Προγραμματισμός Εφαρμογών Διαδικτύου

06

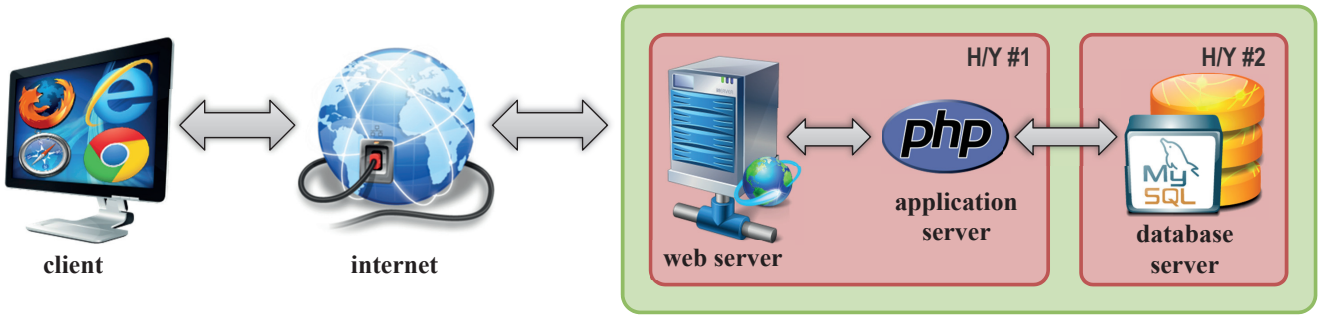
PHP & MySQL

α) Η Βιβλιοθήκη PDO (PHP Data Objects)
β) Βασικό Database Security
γ) Κρυπτογράφηση Ευαίσθητων Δεδομένων
δ) Περιορισμός Αποτελεσμάτων σε SQL Ερώτημα
ε) Σελιδοποίηση Εγγραφών Βάσης Δεδομένων με PHP
στ) Ρυθμίσεις για αποστολή email
ζ) Ανέβασμα (upload) Αρχείου στον Server

Φώτης Κόκκορας
Πανεπιστήμιο Θεσσαλίας

Βάσεις Δεδομένων & WWW (1/2)

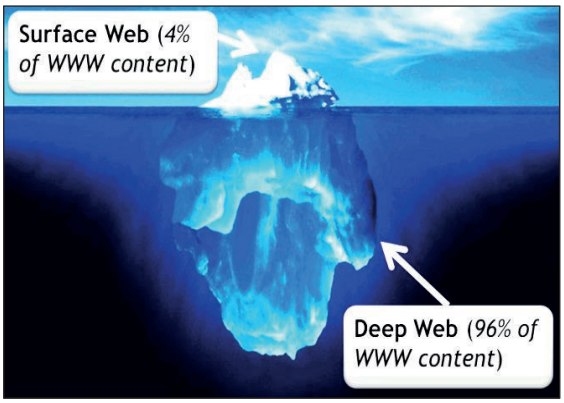
- Οι Βάσεις Δεδομένων χρησιμοποιήθηκαν ως πηγές δεδομένων για τη δημιουργία ιστοσελίδων από τα πρώτα χρόνια ύπαρξης του Παγκόσμιου Ιστού.
- Πρακτικά δε νοείται σύγχρονη εφαρμογή στον Παγκόσμιο Ιστό που να μην συμπεριλαμβάνει διασύνδεση με κάποιας μορφής Βάση Δεδομένων.
- Το κλασικό μοντέλο web εφαρμογών:



- Η υποδομή για server-side scripting (π.χ. PHP) φιλοξενείται στον ίδιο Η/Υ με τον web server.
- Ο database server (MySQL, Oracle, MS SQL Server, κτλ) δύναται να φιλοξενείται:
  - στον ίδιο Η/Υ με τον web server ή
  - σε διαφορετικό Η/Υ αν υπάρχει μεγάλος φόρτος
    - ✓ web φόρτος (πολλοί επισκέπτες)
    - ✓ database φόρτος (πολύπλοκη εφαρμογή ή/και πολλά δεδομένα)

Βάσεις Δεδομένων & WWW (2/2)

