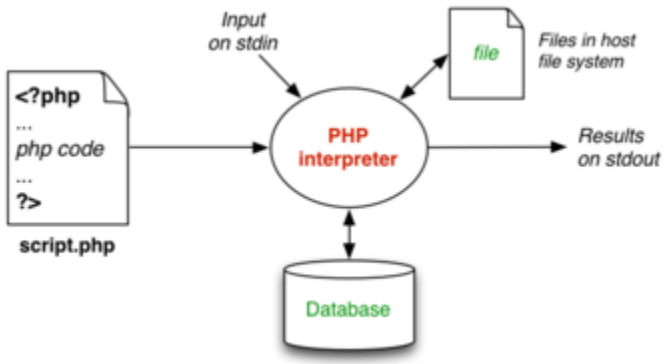
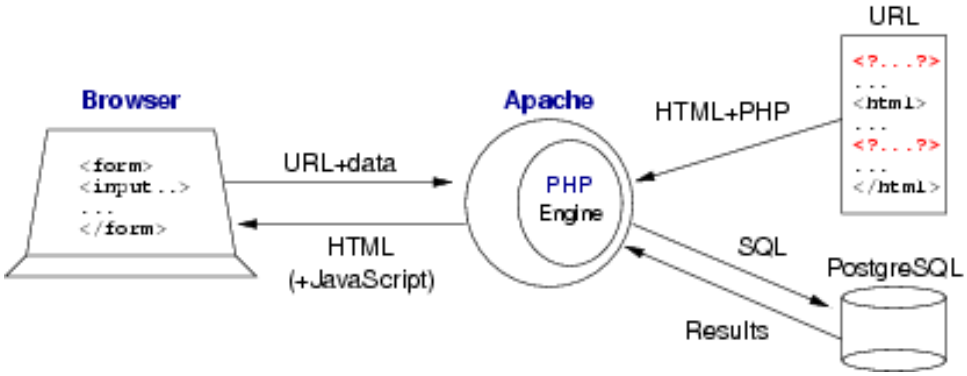
PROGRAMMING WITH DATABASE

PHP



- PHP web scripts are a mixture of HTML and PHP code
  - Stored on web server (Apache) under its DocumentRoot
  - Invoked vis URL (http://server/a/b/script.php)
  - Parameters passed either via **GET** or **PUT**
  - Executed in an engine (Zend) inside the web server
  - With environment/privileges of web server process
  - Having access to cookies (on client) and DBMSs (on server)



- You have an Apache web processor, and have a PHP Engine, which will be run as a module of a web server (running together as the same process). Then have HTML mixed with PHP code (with <? … ?>). And the PHP Engine will have the access to the PostgreSQL, so script can send a SQL to the database and get a result back to the script.

FROM PHP TO HTML

- How the PHP engine treats a script:
  - Scan the script from the top to bottom; interpolate **required** files
    - Similar to the include files in C.
  - Any text not enclosed in <? … ?> is copied to output
    - If not in <? ... ?>, will not be process in the Engine but send to the browser directly.
  - Any PHP expression enclosed in <?=Expr?> is evaluated, and its string representation is copied to output
  - any PHP code enclosed in <?Statements;?> is executed, and any output it produces is sent to output
  - first output is preceded by **HTTP header: Content-type: text/html**
    - after the process of scripts, the html body content will be send to the browser through Apache server. However, before it send the body content, need to send a HTTP header first.
  - **header()** function can be used to produce alternative HTTP header
    (if any output has already been sent when header() called, produces error)
    - make sure call this function BEFORE send the html body to the browser.

```
<html>
<? require("myDefinitions.php");
    $pageName = $_POST["name"]; $max = $_POST["max"];
?>
<body bgcolor='purple'>
<h1>This is <?=$pageName?></h1>
<? if ($max <= 0) { ?>
   <b>There are no numbers to display</b>
<? }
else {
    for ($i = 0; $i <= $max; $i++)
        echo "$i<br>\n";
}
?>
</body>
</html>
```

- first of all, *<html> + </body> + </html >*tells that it's a HTML file
- with <? … ?> are all PHP scripts
- ***require("myDefinitions.php");***
  - it requires another php file in the same folder, same as Include, so we interpolate that file into this document.
- ***<body bgcolor = 'purple'>***
  - The background color of this html page is purple. This is not the PHP
- ***<h1> This is <?=$pageName?></h1>***
  - <h1> + </h1>, this is the header 1 format in html
  - <?=$pageName?>, a variable in the php script, find the value from the previous command
- ***<? If ($max <= 0) { ?> … <?}***
  - If max <= 0, you'll do the following otherwise you wouldn't
  - …

