
Advanced Data Base
8trd157
Lab6
Internet access to an Oracle database with
PHP and a http server (Apache)
(due : June 24th 2013)

Paul Girard, Ph.D.

Objective of lab6

Design and implement a web application (*user transaction*) accessing a user schema of an Oracle database installed on a Unix server *dimensxcn1.uqac.ca* having an http server (*Apache*) and PHP configured to have an OCI access to Oracle.

Pre-requisite procedures

1. First, the installation of Apache and the PHP compiler must be done accordingly to the access type (*native, odbc, ...*), the DBMS type and version (*ex. Oracle 11g with OCI*). Like a CGI program, *phpinfo.php* will execute the PHP function *phpinfo()* which displays all installation parameters.

Example of some information displayed by *phpinfo()*

oci8	
OCI8 Support	enabled
Revision	\$Revision: 1.183.2.18.2.3 \$
Oracle Version	8.1
Compile-time ORACLE_HOME	/disk/disk1/oracle/OraHome1
Libraries Used	no value
oracle	
Oracle Support	enabled
Oracle Version	8.1
Compile-time ORACLE_HOME	/disk/disk1/oracle/OraHome1
Libraries Used	no value

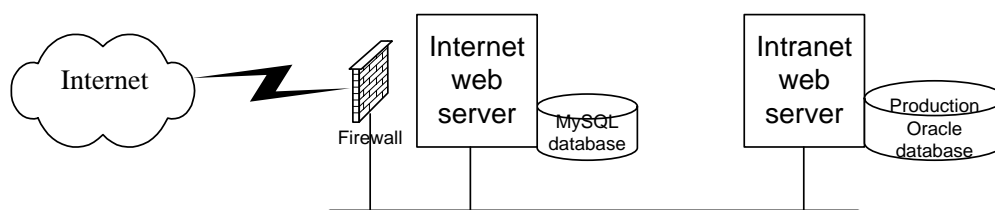
-
2. The installation of all Oracle client libraries must be done on the Apache server because this server is an Oracle client when it sends an SQL request using the OCI (*Oracle Call Interface*). Because Apache-PHP is a client, all environment variables must be specified in the PHP compiler. These variables are :

• ORACLE_HOME	• ORACLE_SID	• LD_PRELOAD
• LD_LIBRARY_PATH	• NLS_LANG	• ORA_NLS33

Example of some Apache environment variables

ORACLE_BASE	/disk/disk1/oracle
ORACLE_HOME	/disk/disk1/oracle/OraHome1
ORACLE_OWNER	oracle
ORACLE_SID	cndb

3. If the access is done with ODBC, an ODBC client must be installed in the Apache server and an ODBC server in your DBMS server. A network architecture must be supported by these drivers (*ex. TCP/IP*). On the HTTP server, the **DSN** accessing the database and the user schema must be defined without requiring a login for each access.
4. PHP functions must be used according to the DBMS, the version and the access type. Complete documentation of PHP is on the web site. The web site <http://www.php.net/manual/en/langref.php> gives the basic syntax of PHP 5.3 and <http://www.php.net/manual/en/book.oci8.php> describe the OCI functions for Oracle 8i used by 11g.
5. Normally a user account should be created for this web access with the only necessary privileges. This special user should not be the owner of the database objects and he should not access to unnecessary columns or use unnecessary SQL DML (*delete, update, insert*). Try a login to the Oracle server with this web account and check that all required transactions are correctly executed because Oracle error messages will not be displayed on the html page when these requests are sent to Oracle by a PHP program. Information will be filtered along the path from **client http request ==> Apache (http server) ==> PHP ==> OCI ==> Oracle ==> PHP ==> Apache ==> client http result ==> display the html page**.
6. A good security procedure would be to use another server machine with the required software: the same DBMS or another one like MySQL with Apache and PHP. Tables could be replicated or exported at certain intervals (*ex. use of triggers*). This way it is easier to assure the protection of a production database. It is never a good idea to open all production data of a private society to the internet. All TCP and UDP ports except TCP 80 should be disabled. TCP port 21 for FTP could be opened using a temporary tunnel protected by a password to update the tables.



Methodology

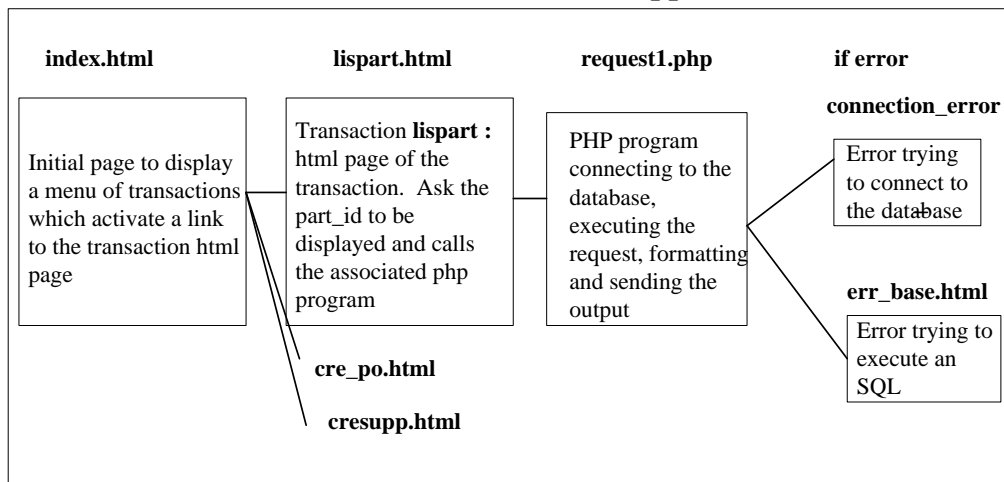
1. Download *lab6.zip* to get 5 html and 2 php files. If it does not exist create in your user account a directory called *public_html* with *read/execute* protection for all. These 7 downloaded files will be under this directory. After your modifications (*user_name*, *password*, *cursor*, *PHP variables*, *tables* and *columns names*), you can test it with the URL

