



Berner Fachhochschule  
Haute école spécialisée bernoise  
Bern University of Applied Sciences

# 4) PHP and MySQL

*Emmanuel Benoist*  
Spring Term 2017

# PHP and MySQL

- Introduction
- Basics of MySQL
  - Create a Table
  - See the content of a DB
  - Tables: Change rows and Insert data
  - Select Information
- PhpMyAdmin
- PHP and MySQL together
  - PDO
- Conclusion

# Introduction

# PHP and the Data Bases

## MySQL syntax

- ▶ Create a new Data Base
- ▶ Set the rights for a DB
- ▶ Create tables
- ▶ Fill information into tables
- ▶ Select information (can sometime be very tricky)
- ▶ Update information

## PHP MyAdmin

- ▶ A PHP program for managing MySQL BD's
- ▶ Graphical and practical
- ▶ Do not require to log on the machine (only web access)

## PHP Library for MySQL

- ▶ The old function-oriented library
- ▶ The new object-oriented library

# MySQL a Data Base for the Web

## Open-Source DB

- ▶ Free
- ▶ Present in any Linux distribution
- ▶ Available for fast any OS (Windows, Free-BSD, Mac-OS X,...)

## The perfect solution for web

- ▶ LAMP architecture (Linux, Apache, MySQL, PHP) is one of the web standards
- ▶ An application (phpMyAdmin) for managing the DB without shell access
- ▶ Perfect integration within PHP.

# Basics of MySQL

# Basics of MySQL commands

## **Creation functions (often done within PHP-MyAdmin)**

- ▶ Create a new table
- ▶ Set the properties of fields (auto-increment, default value,...)

## **Routine functions (will be used in your programs)**

- ▶ Insert an element in a table
- ▶ Select elements out of a table
- ▶ Select elements out of many tables
- ▶ Change the content of a record
- ▶ Delete some records

# Create a Table



# Creation of a table

## Syntax

- ▶ `CREATE TABLE` *table name* (*definition of the fields*)

## Create a small table

```
CREATE TABLE 'category' (  
  'name' VARCHAR( 100 ) NOT NULL ,  
  'categoryID' TINYINT NOT NULL AUTO_INCREMENT ,  
  PRIMARY KEY ( 'categoryID' )  
);
```

- ▶ Create a table with two fields
- ▶ a string which length can not exceed 100
- ▶ A primary key that is a counter

# Create a new table

nur INT und BIGINT benutzer - alles andere ist heute  
nicht mehr wichtig!

## The table can have fields of the following types:

- ▶ TINYINT SMALLINT MEDIUMINT INT BIGINT that are integers (more or less long)
- ▶ VARCHAR for short strings (smaller than 256 chars)
- ▶ TEXT for texts with a fixed length (max 64 kB)
- ▶ DATE date in format YYYY-MM-DD
- ▶ TIMESTAMP contains a unix timestamp
- ▶ TIME format hh:mm:ss
- ▶ DECIMAL number with a point.
- ▶ FLOAT nicht benutzen
- ▶ DOUBLE real numbers
- ▶ BLOB Any Binary data (image, sound, long text, ...)
- ▶ ...

# Create a new table (Cont.)

## **Other attributes or features**

- ▶ `NULL` or `NOT NULL`
- ▶ `AUTO_INCREMENT` for counters

## **The table has also properties**

- ▶ `PRIMARY KEY`
- ▶ `COMMENT` description of the table

# Create other tables

## The article and vat tables

```
CREATE TABLE 'article' (  
  'articleID' INT NOT NULL AUTO_INCREMENT ,  
  'name' VARCHAR( 100 ) NOT NULL ,  
  'vatID' TINYINT NOT NULL ,  
  'categoryID' INT NOT NULL ,  
  'Price' DECIMAL NOT NULL ,  
  PRIMARY KEY ( 'articleID' )  
);
```

```
CREATE TABLE 'vat' (  
  'vatID' TINYINT NOT NULL AUTO_INCREMENT ,  
  'rate' DECIMAL NOT NULL ,  
  PRIMARY KEY ( 'vatID' )  
) COMMENT = 'The table containing VAT rates';
```

See the content of a DB

# See the content of a data base

## See all tables

```
mysql> show tables;
+-----+
| Tables_in_example |
+-----+
| article            |
| category           |
| vat                |
+-----+
3 rows in set (0.00 sec)
```