- Το μεγαλύτερο μέρος του web αποτελείται από ιστοσελίδες που δεν υφίστανται ως αρχεία αλλά δημιουργούνται δυναμικά, τη στιγμή που κάποιος τις ζητά, με βάση:
  - κάποιο **πρότυπο σελίδας** (page template)
  - **δεδομένα** προερχόμενα από κάποια βάση δεδομένων
  - κάποια **server-side-scripting** γλώσσα μέσω της οποίας θα προσδιοριστεί ποια θα είναι τα δεδομένα και πώς θα ενσωματωθούν στο πρότυπο σελίδας



- Αυτό το τμήμα του web ονομάζεται **deep web** (βαθύ web) και είναι συντριπτικά μεγαλύτερο σε μέγεθος από τις στατικές σελίδες.
  - Αναλογιστείτε πόσες ιστοσελίδες βγάζει ένα site ηλεκτρονικού καταστήματος με κατάλογο μερικών χιλιάδων προϊόντων!
  - 1 πρότυπο σελίδας X χιλιάδες προϊόντα στην database = χιλιάδες σελίδες
- Η εξάπλωση των φορητών συσκευών με πρόσβαση στον Παγκόσμιο Ιστό δημιουργεί επιπλέον ανάγκη για server-side εφαρμογές με πρόσβαση σε βάσεις δεδομένων.
  - Τέτοιες εφαρμογές δεν παράγουν απαραίτητα ιστοσελίδες! Μπορεί για παράδειγμα να τροφοδοτούν με XML δεδομένα άλλες εφαρμογές, γραμμένες για αυτές τις συσκευές (smartphones, tablets, κτλ).

PHP & MySQL

- Οι επόμενες διαφάνειες περιγράφουν τα απολύτως απαραίτητα που χρειάζεται κάποιος για να αξιοποιήσει μια βάση δεδομένων σε μια web εφαρμογή με τη βοήθεια της γλώσσας PHP.
- Η βάση δεδομένων θεωρείται ότι βρίσκεται σε database server τύπου MySQL.
  - Εξαιρετικά δημοφιλής συνδυασμός και open source σε όλες του τις διαστάσεις.
  - Δεν είναι περιοριστικό καθώς δίνεται έμφαση στην PDO, τη βιβλιοθήκη της PHP για διασύνδεση με βάσεις δεδομένων που είναι ανεξάρτητη του RDBMS που έχει επιλεγεί.
- Η PHP παρέχει δύο (εξίσου καλές) βιβλιοθήκες για σύνδεση σε βάσεις δεδομένων:
  - **SQLi**: είναι εξέλιξη (i=improved) μιας παλαιότερης βιβλιοθήκης για διασύνδεση που πλέον καταργήθηκε. Η **SQLi** παρέχει ταυτόχρονα συμβατική αλλά και αντικειμενοστραφή σύνταξη και είναι καλή επιλογή για γρήγορη μετατροπή παλαιών εφαρμογών που ήταν χτισμένες **με την παλιά βιβλιοθήκη** διασύνδεσης (εξαιτίας των παρόμοιων εντολών).
    - Η **παλιά βιβλιοθήκη** έχει επίσημα καταργηθεί εδώ και μερικά χρόνια. **DON'T USE IT!!!**
  - **PDO**: μοντέρνα, αντικειμενοστραφής βιβλιοθήκη, προσανατολισμένη στις νεώτερες εκδόσεις της MySQL αν και είναι ανεξάρτητη RDBMS. De-facto επιλογή εδώ και αρκετά χρόνια.

- Το κομμάτι "PHP και MySQL" θα το κάνουμε με τη βιβλιοθήκη PDO.
  - **Το project πρέπει να γίνει σε PDO.**