- Nowadays, most PHP usage in web Application frameworks*
  - Using MVC design pattern
    - (MVC stands for model, view and controller)
    - So kind of a software design pattern: it's a repeatable solution to software design problem that usually happen (component/architecture).
    - Php usually in those kind of framework uses this type of pattern, MVC, which means you can easily customize or extend these php application for different application domain.
  - Providing overarching control of web app (C=control)
  - Template-based HTML rendering (V=view)
  - Providing DBMS-independent DB access (M=model)

PHP LANGUAGE

- The PHP language has the following characteristics:
  - C-like syntax (with some Perl flavor)
  - "loose" attitude to types (determined by context)
  - Very easy to manipulate strings
  - Associative arrays (cf. Perl's hashes)
  - Extensive libraries of functions (2000 pages manual)
    - Can use function calls
  - Supports object-orientation (cf. Perl)
  - Comments introduced via # or //

- When PHP programs are executed in web server …

- The HTTP request supplies the parameters. (or they're available in $argv[] if run from the command-line)
- CGI params available in array $_GET, $_POST, $_REQUEST.
- E.g.

```
http:// server/user/list.php?name=John&age=21
```

  - $_GET is for us to retrieve some information from the web resources.
  - $_POST is you can use API call to post a new message, to create/post something to the resources

in the script, the parameters would be accessed as:

```
print "Name is $_REQUEST[name]\n";
print "Age is ".$_REQUEST['age']."\n";
print "Name: $_GET[name] Age: $_GET[age]\n";
```

  - Inside of the PHP script:
    - If you access to a variable called REQUEST(super variable), call [name], which is the key for, name=John, this parameter you can get the name John from the URL, same as age.

VARIABLES

```
$foo = 3;           # $foo is an int, value 3

$foo = "8";         # $foo is now a string, value "8"

$foo = $foo + 2;    # $foo is now an int, value 10

$foo = "$foo green bottles";
                    # $foo is now "10 green bottles"

$foo = 3.0 * $foo; # $foo is now double, value 30.0

$foo = (int)$foo;   # $foo is now an int, value 30
```

- No variable declarations are required.
- Variables are created by assigning a value to them.
- **All variable names are preceded by $** (note: $i, $i++, $++i)
- The type of a variable is that of the **last assigned value**.
  - If you first assign $i to string, then $i to an integer, the type of $i would be integer, see the example above!
- Check/set variable *type* via **gettype/settype** functions.
- Convert variable *value* via casting (e.g. (int), (string), ...)
- Default value of unassigned variable is null (distinguished constant)
  (if unset variable used, get 0 or "" or false, depending on context, and error in log)
- The life time of all the variables are WITHIN the current script.
- Variables defined outside of any function:
  - Have global scope (over whole to script)
  - But are not accessible with functions UNLESS "requested": *Function f() { **global** $max_num, $colour; ... }*
- "super-global" arrays (e.g. **$_GET, $_PUT, $_SERVER, $_COOKIE,** ...):
  - Contain "environment" values (CGI params, server ENV, request data)
  - Are accessible from anywhere in the script

CONSTANTS

- Constants are defined using the **define()** function:

```
define("CONSTANT", "Hello world.");
define("MaxLevel", 6);
echo CONSTANT; // outputs "Hello world."
echo Constant; // outputs "Constant" and gives error
if ($i > MaxLevel) { echo "Yes"; }
```

- May only evaluate to scalar type values (e.g. int, float, string)
  - It cannot be array
- Have case-sensitive names, written without dollar sign $
  - Uppercase/ lowercase are different, see the example above
- Are always available globally (like super-variables)
- May not be redefined or undefined once they have been set

TYPES

- Boolean, with values **true** and **false** (case-sensitive)
  - Use C-like interpretation for false (i.e. 0, "", …)
  - All non-zero values are treated as true (beware: this includes negative error status values)
- Integer, e.g. 0, 1, -999, … (standard 32-bit format)
- Float, e,g, 3.14, 2.0e6, … (IEEE floating point format)
- String …

STRINGS

- Strings: sequences of characters, similar to perl
  - **Double-quotes** strings (" … ") permit interpolation!!

```
$x = 5;  print "Value of x is $x\n";
// prints "Value of x is 5"
```