# See the content of a data base (Cont.)

## See all columns of a table

```
mysql> show columns from vat;
```

| Field | Type          | Null | Key | Default | Extra          |
|-------|---------------|------|-----|---------|----------------|
| vatID | tinyint(4)    |      | PRI | NULL    | auto_increment |
| rate  | decimal(10,2) |      |     | 0.00    |                |

2 rows in set (0.00 sec)

# Tables: Change rows and Insert



# Change a Table - ALTER

## Remove columns

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

## Add a new column

```
ALTER TABLE 'article' ADD 'description' BLOB  
NOT NULL ;
```

## Change an existing column

```
ALTER TABLE 'article' CHANGE 'Price' 'price'  
DECIMAL( 10, 2 ) DEFAULT '0' NOT NULL;
```

# Fill data into a table - INSERT

## Syntax

- ▶ `INSERT INTO tablename [(list of fields)] VALUES (list of values);`
- ▶ all not null fields must be set, other can be just two commas.

## Insert a row in a table

```
INSERT INTO 'article' ( 'articleID' , 'name' , 'vatID' ,  
    'categoryID' , 'price' , 'description' )  
VALUES ( '', 'Pencil', '0', '0', '1.50', '' );
```

## Other possibility

```
INSERT INTO article values  
    ( '', 'Mercedes Class E', '0', '0', '100000',  
    'The same Mercedes Lady Diana has used'  
    );
```

# Change the content of one or many rows

UPDATE **a table**

```
UPDATE 'article' SET 'description' =  
    'A very nice black pencil with white stripes'  
WHERE 'articleID' = '1' LIMIT 1 ;
```

# Select Information

# Select information

## Syntax

- ▶ `SELECT Field list FROM list of tables [WHERE conditions] [LIMIT limits]`
- ▶ Field list can also be a joker (\*)
- ▶ Conditions can be combined with boolean connectors (AND, OR, NOT)
- ▶ If we only want to see a part of a list, we can limit it.

## Select all the rows and columns of a table

```
mysql> select * from vat;
```

|   |       |   |       |   |
|---|-------|---|-------|---|
| + | ----- | + | ----- | + |
|   | vatID |   | rate  |   |
| + | ----- | + | ----- | + |
|   | 1     |   | 7.00  |   |
|   | 2     |   | 7.65  |   |
| + | ----- | + | ----- | + |

# Select information(Cont.)

## Select only some columns

```
mysql> select name, price from article;
```

| name             | price     |
|------------------|-----------|
| Pencil           | 1.70      |
| Mercedes Class E | 100000.00 |

```
2 rows in set (0.00 sec)
```

# Select data

## Select only some rows

```
mysql> select name, price from article  
      -> where articleID=1;
```

```
+-----+-----+
```

```
| name   | price |
```

```
+-----+-----+
```

```
| Pencil | 1.70 |
```

```
+-----+-----+
```

```
1 row in set (0.01 sec)
```

# Merge data from different tables

## Merge two tables

- ▶ Fields must know from which table they come (the same field can be in the two tables).
- ▶ We can rename a requested field with the AS keyword.

```
mysql> select article.name, vat.rate, article.price  
-> from article, vat where article.vatID= vat.vatID;
```

| name             | rate | price     |
|------------------|------|-----------|
| Pencil           | 7.00 | 1.70      |
| Mercedes Class E | 7.00 | 100000.00 |



# Merge ...(Cont.)

## Merge and compute

```
mysql> select article.name, vat.rate, article.price,  
-> article.price*(1+vat.rate/100) as priceWithVAT  
-> from article, vat where article.vatID= vat.vatID;
```

| name             | rate | price     | priceWithVAT |
|------------------|------|-----------|--------------|
| Pencil           | 7.00 | 1.70      | 1.8190       |
| Mercedes Class E | 7.00 | 100000.00 | 107000.0000  |

```
2 rows in set (0.00 sec)
```

# Join

**INNER JOIN** If there is no match, the row is not shown

```
select article.name, vat.rate, article.price  
      from article inner join vat  
      on article.vatID= vat.vatID;
```

**LEFT JOIN** If there is no match, the second table is replaced by an empty record.

```
select article.name from article left join vat  
      on article.vatID= vat.vatID  
      where vat.rate is null;
```

(gives the list of articles with undefined VAT)