<http://dimensxcn1.uqac.ca/~ora11>***

```
dimensxcn1:ora*****> mkdir public_html
dimensxcn1:ora*****> chmod 755 public_html
```

2. The program *phpinfo.php* may be executed without modification to display all internal parameters of PHP

General architecture of the application



3. Test the application by executing <http://dimensxcn1.uqac.ca/~pgirard/index.html>, this test will give you the same result presented in Annex 1 according to this architecture. Annex 2 displays the source code of each file. Annex 3 explains each PHP functions used and Annex 4 gives a short explanation on some basic language elements of PHP.
4. Edit the columns, tables, user code and password in the file *request1.php* . Make sure that PHP variables used in the form and the PHP program are the same. Then activate the page with http://dimensxcn1.uqac.ca/~ora*****/index.html to see the result. Create the most simple program and gradually add more functionalities.
5. Modify *index.html* to display all transactions for your user type (see *lab3*). **Select a transaction different from *lispart* as long as this transaction needs a data entry** ; create a link to this html page which will support this transaction like *lispart.html*. The name of the transaction will be identified at the top of the page and will ask to the user a data entry (*or multiples data entries*) using a form (*like lispart.html*). A *submit* button will activate a php page to process the transaction and display the result in html format. Each non supported transaction will be preceded by *. Each non supported transaction will activate a dummy html page (*like cre_po.html*). **Make sure to mention the data supported by your database on the html page.(example: enter a part id from 1001-1006).**

TEAM REPORT

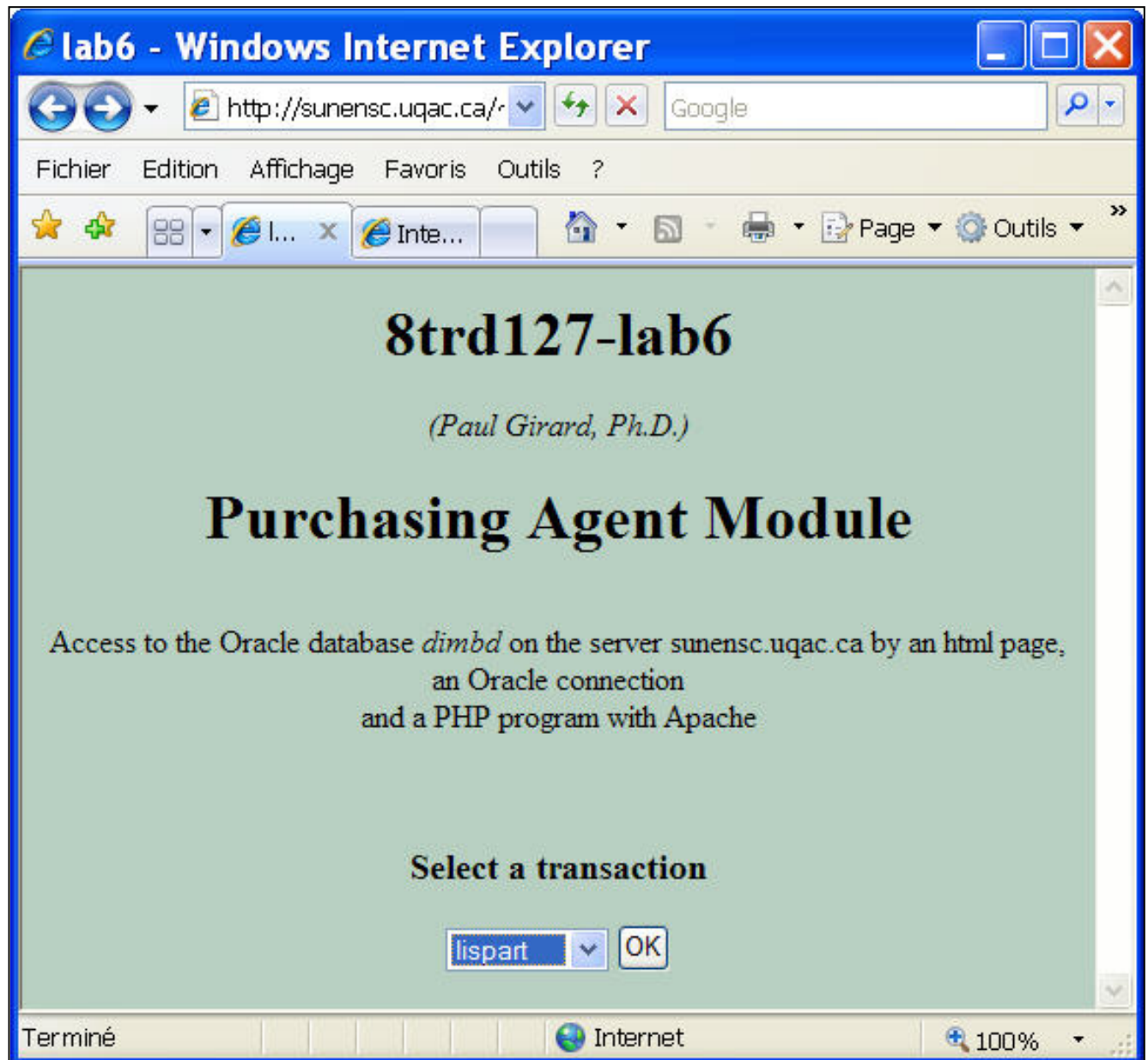
One page specifying each student name in your team, TUT code, UQAC code, user_name and password used, and the URL address to test your application.