    - Must escape embedded via \, escape sequences work, variable interpolation works
  - **Single-quoted** strings DO NOT do interpolation!!

```
$x = 5;  print 'Value of x is $x\n';
// prints "Value of x is $x"
```

    - No variable interpolation, no escape sequences work (including no \)
  - **Non-quotes** strings (abc) (only work in some contexts)
    - Non-quotes strings look like C/JAVE variables; PHP variables look like $abc
    - Non-quotes strings are actually an ERROR, normally used for constants. In some contexts they produce a value which is the same as their name
- Strings (cont) "heredoc" strings available for large multi-line strings

```
print <<<XYZ
This is a "here" document. It can contain
many lines of text, with interpolation.
Such as the value of x is $x
With any old "quotes' the we ``like''
XYZ;

$str = <<<aLongString
This is my "long" string.
Ok, it's not really so long
aLongString;
```

It (multiline string will be treat as one string value) start with the <<< (identify) and end with the same (identify);

- When variables are used inside a "…" string or heredoc'
  - Their value is interpolated into string
  - After being converted to a suitable string representation

```
$a = 1;  $b = 3.5;  $c = "Hello";
$str = "a:$a,  b:$b,  c:$c";
// now $str == "a:1, b:3.5, c:Hello"
```

- Note that interpolation does occur in "This is '$it'"

I.e. <? $it = 5; print "This is '$it'"; ?> displays This is '5'

- This is important in producing HTML in PHP since attribute values for HTML tags should be quoted.
- Example: we want to create a text input box to collect a new value for parameter name, and display its current value:
  print "<input type='text' name='qty' value='$_GET[qty]'>\n";
- Note: If the qty parameter is not set, then the $_GET["qty"] will have no value, and the text box will be empty.
- Other operations on strings:
  - Dot for string concatenation

```
$x = 127;
print "Result is ".sqrt($x)."\n";
```

  - **Trim()** removes whitespace from the left and right end of the string
- **Preg_split()** partitions string into ARRAY via PERL REGEXP / split string by a regular expression

```
// $s == "  ab  cde fg"
$a = preg_split('/\s+/',$s);
// $a[0]=="" && $a[1]=="ab"
// && $a[2]=="cde" $a[3]=="fg"
```

  - Suppose we have a string, they're space separated
    - In the beginning there are some space
    - If you don't want the first element to be empty/space, use trim() function call
    - / means start or end of pattern string
    - \s means space
    - + means one or many
- **Join()** assembles strings from an array

```
// $a[0]=="" && $a[1]=="ab"
// && $a[2]=="cde" $a[3]=="fg"
$s = join(":",$a);
// $s == ":ab:cde:fg";
```

  - Join all the array element back together as a string.

ARRAY

- PHP arrays = sequence of values accessible via index.
- Indexes can be values of any scalar type including strings.
- This provides both scalar and associative arrays (hash tables)

```
$a[0] = "abc";  $a[1] = 'def';  $a[2] = ghi;

$b['abc'] = 0;  $b[def] = 1;    $b["ghi"] = 2;
```

  - An array is a special variable, which can hole more than one value at a time
  - What if you want to loop through the cards and find the specific one? And what if you had not 3 cars but 300?
  - The you can CREATE an array.
  - Array can hold many values under a single name, and you can access the values by referring to index number.
  - **Array()**
    - INDEXED ARRAY – arrays with a numeric index
    - ASSOCIATIVE ARRAY – arrays with named KEYS
    - MULTIDIMENSIONAL ARRAYS – arrays containing one or more arrays
  - **Count()** to count array length / how many variables are there in the array

```php
<!DOCTYPE html>
<html>
<body>

<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " . $cars[2] . ".";
?>

</body>
</html>
```

I like Volvo, BMW and Toyota.

```php
<!DOCTYPE html>
<html>
<body>

<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>

</body>
</html>
```

Peter is 35 years old.