# More on SELECT

**Result of a select can be put into a temporary table**

```
create temporary table valueVAT
  (select vat.rate, article.name
   from vat,article
   where vat.vatID=article.vatID
  )
;
```

**You can access to the content and then delete this table**

```
select * from valueVAT;

drop table IF EXISTS valueVAT;
```

# Select and more options

## Order result (DESC or ASC)

```
select name, price from article order by price desc;
```

## Group rows

```
mysql> select vatID, count(vatID)  
      > from article GROUP BY vatID;
```

|               |              |
|---------------|--------------|
| +-----+-----+ |              |
| vatID         | count(vatID) |
| +-----+-----+ |              |
| 1             | 2            |
| +-----+-----+ |              |

```
1 row in set (0.00 sec)
```

**SELECT can have a lot of functions and combine all of them**

# Delete fields

**Delete the content of a table respectively to a where clause**

```
delete from article where articleID=3;
```

# PhpMyAdmin

# Administrate MySQL with a Web interface

## **phpMyAdmin**

- ▶ A PHP program for managing MySQL Data Bases
- ▶ Free available at <http://www.phpmyadmin.net>
- ▶ Included in most of the Linux distrib
- ▶ Internationalization

## **Management made easy**

Generate and displays the SQL query corresponding.

- ▶ Create a new Data Base
- ▶ Create a new Table
- ▶ Add or remove a column in a table

# phpMyAdmin

## Management of data

- ▶ Select data made easy
- ▶ Update using a visual interface (does not work for BLOBs)
- ▶ A lot of selection boxes containing all the possible values

## Import / Export of data

- ▶ Can create SQL Dump
- ▶ Can export in a lot of formats: SQL, CSV, LaTeX, CSV for excel, XML
- ▶ With a lot of properties (zipped, gzipped, with delete tables or not, ...)



# Conclusion - MySQL/phpMyAdmin

## **Not as much powerful as other DB's**

- ▶ MySQL does not implement all of SQL
- ▶ It is enough to handle a small web site
- ▶ Very useful and easy to install, configure and manage

## **PHP supports all other DB's**

- ▶ Oracle
- ▶ ODBC (MS-SQL server, Access)
- ▶ Postgress
- ▶ DBase
- ▶ ...

# PHP and MySQL together

# PDO

# PHP Data Objects - PDO

- ▶ **PDO is a generic connector for any database**
  - ▶ Code is (almost) platform independant
  - ▶ Support is provided for all major Data Bases servers (MySQL, Oracle, MS SQL Server, Sybase, Informiy, ODBC, PostgreSQL, SQLITE)
- ▶ **PDO is an object oriented interface**
  - ▶ Creates a connection object
  - ▶ Creates Statements and Results that are objects

# Create a connection

- ▶ **The first step is to create a connection to the DB server**
  - ▶ Needs an URL for connecting:
  - ▶ `protocol:host=<hostname>;dbname=<dbname>`
  - ▶ Arguments are username and password
- ▶ **Close the connection by setting the handler to null**

```
<?php
$hostname = 'localhost';
$username = 'username';
$password = 'password';
try {
    $dbh = new PDO("mysql:host=$hostname;dbname=example",
                  $username, $password);

    echo 'Connected to database';
    /** close the database connection */
    $dbh = null;
}
catch(PDOException $e) { echo $e->getMessage(); }
?>
```

# Execute a SQL query

- ▶ **Execute a SQL Query** : `$dbh->exec($sql);`
- ▶ **Value returned** = number of affected rows
- ▶ **Should be used if no result set is returned (INSERT / UPDATE)**

```
try {  
    $dbh = new PDO("mysql:host=$hostname;dbname=animals",  
                   $username, $password);  
  
    /** INSERT data **/  
    $sql = "INSERT INTO article (name, price) VALUES ('Journal', '1.9')\  
    →";  
    $count = $dbh->exec($sql);  
    /** echo the number of affected rows **/  
    echo $count;  
    /** close the database connection **/  
    $dbh = null;  
}  
catch(PDOException $e){  
    echo $e->getMessage();  
}
```

# Update records

- ▶ `PDO::exec()` is used each time no selection is needed

```
/** INSERT data **/  
$query=" UPDATE_animals SET_animal_name='bruce'";  
$query .= " WHERE_animal_name='troy';"  
$count = $dbh->exec($query);  
/** echo the number of affected rows **/  
echo $count;
```