DO NOT FORGET TO SPECIFY THE SUPPORTED TRANSACTION AND THE NECESSARY DATA TO TEST IT *(non supported transaction are shown by an * preceding the transaction in the menu).* **Any user should be able to test it on the internet.**

Include this document to the final report (in the second page after the title page).

Annex 1
Examples of different transactions ouput

Index.html



lispart.html

The screenshot shows a Windows Internet Explorer browser window with the title "List a part - Windows Internet Explorer". The address bar shows the URL "http://sunensc.uqac.ca/". The menu bar includes "Fichier", "Edition", "Affichage", "Favoris", and "Outils". The toolbar contains various icons for navigation and actions. The main content area has a light green background and contains the following text:

8trd127 - lab6

List part

Execution of the following command : *select part_id, part_name from part where part_id= like 'Spart_id%'*

Enter a part number (complete or partial from 1001-1006)

Below the text is a text input field containing "1001" and a "Submit" button.

output of request1.php

The screenshot shows a Windows Internet Explorer browser window with the title "Request SQL - Windows Internet Explorer". The address bar shows the URL "http://sunensc.uqac.ca/~pgirard/tut_". The menu bar includes "Fichier", "Edition", "Affichage", "Favoris", and "Outils". The toolbar contains various icons for navigation and actions. The main content area has a white background and contains the following text:

Version of this server : Oracle8i Enterprise Edition Release 8.1.7.4.0 - Production With the Partitioning option JServer Release 8.1.7.4.0 - Production

A form on an html page calls a PHP program executing an SQL request to an Oracle server. The PHP output is sent to Apache like a CGI program. Apache redirects this output to the HTTP client (*ex. Internet Explorer*) which displays the result

Result of the SQL request

PART ID	PART NAME
1001	'motor 1'

At the bottom of the window, the status bar shows "Terminé" and "Internet".

lispart.html (example 2)

List a part - Windows Internet Explorer

http://sunensc.uqac.ca

Fichier Edition Affichage Favoris Outils ?

8trd127 - lab6

List part

Execution of the following command : *select part_id, part_name from part where part_id= like 'Spart_id%'*

Enter a part number (complete or partial from 1001-1006)

100 Submit

Internet 100%

output of request1.php (example 2)

Request SQL - Windows Internet Explorer

http://sunensc.uqac.ca/~pgirar

Fichier Edition Affichage Favoris Outils ?

Version of this server : Oracle8i Enterprise Edition Release 8.1.7.4.0 - Production With the Partitioning option JServer Release 8.1.7.4.0 - Production

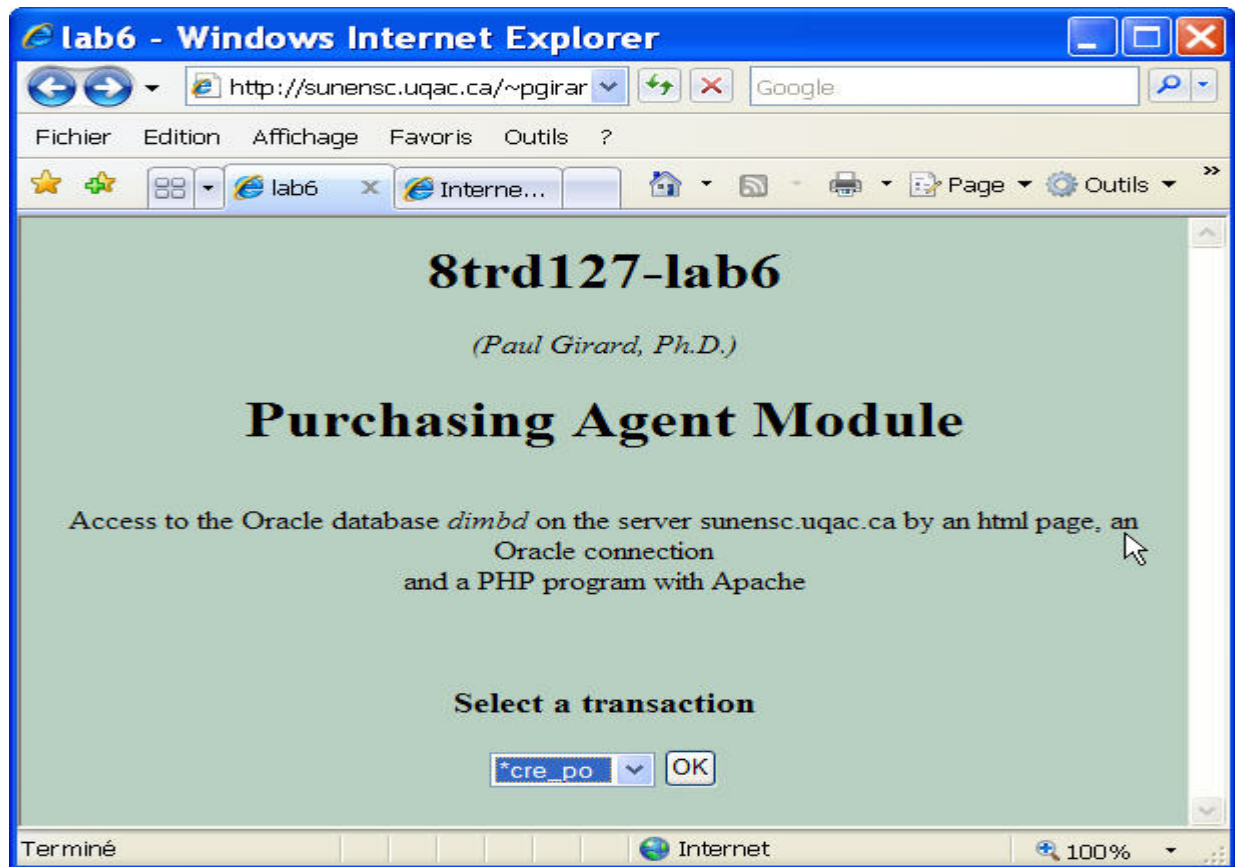
A form on an html page calls a PHP program executing an SQL request to an Oracle server. The PHP output is sent to Apache like a CGI program. Apache redirects this output to the HTTP client (*ex. Internet Explorer*) which displays the result

