $book);
}
```

Placeholders in Queries

- The following placeholders are available
  - ? … a quoted string or number
  - | … an unquoted string or number
  - & … a filename, whose contents to include

Prepare/Execute

- If a query should be issued many times, it is more efficient to split it into a prepare and an execute phase:

$comp = $db->prepare($sql)

- returns a compiled query object

$res = $db->execute($comp, $arr)

- issues the query, filling in elements in $arr for placeholders in the sql string

Shortcuts

- These shortcuts perform a query and get the result in one go:
  - *$db->getOne($sql)* … fetches first column of first row of an sql query
  - *$db->getRow($sql)* … fetches first row of an sql query
  - *$db->getCol($sql, $col)* … fetches column $col from an sql query
  - *$db->getAll($sql)* … fetches an array of all rows of an sql query

Information in Query Response

- The following functions provide information on the response of a query:
  - *$resp->numRows()* … the number of rows in the reponse
  - *$resp->numCols()* … the number of columns in the response
  - *$resp->affectedRows()* … the number of rows affected by an INSERT etc
  - *$resp->tableInfo()* … information on types and flags returned by a SELECT query

Example: Book Store Query

```
<html><head><title>Books</title></head>
<body>

<table border=1>
<tr><th>Book</th><th>Year</th><th>Author</th></tr
>
<?php
// connect
require_once 'DB.php';
$db =
DB::connect("mysql://user0:passw0rd@localhost/lib
rary0");
if (DB::iserror($db)) {
  die($db->getMessage());
}
```

Example: Book Store Query

```
// issue a query
$sql = "SELECT books.title, books.year,
                books.author
        FROM books, authors
        WHERE books.authorid=authors.authorid
        ORDER BY books.year ASC";

$q = $db->query($sql);
if (DB::iserror($q)) {
  die ($q->getMessage());
 }
```

Example: Book Store Query

```
// generate table
while ($q->fetchInto($row)) :
?>
<tr>
<td><?php echo $row[0] ?> </td>
<td><?php echo $row[1] ?> </td>
<td><?php echo $row[2] ?> </td>
</tr>
<?php endwhile; ?>
</table>
<?php $q->free(); $db->disconnect(); ?>
</body>
</html>
```

Database-specific Interfaces

- PHP also provides direct support of database-specific interfaces to generate, modify and query databases.
- Common databases such as MySQL, Oracle etc are supported.
- These interfaces often provide additional facilities, not found in the PEAR interface.
- In general, these interfaces are also more efficient, which can be a design consideration when working with huge databases

Example

- PHP script for generating Item Detail web page dynamically.
  - The script requires an *itemID* parameter entered by the user.
- The script uses the PHP built in session tracking mechanism to keep track of the number of users visiting.
- It is assumed that there is a *bookstore* database hosted by mysql DBMS.

Example

```
<?php // Starting/loading user session to keep track
   // of number of visits
    session_start();
    session_register('visits');
    ++$visits;  // counter
?>
<HTML><HEAD><TITLE>
PHP Item Details Page
</TITLE></HEAD>
<BODY>
<H1> Item Details Page </H1>
<?php
   // get itemID parameter contained in the request.
   $itemId = $_GET['itemID'];
```

Example (cont'd)

```
    $DBhost = "localhost";
    $DBuser = "root";
    $DBName = "bookstore";
    mysql_connect($DBhost, $DBuser);
    mysql_select_db($DBName);

//Execute a sql query getting item
//information from item and author
// tables.
 $query = "SELECT i_title, a_fname,a_lname
             FROM item, author
         WHERE i_id=$itemId AND i_a_id=a_id";
 $result = mysql_query($query);
```

Example (cont'd)

```
    // Generate HTML code containing no of visits
   // made by a customer and item information.
   echo "<H3>Number of Visits: ", $visits, "</H3>";
   echo "<H3>Item ID: ", $itemId, "</H3>";
   echo "<H3>Book Title: ",
       mysql_result($result, 0, 'i_title'), "</H3>";
   echo "<H3>Author First Name: ",
  mysql_result($result, 0, 'a_fname'), "</H3>";
   echo "<H3>Author Last Name: ",
  mysql_result($result, 0, 'a_lname'), "</H3>";
   mysql_close();
?>
</BODY>
</HTML>
```

PHP on TRAX

- Yet another high-level database interface is “PHP on TRAX”.
- It is based on concepts developed in “Ruby on Rails”.
- It provides a deep embedding of database access into the programming language.
- This saves considerable amount of coding
- See the web pages for details:

http://www.phpontrax.com/

Summary

- PHP provides integrated support for interacting with (external) databases.
- Such databases often run in the background to manage the *state* in a web application.
- Two interfaces are provided:
  - Database-independent (PEAR)
  - Database-specific interfaces for MySQL etc