# Select queries

- ▶ **SELECT** returns a result set
- ▶ **Method is** `PDO::query()`
- ▶ **Returned statement can be visited like an array**  
**(implements the SPL traversible iterator)**

```
/** The SQL SELECT statement */  
$sql = "SELECT * FROM animals";  
foreach ($dbh->query($sql) as $row)  
{  
    print $row['animal_type'] . ' - ' .  
        $row['animal_name'] . '<br/>';  
}
```



# Fetch the result set

- ▶ **There are multiple ways to visit a result set**

- ▶ The SPL traversible iterator (i.e. a foreach on the resultset itself)
- ▶ Fetch the result in arrays or objects

- ▶ **Fetch :**

- ▶ `$result = $stmt->fetch(PDO::FETCH_ASSOC);` : Creates an associative array `$result` containing one record.
- ▶ If fetch is repeated, moves one record ahead
- ▶ Visit with a while (fetch returns false at the end of the selection)

```
$sql = "SELECT * FROM article";  
/*** fetch into an PDOStatement object ***/  
$stmt = $dbh->query($sql);  
while ($result = $stmt->fetch(PDO::FETCH_ASSOC)){  
    echo 'Name='.$result['name'].', price='.$result['price']." <br>  
    →/>\n";  
}
```

# Different types of Fetch

- ▶ **Into an associative array:**

```
$result = $stmt->fetch(PDO::FETCH_ASSOC);  
...  
echo $result['name'];
```

- ▶ **Into a numeric array**

```
$result = $stmt->fetch(PDO::FETCH_NUM)  
..  
echo $result[1]
```

- ▶ **Into both associative and numeric array**

```
$result = $stmt->fetch(PDO::FETCH_BOTH)  
..  
echo $result[1].', '.$result['name'];
```

# Fetch into Objects

- ▶ **Fetch can create a ad-hoc object, having the columns names as properties**

```
$obj = $stmt->fetch(PDO::FETCH_OBJ);
```

```
/** Visit the object directly */  
echo $obj->name." <br_/_>\n";  
echo $obj->price;
```

# Fetch Lazy

- ▶ PDO::FETCH\_LAZY is **odd as it combines** PDO::FETCH\_BOTH **and** PDO::FETCH\_OBJ.

```
$obj = $stmt->fetch(PDO::FETCH_LAZY);
```

```
/** Visit the object directly */  
echo $obj->name." <br_/_>\n";  
echo $obj[1]." <br_/_>\n";  
echo $obj['price'];
```

# Fetch a record into an object of a given class

- ▶ **PDO::FETCH\_CLASS instantiates a new instance of the specified class.**
  - ▶ The field names are mapped to properties (variables) within the class called.
  - ▶ This saves quite a bit of code and speed is enhanced as the mappings are dealt with internally.
- ▶ **method `fetchALL()` creates an array containing all the records.**

# Fetch a record into an object of a given class (Cont.)

```
class article{
    public $articleID;
    public $name;
    public $vatID;
    public $categoryID;
    public $price;
    public $description;
    public function displayElementLine(){
        echo $this->name.", ".$this->price." <br_/_>\n";
    }
}

...
$stmt = $dbh->query($sql);
$obj = $stmt->fetchAll(PDO::FETCH_CLASS, 'article');
foreach($obj as $article){
    $article->displayElementLine();
}
```

# Fetching into an new Object

- **We define the fetch mode of a statement**

```
$sql = "SELECT * FROM article";
```

```
/** fetch into an PDOStatement object */
```

```
$stmt = $dbh->query($sql);
```

```
/** set the fetch mode with PDO::setFetchMode() */
```

```
$stmt->setFetchMode(PDO::FETCH_INT, new article);
```

```
/** loop over the PDOStatement directly */
```

```
foreach($stmt as $article){  
    $article->displayElementLine();  
}
```

# Error Handling

- ▶ **Default: Errors are Dye statements**
  - ▶ Program is interrupted
  - ▶ Error is displayed on Screen
- ▶ **We should throw exceptions**
  - ▶ Change the error mode such that it sends exceptions,
  - ▶ Then catch all axceptions
  - ▶ It prevents an attacker to access to internal information.