Result of the SQL request

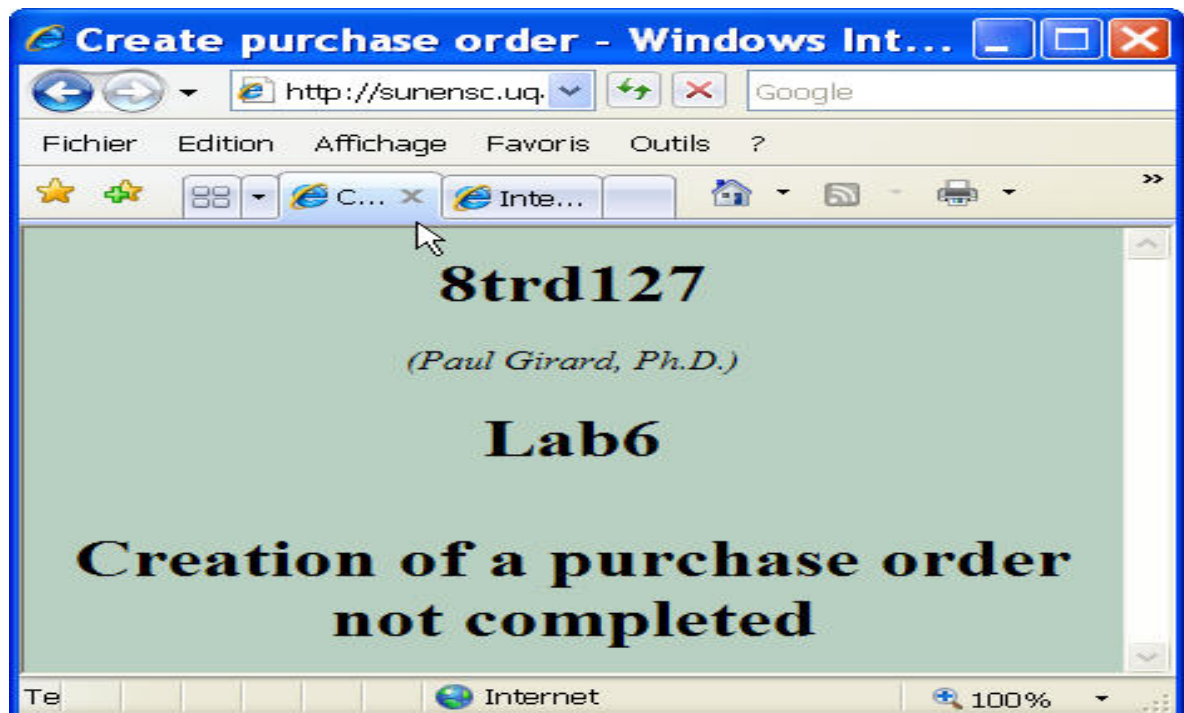
PART ID	PART NAME
1001	'motor 1'
1002	'motor 2'
1003	'batteries AA'
1004	'batteries 90C'
1005	'spark plug 1'
1006	'spark plug 2'

Terminé Internet 100%

Example of calling a transaction not supported yet



Output of cre_po.html



Annex 2

Source of each file

index.html

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html><head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="GENERATOR" content="Mozilla/4.7 [fr] (Win98; I) [Netscape]">
  <title>lab6</title>
</head>

<body text="#000000" bgcolor="#B7D0C1" link="#0000EE" vlink="#551A8B" alink="#FF0000">
<center><h1><font color="#CCFFFF"></font>8trd127-lab6</h1></center>
<center><i>(Paul Girard, Ph.D.)</i>
<h1>Purchasing Agent Module<br><br></h1></center>
<center><p>Access to the Oracle database <i>dimbd</i> on the server dimensxcn1.uqac.ca
by an html page, an Oracle connection <br> and a PHP program with Apache<br>
<br><br><h3>Select a transaction</h3>