Multiple values can be extracted from arrays via **list()**:

```php
$a = array(5, 4, 3, 2, 1);
list($x,$y,$z) = $a;
# $x==5, $y==4, $z==3
```

Multi-dimensional arrays work ok (array elements can be any type)

```php
$fruits = array ( "fruits"  => array ( "a" => "orange"
                                     , "b" => "banana"
                                     , "c" => "apple"
                                     )
                , "numbers" => array ( 1,2,3,4,5,6 )
                , "holes"   => array (      "first"
                                     , 5 => "second"
                                     ,      "third"
                                     )
                );
```

- Multi-dimensional array -> assign new array element inside of array

```php
<?php
for ($i = 0; $i < count($word); $i++)
// $word = array("a"," b","c")
// count() count the element of array
    print "word[$i] = $word[$i]\n";
// output:
// word[0]=a, word[1]=b, word[2]=c
?>

another way to do it
<?php
foreach ($words as $w)
// for each element in the array will be view as $w
    print "word = $w\n"
?>

<?php
// reset() = set the array's internal pointer
// back to the first element in the array

// key() = current internal pointer position
// which is similar to the index for c
for (reset($marks); $name = key($marks)); next($marks))
    print "mark for $name = $marks[$name]\n"
?>
```

Other PHP Types

- PHP has standard notion of **class**: data values + method

```
// creating an object of class foo
$x = new foo;  $x->method(1,'a');
```

- Resource: special type for references to external resources
    - E.g. database connections/cursors, file handles, …
- NULLL: a distinguished value NULLL

VARIABLE CHECKING

- Isset($v) = $v has non-NULL value
    - Can check whether an array has a value for a given index
- Is_null($v) = $v has the value NULL
- Empty($v) = $v has value NULL or 0 or “” or arrat()
- Unset($v) = effectively removes variable $v

VARIABLE VARIABLES

- PHP provides a dynamically create variable names

```
for ($i = 0; $i < $MAX; $i++) {
   $varname = "myVar$i";
   $value   = ${$varname};
   print "Value of $varname = $value\n";
}
```

    - Access variables called myvar1, myvar2, mybar3, …

CONTROL STRUCTURE

```
{ Statement1; Statement2; ... }

if (Expression1) Statement1
[elseif (Expression2) Statement2 ...]
[else Statementn]

switch (Expression1) {
case Value1: Statement1; break; ...
[case Value2: Statement2; break; ...]
}

while (Expression) Statement
for (Init; Test; Next) Statement
foreach (ArrayVar as [KeyVar =] ValVar) Statement
```

DEBUGGING

- Print_r($v) displays representation of $v’s value
- Var_dump($v) displays more info on $v’s value
- Error_reporting(Level) controls how much error display
- @func() executes func() and suppresses error reporting

PHP and PostgreSQL

- PHP has a library of functions for PostgreSQL interaction
- Follow typical PL/DBMS interaction pattern:
  - Send SQL query, retrieve results one-at-a-time
  - Access to database and result-set metadata
- Obvious problem: code written using it is non-portable
- There is also a generic DB-access library called PDO

```
- pg_connect() = connect to the database
- pg_query() = send SQL statement for processing
- pg_fetch_array() = retrieve the next result tuple
- pg_num_rows() = count # rows in result
- pg_affected_rows() = count # rows changed
```

PG_CONNECT ()

- **Resource pg_connect (string** ConnParams**)**
  (before we can access data in MySQL database, we need to be able to connect to the server)
  - Attempts to connect to datavase specified in ConnParams
  - Precise format of ConnParams depends on configuration:

```
$db = pg_connect("dbname = mydb");
# or
$cp = "dbname=hisdb user=fred password=abc";
$db = pg_connect($cp);
```

  - Returns a **resource**, which is used for DB interactions
  - If any problems, return 0 (illegal connection)
    - Possible problems: invalid password, unknown DB, …
  - The **pg_last_error()** function gives details of any error.

PG_QUERY ()

- **Resource pg_query (resource** db**, string** Stmt**)**
  (execute a query on the specified connection)
  (you can save the database resource and the query string in php variable, $variable.)
  - Send the SQL statement STMT to the database db
  - STMT can be either a query or insert/delete/update
  - Returns a resource, which is either
    - A cursor on the result set for query
    - Nothing useful for insert/delete/update
  - If any problems return 0 (illegal cursor)
    - Possible problems: invalid db, syntax error in STMT
  - Subsequent attempts to use illegal cursor give PHP error

```
example:
$unidb = pg_connect("dbname = UniDB");
$query = "select name from Staff where dept=2";
$result = pg_query($unidb, $query);
if (!$result)
        print "something wrong with query!\n"
else
        // proess the result set...
// to find ouu exactly what was wrong with query
if (!$result)
        print pg_last_error();
```

PG_NUM_ROWS ()