A. PDO – PHP Data Objects

- Πρόκειται για επέκταση/library της PHP που παρέχει μια τυποποιημένη προγραμματιστική διεπαφή μεταξύ PHP και του RDBMS (Συστήματος Διαχείρισης Σχεσιακών Βάσεων Δεδομένων) που χρησιμοποιείται.
  - Είναι επέκταση/extension της PHP (από έκδοση 5.1 και μετά) που πρέπει να είναι ενεργοποιημένο στο αρχείο ρυθμίσεων της PHP (php.ini σε Windows εγκατάσταση)
    - Στις πρόσφατες εκδόσεις της PHP είναι ενεργοποιημένη by default.
- On-line τεκμηρίωση: http://www.php.net/manual/en/book.pdo.php
- **Βασικά πλεονεκτήματα χρήσης του PDO:**
  - ίδιος τρόπος επικοινωνίας με database, ασχέτως του RDBMS που χρησιμοποιείται
  - μοντέρνα αντικειμενοστραφής σχεδίαση (και υλοποίηση) με ενσωματωμένο μηχανισμό εξαιρέσεων (exceptions)
  - γρήγορη διεπαφή (σε compiled C) (σε σχέση με άλλες λύσεις που είναι γραμμένες σε PHP)
  - χρήση prepared εκφράσεων (μια μορφή compiled SQL ερωτημάτων)
    - βέλτιστος τρόπος για να εκτελέσουμε ένα ερώτημα πολλές φορές, με διαφορετικές παραμέτρους κάθε φορά
    - ασφάλεια σε SQL Injection

  - υποστήριξη σε transactions (στην MySQL μόνο σε συνδυασμό με InnoDB engine)
    - εξασφαλίζει ορθή εκτέλεση εντολών ή ελεγχόμενη αναίρεση εκτέλεσης εφόσον αυτή δεν έχει ολοκληρωθεί (ACID - Atomicity, Consistency, Isolation and Durability).

PDO σε χρήση (1/2)

1. **Εκκίνηση PDO:** *(πρέπει να έχουν οριστεί πριν οι 4 παράμετροι/μεταβλητές)*
   $pdoObject = new PDO("mysql:host=$dbhost; dbname=$dbname;", $dbuser, $dbpass);
2. **Συγγραφή** παραμετρικών ερωτημάτων (παράδειγμα ερωτήματος SELECT):
   - $sql = 'SELECT * FROM movies WHERE movieCat= **:category** AND year > **:year**';
   - τα **:category** και **:year** είναι παράμετροι και θα πάρουν τιμές μετά το "prepare"
3. **Προετοιμασία** (του προηγούμενου) ερωτήματος:
   - $statement = $pdoObject -> prepare($sql);
4. **Εκτέλεση** του ερωτήματος με πέρασμα παραμέτρων στα **:category** και **:year** :
   - $statement->execute(array('**:category**'=>$myCategory, '**:year**'=>$myYear));
     - η execute επιστρέφει **false** αν αποτύχει, **true** διαφορετικά (χρήσιμο σε INSERT και UPDATE)
5. "**Κατανάλωση**" αποτελεσμάτων ερωτήματος:

```
while ( $record = $statement -> fetch() )  {
    // δες παράδειγμα SELECT παρακάτω για χειρισμό του $record
}
```

6. **Κλείσιμο ερωτήματος**: $statement->closeCursor();
   - μετά από αυτό το σημείο μπορούμε αν χρειάζεται να ξαναεκτελέσουμε το ερώτημα με νέες παραμέτρους ή να ορίσουμε και να εκτελέσουμε εντελώς νέο ερώτημα (με το ίδιο PDO object)
7. **Καταστροφή** (kill) ξεκινημένου PDO Object: $pdoObject = null;
   - την εκτελούμε όταν ολοκληρωθούν όλες οι εργασίες με την database

PDO σε χρήση (2/2)

- Αν έχετε πρόβλημα στο να πάρετε σωστά Ελληνικά σε απαντήσεις SQL ερωτημάτων ενώ τα δεδομένα στην database είναι OK, τότε μάλλον δεν είναι σωστά ρυθμισμένος ο mySQL server. Η ακόλουθη γραμμή μετά το βήμα 1 του προηγούμενου slide λογικά θα λύσει το πρόβλημα: `$pdoObject -> exec('set names utf8');`
- Είναι σημαντικό να γίνει prepare στα ερωτήματα που ενσωματώνουν POST ή GET δεδομένα (εξωτερικά δεδομένα από html forms ή από παραμέτρους σε URL).
  - Αυτό σας προστατεύει από SQL Injection καθώς η δομή του SQL "κλειδώνει" μετά το prepare και παύει να ισχύει το "πάτημα" των κακόβουλων χρηστών για SQL Injection.
- Αν το SQL ερώτημα δεν περιλαμβάνει "εξωτερικά" δεδομένα, τότε δεν χρειάζεται να κάνετε prepare. Επιπρόσθετα, τα βήματα 3 και 4 του προηγούμενου slide γίνονται:

  ```
  $statement = $pdoObject->query($sql);
  ```