<form name="form1">
  <select name="liste1" size="1">
    <option value="lispart.html">lispart</option>
    <option value="crepart.html">*crepart</option>
    <option value="cre_po.html">*cre_po</option>
    <option value="cresupp.html">*cresupp</option>
    <option value="creprod.html">*creprod</option>
  </select>
  <input type="button" value="OK" onClick="if (form1.liste1.selectedIndex != 5)
                                location = form1.liste1.options[form1.liste1.selectedIndex].value;
                                else alert('Pleaae make a choice.')">
</form>
</body></html>
```

lispart.html

```
<!doctype html public "-//w3c//dtd html 4.0 transitional//en">
<html><head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="GENERATOR" content="Mozilla/4.7 [fr] (Win98; I) [Netscape]">
  <title>List a part</title>
</head>
<body text="#000000" bgcolor="#B7D0C1" link="#0000EE" vlink="#551A8B" alink="#FF0000">
<center><h1><font color="#CCFFFF"></font>8trd127 - lab6<br><br>List part</h1>
<br><br><br><br>
<p><h4><center>Execution of the following command :
  <i>select part_id, part_name from part where part_id= like '$partid%'</i></h4>
<br><h3>Enter a part number (complete or partial from 1001-1006)</h3>

  <form action="request1.php" method="post">
    <input type="text" name="partid" size=4>
    <input type="submit" value="Submit"></center>
  </form>
</body></html>
```

variable read by PHP program

request1.php

```
<?php
/*****
*      Author: Paul Girard, Ph.D., UQAC
*      Date:   March 2008
*      Course: 8trd127
*      Objective: Show an example of SQL request activated by an html page
*               on the table part of a user schema defined in dimbd
*****/

*      1. Creation of a connection identifier in the user schema to the Oracle
*      database. OCIError returns false if there is a connection error.
*      The function header with the parameter Location can REDIRECT the execution to
*      another html page.
*/
$bd = "cnbd";
$connection = OCI_connect("user_name", "passwod", $bd);
if(OCIError($connection))
{
    $url = "connection_error.html";
    header("Location: $url");
    exit;
};

/*      The complete content of the result is formatted in html by the concatenation
*      of all information in the string variable $chaine0. If we use echo, the redirect
*      is no more possible. So echo is used only at the end. The header function
*      specifies a new HTTP header to use a redirect and this header must be sent before
*      any data to the client with echo. The final string is sent to Apache which will transmit
*      it to the HTTP client.
*/

$chain = "<HTML><HEAD><TITLE>Request SQL</TITLE></HEAD><body>\n";
$chain .= "<P align = \"left\"><font size=4> A form on an html page calls a PHP program executing an&nbsp;";
$chain .= "SQL request to an Oracle server. The PHP output is sent to Apache like a CGI program. Apache &nbsp;";
$chain .= "redirects this output to the HTTP client <i>(ex. Internet Explorer)</i> which displays the result\n";
$chain .= "</font><br><br>\n";
$chain .= "<center><b><font size=+3>Result of the SQL request</font></b></center>\n";

/*      2. Analysis of the SQL request      */

$curs1 = OCIparse($connection, "SELECT part_id, part_name FROM part where part_id like '$partid%'");
if(OCIError($curs1))
{
    OCIlogoff($connection);
    $url = "err_base.html";
    header("Location: $url");
    exit;
};

/*      3. Assign Oracle table columns names to PHP variables
*      note 1: The definition of these columns must always be done before an execution;
*      note 2: Oracle always uses capital letters for the columns of a table
*/
```

```

OCI_Define_By_Name($curs1,"PART_ID",$part_id);
OCI_Define_By_Name($curs1,"PART_NAME",$part_name);

/*      4. Execution of the SQL request with an immediate commit to free locks */

OCIExecute($curs1, OCI_COMMIT_ON_SUCCESS);
$chain .= "<b>PART ID PART NAME</b><br>\n";

/*      5. Read each row from the result of the Sql request */

while (OCIfetch($curs1))
    $chain .= "$part_id &nbsp; &nbsp; &nbsp; &nbsp; &nbsp; $part_name<br>\n";

/*      6. Terminate the end of the html format page */

$chain .= "</body></html>\n";
print "<b>Version of this server :</b> " . OCIServerVersion($connection);

/*      7. Free all ressources used by this command and quit */

OCIFreeStatement($curs1);
OCIlogoff($connection);

/*      8. Transmission of the html page ==> Apache ==> client */

echo($chain);
?>

```

Note: **<?php** and **?>** are delimiters of the PHP code which will be executed on the server side.
<SCRIPT> and **</SCRIPT>** are delimiters for JavaScript code client side.

cre_po.html *(dummy transaction)*

```

<html><head>
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="GENERATOR" content="Mozilla/4.7 [fr] (Win98; I) [Netscape]">
  <title>Create purchase order</title>
</head>

<body text="#000000" bgcolor="#B7D0C1" link="#0000EE" vlink="#551A8B" alink="#FF0000"> <center>
  <h1><font color="#CCFFFF"></font>8trd127</h1><i>(Paul Girard, Ph.D.)</i>
  <h1>Lab6<br><br>Creation of a purchase order not completed</h1></center>
</body>
</html>

```

err_base.html