- **Int pg_num_rows (resource** Result**)**
  - Returns the number of tuples in a **pg_query** query result
  - Zero if the **pg_query** statement was an UPDATE

```
example:
$query = "select * from emplyees where department='sales'";
$result = pg_query($db, $query);
if (!$result)
        print pg_last_error();
else if (pg_num_rows($result) > 20)
        print"this is a very big department\n";
```

PG_AFFECTED_ROWS ()

- **Int pg_affected_rows (resource** Result**)**
  If your query itself is update/delete/insert then this function will return the number of rows that has been affected
  - **Returns # modified tuples in a pg_query update**
  - Zero, if the pg_query statement was a query

```
example:
$query = "delete from Enrolments where course='COMP3311'";
$result = pg_query($db, $query);
if (!result)
        print pg_last_error();
else {
        $nstudes = pg_affected_rows($result);
        print "dropped $nstudes from COMP3311\n";
}
```

PG_FETCH_ROW ()

- **Array pg_fetch_rows (resource** Res**, int** which**)**
  Fetches one row of daya from the result associated with the specified result resource.
  - Fetches the ith tuple in a query result set
  - If no which argument, fetches next tuple
  - Returns an **array** value that can be treated as a result row
  - Field are accessed by position; based on query select list
  - If no more element left returns 0

```
example:
$query = "select id, name from Staff";
if ($result = pg_query($db, $query)) {
        $n = pg_num_rows($result);
        for ($i=0;$i<$n;$i++) {
                $item = pg_fetch_row($result,$i);
                print "Name=$item[1], StaffID=$item[0]\n";
        }
}
```

PG_FETCH_ARRAY ()

- **Array pg_fetch_array (resource** Res**, int** which**)**
  - Fetches the ith element (tuple) in a query result set
  - Returns an **array** value that can be treated as a result row
  - Array is indexed by field names as well as position
  - If no more elements left, returns 0
  - If no which argument, fetches next tuple

```
example:
$query = "select id, name from Staff";
if (!($result = pg_query($db, $query)))
        print "Error: ".pg_last_error();
else {
        $n = pg_num_rows($result);
        for ($i=0;$i<$n;$i++) {
                $item = pg_fetch_array($result,$i);
                $nm = $item["name"];$id = $item["id"];
                print "Nmae=$nm, StaffId=$id\n";
        }
}
```

COMP3311 DATAVASE LIBRARY

- accessDB (dbname) = establish connection to DB
- dbQuery(db,sql) = send SQL statement for execution
- dbNext (res) = fetch next tuple from the result set
- dbOneTuple (db,sql) = run SQL to get single tuple
- dbOneValue (db,sql) = run SQL to get a single value
- dbUpdate(db,sql) = send SQL insert/delete/update
- mkSQL (fmt,v1,v2, …) = build an SQL statement string

```
$db = dbConnect("dbname = mydb");
...
$min = ...;
$query = "select a,b,c from R where c >= %d";
// the php variable $min will be as the %d
$result = dbQuery($db, mkSQL($query,$min));
while ($tuple = dbNext($r)) {
// call the result one by one by dbNext instead of pg_fetch_array
        list($a,$b,$c) = $tuple;
        $tmp = $a - $b - $c;
        # or
        $tmp = $tuple["a"] - $tuple["b"] - $tuple["c"];
}
```

mkSQL

- string mkSQL (string querytemplate, any v1, any v2, …)
    - queries are often constructed by interpolating variables
    - ensures that values are appropriately quoted/escaped
    - uses printf-like mechanism for specifying interpolated values

```
example: MKSQL()
$name = "O'Brien";
$tmpl = "select * from Employees".
        "where name = %s and salary > %d";
$qry = mkSQL($tmpl, $name, 50000);
// which produces the query string
// select *from Employees
// where name = 'O'Brien' and salary > 50000
```

```
example:
$db = accessDB("mymyunsw");
$query = <<_SQL_
select s.sid, p.name
from students s, prople p, courses c, subject sb, courseEnrol e, terms t
where s.id = p.id and e.student = s.id and e.course = c.id
      and c.subject = su.id and su.code = %s and c.term = t.id
      and t.year = %d and t.sess = %s
order by s.sid
_SQL_;
// $subj, $year, $sess for the three para,s, %s, %d, %s above
$sql = mkSQL($query, $subj, $year, $sess);
// you can then execute the query, and get the result out
$result = dbQuery($db, $sql);
// each tuple t from the db
while ($t = dbNext($result)) echo "$t[sid] $t[name]\n";
```