# Error Handling (Cont.)

```
try {  
    $dbh = new PDO("mysql:host=$hostname;dbname=animals",  
                  $username, $password);  
    /** echo a message saying we have connected ***/  
    echo 'Connected to database<br/>';  
    /** set the error reporting attribute ***/  
    $dbh->setAttribute(PDO::ATTR_ERRMODE,  
                      PDO::ERRMODE_EXCEPTION);  
    ...  
    $dbh = null;  
}  
catch(PDOException $e){  
    echo $e->getMessage();  
}
```

# Prepared Statement

## ► A precompiled SQL statement

- Accepts 0 or more parameters
- Usefull for using a query multiple times
- Usefull for preventing SQL Injection

```
$sql = "SELECT * FROM article".  
      "WHERE articleID=:article_id OR name=:name";  
$stmt = $dbh->prepare($sql);  
$stmt->bindParam(':article_id', $article_id, PDO::PARAM_INT);  
$stmt->bindParam(':name', $name, PDO::PARAM_STR, 5);  
$stmt->execute();  
$result = $stmt->fetchAll();  
foreach($result as $row){  
    echo $row['articleID'].',';   
    echo $row['name'].',';   
    echo $row['price']. "<br/>\n";  
}
```

# Using a prepared statement

```
$article_id = 6; $name = '20_Minuten';
$sql = "SELECT * FROM article WHERE articleID=:article_id OR name\
→=:name";
$stmt = $dbh->prepare($sql);
$stmt->bindParam(':article_id', $article_id, PDO::PARAM_INT);
$stmt->bindParam(':name', $name, PDO::PARAM_STR, 5);
$stmt->execute();
$result = $stmt->fetchAll();
foreach($result as $row){ echo $row['name']." <br_/_>\n"; }
$article_id = 5;
$name = '24_Heures';
$stmt->execute();
$result = $stmt->fetchAll();
foreach($result as $row){ echo $row['name']." <br_/_>\n"; }
$article_id = 1;
$name = 'Nesquik';
$stmt->execute();
$result = $stmt->fetchAll();
foreach($result as $row){ echo $row['name'].'_/_'; }
```

# Transaction

- ▶ **Transactions are used to group requests that must remain together**
  - ▶ For efficiency reason (do not lock the file too many times)
  - ▶ For consistency of database (Group some queries that should remain together)
- ▶ **Examples**
  - ▶ Insertion of many records
  - ▶ Credit and Debit operations (Bookkeeping)
  - ▶ Delivering and stock management
  - ▶ ...
- ▶ **Syntax**
  - ▶ At the begining of the transaction  
`$dbh->beginTransaction();`
  - ▶ At the end of the transaction `$dbh->commit();`
  - ▶ In order to cancel a transaction (before commit)

# Transaction (Example)

```
try{
...
    $dbh->beginTransaction();
    $table = "CREATE_TABLE_animals_(
        animal_id_MEDIUMINT(8)_NOT_NULL_AUTO_INCREMENT_
        →PRIMARY_KEY,
        animal_type_VARCHAR(25)_NOT_NULL,
        animal_name_VARCHAR(25)_NOT_NULL
    )";
    $dbh->exec($table);
    $dbh->exec("INSERT INTO_animals_(animal_type,animal_name)_
        →VALUES_('emu','bruce')");
    ...
    $dbh->exec("INSERT INTO_animals_(animal_type,animal_name)_
        →VALUES_('lizard','bruce')");
    $dbh->commit();
    echo 'Data_entered_successfully<br_/>';
}
catch(PDOException $e){
    $dbh->rollback();
    echo $sql.'<br_/>'. $e->getMessage(),
```

# Get the index of the last inserted element

- ▶ **When inserting an element, index may be autoincremented**
  - ▶ Programmer needs a way to access the index
  - ▶ Can update other tables
  - ▶ Can create cross reference with new records

```
/** INSERT a new row */  
$sql = "INSERT INTO article (name, price) VALUES ('Laptop\  
→', 500)";  
$dbh->exec($sql);
```

```
/** display the id of the last INSERT */  
echo "The last inserted element has ID: ";  
echo $dbh->lastInsertId()." <br/>\n";
```

# Conclusion

# Conclusion

## **DB are the center of our work**

- ▶ Do not require a programmer to write HTML
- ▶ they are used to access DB's
- ▶ forms and db's are the two pillars of web programming
- ▶ a lot of other finesses to be discovered
- ▶ SQL : a semester course of 2 hours a week

## **PHP supports all data bases**

- ▶ A standard web architecture is LAMP: Linux Apache MySQL PHP