```

<html>
<head><title>Cursor error</title>
</head>
<body bgcolor="#FFFFFF"><p><p>
<h3 align="center">Error in executing a select after a successful OCI connection.</h3>
</body>
</html>

```

connection_error.html

```
<html><head><title>connection_error</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body bgcolor="#FFFFFF"><p>&nbsp;</p><p>&nbsp;</p>
<h3 align="center">Error in trying to connect to the Oracle database dimbd </h3>
</body>
</html>
```

phpinfo.php

```
<html> <head> <title> PHP Info </title> </head>
<body> <? phpinfo(); ?> </body></html>
```

Annex 3

Some PHP Functions

A traditional Chinese version is available is the web site <http://www.php.net/download-docs.php>

OCI_Define_By_Name uses a PHP variable for the define-step during a SELECT.

int oci_definebyname (resource *stmt*, string *Column-Name*, mixed *variable* [, int *type*])

oci_definebyname() returns TRUE on success or FALSE on failure. It copies the values from an Oracle table column (all in capital letters) in a PHP variable. If you define a variable that do not exist in a select command, you will not be warned by an error. If you need to define an abstract data type (*ex. LOB/ROWID/BFILE*), you must allocate memory with the function **oci_new_descriptor()**. Read also **oci_bindbyname()**.

OCI_connect establishes a new connection to the Oracle server. It returns a connection identifier or **FALSE** on error.

int oci_connect (string *username*, string *password* [, string *db*])

oci_logon() returns a connection identifier, necessary for most OCI functions. If the db is not specified, it will use the environment variable ORACLE_SID to determine the connection server. COMMIT and ROLLBACK will apply to all connexions opened in the same php page.

OCI_LogOff closes an Oracle connection

int oci_logoff (resource *connection*)

OCI_Parse prepares an Oracle statement for execution.

int oci_parse (oci_free_desc *conn*, string *query*)

oci_parse() prepares the *query* using *connection* and returns the statement identifier. It returns a statement handler on success, or **FALSE** on error. *query* is any sql statement

OCI_Execute executes a previously parsed *statement* .

int oci_execute (resource *statement* [, int *mode*])

oci_execute() returns TRUE on success or FALSE on failure. Allows you to specify the execution mode (default is OCI_COMMIT_ON_SUCCESS). If you don't want statements to be committed automatically, you should specify OCI_DEFAULT as your *mode* .

OCI_Fetch fetches the next row into result buffer

int oci_fetch (resource *statement*)

oci_fetch() returns TRUE on success or FALSE on failure.

OCICommit commits outstanding statements

`int ocicommit (resource connection)`

ocicommit() commits all outstanding statements for the active transaction on the Oracle *connection* . It returns **TRUE** on success or **FALSE** on failure.

OCIRollback rolls back outstanding transaction

`int ocirollback (resource connection)`

ocirollback() rolls back all outstanding statements for the Oracle *connection* . It returns **TRUE** on success or **FALSE** on failure.

OCIFreeStatement frees all resources associated with statement or cursor

`int ocifreestatement (resource stmt)`

ocifreestatement() frees resources associated with Oracle's cursor or statement, which was received from as a result of [oci_parse\(\)](#) or obtained from Oracle. It returns **TRUE** on success or **FALSE** on failure.

OCIError returns the last error found

`array ocierror ([int stmt/conn])`

ocierror() : If no error is found, **ocierror()** returns **FALSE**. **ocierror()** returns the error as an associative array. In this array, *code* consists the oracle error code and *message* the oracle error string.

header sends a raw HTTP header

`int header (string string)`

header() must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP.

```
<?php
header("Location: http://www.php.net"); /* Redirect the client to another website */
exit();                               /* Make sure the following code will never be executed */
?>
```

Example of PHP functions used with the OCI interface.

```
<?php                                     // Begin a PHP program
print "<HTML><PRE>";
$db = "";
$c1 = oci_connect("scott","tiger",$db);
$c2 = oci_connect ("scott","tiger",$db);
function create_table($conn)
{ $stmt = ociparse($conn,"create table scott.hallo (test varchar2(64))");
  ociexecute($stmt);
echo $conn." created table\n\n";
}
function drop_table($conn)
{ $stmt = ociparse($conn,"drop table scott.hallo");
  ociexecute($stmt);
  echo $conn." dropped table\n\n";
}
function insert_data($conn)
{ $stmt = ociparse($conn,"insert into scott.hallo values('$
conn' || ' ' || to_char(sysdate,'DD-MON-YY HH24:MI:SS'))");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." inserted hallo\n\n";
}
function delete_data($conn)
{ $stmt = ociparse($conn,"delete from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." deleted hallo\n\n";
}
function commit($conn)
{ ocicommit($conn);
  echo $conn." committed\n\n";
}
function rollback($conn)
{ ocirollback($conn);
  echo $conn." rollback\n\n";
}
function select_data($conn)
{ $stmt = ociparse($conn,"select * from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn."---selecting\n\n";
  while (ocifetch($stmt))
  echo $conn." <".ociresult($stmt,"TEST").">\n\n";
  echo $conn."---done\n\n";
}
create_table($c1);
insert_data($c1);    // Row insert with c1
insert_data($c2);    // Row insert with c2
select_data($c1);    // The result of these 2 inserts are displayed
select_data($c2);
rollback($c1);       // Rollback c1
select_data($c1);
```

```
select_data($c2);
insert_data($c2);    // Row insert with c2
commit($c2);         // Validation using c2
select_data($c1);    // Result of c2
delete_data($c1);
select_data($c1);
select_data($c2);
commit($c1);
select_data($c1);
select_data($c2);
drop_table($c1);
print "</PRE></HTML>"; //terminate the html format
?>                    // end of PHP program
```

Annex 4

Overview of of PHP language

1. What is PHP

PHP (*H*ypertext *P*re*P*rocessor") is a widely-used Open Source general-purpose scripting language that is especially suited for Web development and can be embedded into HTML.

Example of a simple PHP script: *hello.php*

```
<html>
<head>
  <title>PHP Test</title>
</head>
<body>
  <?php echo '<p>Hello World</p>'; ?>
</body>
</html>
```

2 Instruction separation

As in C or Perl, PHP requires instructions to be terminated with a semicolon at the end of each statement. There is no need to have a semicolon terminating the last line of a PHP block.

Equivalent PHP blocks

```
<?php
echo 'This is a test';
?>

<?php echo 'This is a test' ?>
```

3 Comments

PHP supports 'C', 'C++' and Unix shell-style (Perl style) comments.

```
<?php
echo 'This is a test'; // This is a one-line c++ style comment
/* This is a multi line comment
   yet another line of comment */
echo 'This is yet another test';
echo 'One Final Test'; # This is a one-line shell-style comment
?>
```

4 Data Types

PHP supports eight primitive types :

- Four scalar types: **boolean** , **integer** , **float** , **string**
- Two compound types: **array** , **object**
- Two special types: **resource** , **NULL**

We will explain briefly the most common types used

4.1 Boolean

To specify a boolean literal, use either the keyword TRUE or FALSE.

```
<?php
$foo = True; // assign the value TRUE to $foo
?>
```

4.2 Integer

Integers can be specified in decimal (*10-based*), hexadecimal (*16-based*) or octal (*8-based*) notation, optionally preceded by a sign (- or +). If you use the octal notation, you must precede the number with a 0 (*zero*), to use hexadecimal notation precede the number with 0x.

```
<?php
$a = -123;    // a negative number
$a = 0123;    // octal number (equivalent to 83 decimal)
$a = 0x1A;    // hexadecimal number (equivalent to 26 decimal)
?>
```

4.3 Float

Floating point numbers can be specified using any of the following syntaxes:

```
<?php
$a = 1.234;
$b = 1.2e3;
$c = 7E-10;
?>
```

4.4 String

A **string** is series of characters. In PHP, a character is the same as a byte, that is, there are exactly 256 different characters possible. It is no problem for a string to become very large.

Examples of strings and concatenation

```
<?php
/* String example */
$str = "This is a string ";
/* Concatenation of 2 strings */
$str = $str . " string extension";
?>
```

Special characters

```
\n  linefeed, LF ou 0x0A en ASCII
\r  carriage return, CR ou 0x0D en ASCII
\t  Tabulation (HT ou 0x09 en ASCII)
\\  Antislash
\$  Character $
\"  double quote
```

4.5 Array

An array in PHP is actually an ordered map. A map is a type that maps *values* to *keys*. This type is optimized in several ways, so you can use it as a real array, or a list (vector), hashtable (which is an implementation of a map), dictionary, collection, stack, queue and probably more.

One dimensional array

```
<?php
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
?>
```

Values may be assigned easily

```
<?php
$a[] = "Bonjour"; // $a[2] == "Bonjour";
$a[] = "Monde";   // $a[3] == "Monde";
?>
```

Multi-dimensional array

```
<?php
$a[1] = $f;           # 1-dimension array
$a["foo"] = $f;
$a[1][0] = $f;        # 2-dimension array
$a["foo"][2] = $f;    # one can mix number and associative indices
$a[3]["bar"] = $f;
$a["foo"][4]["bar"][0] = $f;    4-dimension array
?>
```

4.6 Object

To initialize an object, you use the *new* statement to instantiate the object to a variable.

```
<?php
class foo
{
    function do_foo()
    {
        echo "Doing foo.";
    }
}
$bar = new foo;    // create an object
$bar->do_foo();    // execute do_foo
?>
```

5 Variables

Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is case-sensitive. Variable names follow the same rules as other labels in PHP. A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

```
<?php
$var = 'Bob';
$Var = 'Joe';
echo "$var, $Var";      // outputs "Bob, Joe"
$4site = 'not yet';     // invalid; starts with a number
$_4site = 'not yet';    // valid; starts with an underscore
$täyte = 'mansikka';    // valid; 'ä' is (Extended) ASCII 228.
?>
```

Predefined variables

PHP provides a large number of predefined variables to any script which it runs. Many of these variables, however, cannot be fully documented as they are dependent upon which server is running, the version and setup of the server, and other factors. For example, SERVER_NAME, REQUEST_METHOD, \$_GET, \$_POST

Variables from outside PHP

When a form is submitted to a PHP script, the information from that form is automatically made available to the script. There are many ways to access this information, for example:

Example 1 A simple HTML form

```
<form action="foo.php" method="post">
  Name: <input type="text" name="username" /><br />
  Email: <input type="text" name="email" /><br />
  <input type="submit" name="submit" value="Submit me!" />
</form>
```

6 Constant

A constant is an identifier (*name*) for a simple value. As the name suggests, that value cannot change during the execution of the script.

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
echo Constant; // outputs "Constant" and issues a notice.
?>
```

7 Operators

- assignment: =
- arithmetic : +, -, *, /, %, ++, --
- logical and comparison: <, >, <=, >=, ==, !=, AND, &&, OR, ||, XOR, !
- concatenation : .

8 Control Structures

8.1 Switch structure IF

The *if* construct is one of the most important features of many languages, PHP included. It allows for conditional execution of code fragments. PHP features an *if* structure that is similar to that of C.

```
<?php
if ($a > $b)
{
    echo "a is bigger than b";
} elseif ($a == $b)
{
    echo "a is equal to b";
}
else
{
    echo "a is smaller than b";
}
?>
```

```
<?php
if ($a > $b)
{
    echo "a is bigger than b";
} else
{
    echo "a is NOT bigger than b";
}
?>
```

8.2 FOR

FOR loops are the most complex loops in PHP. They behave like their C counterparts. The syntax of a *for* loop is:

```
for (expr1; expr2; expr3) statement
```

The first expression (*expr1*) is evaluated (*executed*) once unconditionally at the beginning of the loop. In the beginning of each iteration, *expr2* is evaluated. If it evaluates to **TRUE**, the loop continues and the nested statement(s) are executed. If it evaluates to **FALSE**, the execution of the loop ends. At the end of each iteration, *expr3* is evaluated (*executed*).

```
<?php
for (expr1; expr2; expr3)
statement
?>
```

example 1

```
for ($i = 1; $i <= 10; $i++) {echo $i;}
```

example 2 (*expr2 is always true, break is used*)

```
for ($i = 1; ; $i++)
{
    if ($i > 10)
    {
        break;
    }
    echo $i;
}
```

example 3 (*infinite loop, break is used*)

```
$i = 1;
for (; ; )
{
    if ($i > 10)
    {
        break;
    }
    echo $i;
    $i++;
}
```

example 4

```
for ($i = 1, $j = 0; $i <= 10; $j += $i, print $j, $i++);
```

8.3 WHILE

WHILE loops are the simplest type of loop in PHP. They behave just like their C counterparts. The meaning of a *while* statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the *while* expression evaluates to **TRUE**. The basic forms of a *while* statement is:

```
<?php
while      (expression)
statement
?>
```

OR

```
while (expr):
    statement
    ...
endwhile;
```

example 1

```
$i = 1;
while ($i <= 10)
{
    print $i++;
}
```

example 2

```
$i = 1;
while ($i <= 10):
    print $i;
    $i++;
endwhile;
```

8.4 DO WHILE

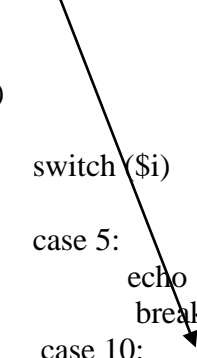
DO WHILE loops are very similar to *while* loops, except the truth expression is checked at the end of each iteration instead of in the beginning. The main difference from regular *while* loops is that the first iteration of a *do-while* loop is guaranteed to run (the truth expression is only checked at the end of the iteration), whereas it's may not necessarily run with a regular *while* loop (the truth expression is checked at the beginning of each iteration, if it evaluates to **FALSE** right from the beginning, the loop execution would end immediately).

There is just one syntax for *do-while* loops:

```
<?php
$i = 0;
do {echo $i;} while ($i > 0);
?>
```

8.5 Break

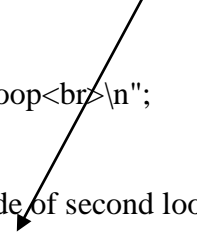
Break ends execution of the current *for*, *while*, *do-while* or *switch* structure. *break* accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.



```
<?php
$i = 0;
while (++$i)
{
    switch ($i)
    {
        case 5:
            echo "At 5<br />\n";
            break 1;      /* Exit only the switch. */
        case 10:
            echo "At 10; quitting<br />\n";
            break 2;      /* Exit the switch and the while. */
        default:
            break;
    }
}
?>
```

8.6 Continue

continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and then the beginning of the next iteration. *continue* accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.



```
<?php
while (1) {
    echo " Middle of loop<br>\n";
    while (1)
    {
        echo " Inside of second loop<br>\n";
        continue 2;
    }
    echo "This is never printed<br>\n";
}
?>
```

8.7 Switch

The **switch** statement is similar to a series of IF statements on the same expression. In many occasions, you may want to compare the same variable (*or expression*) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the *switch* statement is for. Note that unlike some other languages, the **continue** statement applies to *switch* and acts similar to *break*. If you have a switch inside a loop and wish to continue to the next iteration of the outer loop, use *continue 2*.

Example1

```
<?php
if ($i == 0) {
    echo "i equals 0";
} elseif ($i == 1) {
    echo "i equals 1";
} elseif ($i == 2) {
    echo "i equals 2";
}
switch ($i) {
    case 0:
        echo "i equals 0";
        break;
    case 1:
        echo "i equals 1";
        break;
    case 2:
        echo "i equals 2";
        break;
}
?>
```

Example 2

```
<?php
switch ($i) {
    case "apple":
        echo "i is apple";
        break;
    case "bar":
        echo "i is bar";
        break;
    case "cake":
        echo "i is cake";
        break;
}
?>
```