  - δηλαδή εκτελούμε το SQL ερώτημα κατευθείαν (χωρίς prepare)
    - δείτε παραδείγματα σε εργαστήριο 09-10
- Μετά το βήμα 5 μπορείτε να ξανατρέξετε το ίδιο ερώτημα με διαφορετικές παραμέτρους (δηλ. όπως στο βήμα 4).
- Αν φτάσετε στο βήμα 6, μπορείτε, αντί να προχωρήσετε στο 7, να ορίσετε νέο SQL ερώτημα και πρακτικά να επαναλάβετε τα βήματα 2-6.
  - Δηλαδή, μια σύνδεση PDO μπορεί να διαχειρίζεται ταυτόχρονα πολλαπλά statements.

Διαχείριση Εξαιρέσεων / Exception Handling

```
try {
    // Στο try τμήμα μπαίνει ο κώδικας που θέλουμε να ελέγξουμε.
    // Συνήθως είναι η δημιουργία του PDO αντικειμένου, η σύνδεση
    // σε database (που μπορεί να μην είναι διαθέσιμη), κτλ
    $pdo = new PDO("mysql:host=$host;dbname=$dbname;", $dbuser, $dbpass);
} catch (PDOException $e) {  //μέσω της $e βλέπουμε τι "κακό" συνέβη!
    // Εδώ μέσα θα βρεθούμε μόνο όταν συμβεί κάτι πολύ κακό μέσα στο
    // try{...} κομμάτι του κώδικα, όπως πχ αδυναμία εκκίνησης του PDO.
    // Τύπωσε τα μηνύματα λάθους και διέκοψε την εκτέλεση.
    print "Database Error: " . $e->getMessage();
    // άμεση διακοπή εκτέλεσης του PHP κώδικα της σελίδας
    die("Αδυναμία δημιουργίας PDO Object");
}
```

- **PDOException**: προκαθορισμένη κατηγορία εξαιρέσεων του PDO
  - Βάζοντας πολλές κρίσιμες εντολές στο try block, ίσως χάσουμε τη δυνατότητα να εντοπίσουμε το ακριβές πρόβλημα μιας εξαίρεσης γιατί βλέπουμε μόνο την τελευταία!
  - **getMessage( )**: μέθοδος του PDOException object που επιστρέφει πληροφορίες για την τελευταία εξαίρεση που συνέβη.
- Η εντολή **die** (ή **exit**) τερματίζει την εκτέλεση της PHP, τυπώνοντας το μήνυμα.

Παράδειγμα UPDATE

- **Σενάριο**: έστω μια φόρμα που στέλνει με POST τον κωδικό (movieID), τον τίτλο (movieTitle) και το έτος κυκλοφορίας (movieYear) μιας κινηματογραφικής ταινίας, στο πλαίσιο μιας **διαδικασίας μεταβολής στοιχείων**.
  - ο πίνακας λέγεται **movies** και έχει πεδία **movieID**, **movieTitle** και **movieYear**
- Στο αρχείο που παραλαμβάνει τα δεδομένα του POST θα έχουμε:

```
//αρχικοποίηση PDO ((πρέπει να έχουν οριστεί πριν οι 4 παράμετροι/μεταβλητές)
1  $pdoObject = new PDO("mysql:host=$dbhost;dbname=$dbname;", $dbuser, $dbpass);
   //δημιουργία παραμετρικού ερωτήματος
2  $sql="UPDATE movies SET movieTitle=:myMovieTitle, movieYear=:myMovieYear, WHERE movieID=:myMovieID";
   //compile ερωτήματος σε PDO
3  $statement = $pdoObject->prepare($sql);
   //πέρασμα παραμε΄τρων και εκτέλεση ερωτήματος
4  $result=$statement->execute( array( ':myMovieID'=>$_POST['movieID'],
                                       ':myMovieTitle'=>$_POST['movieTitle'],
                                       ':myMovieYear'=>$_POST['movieYear'] )   );
   //απελευθέρωση πόρων που δέσμευσε το PDO στο server
5  $statement->closeCursor();
6  $pdoObject = null;
```

- **Σημείωση**: για λόγους απλότητας, δεν υπάρχει κώδικας παγίδευσης "απρόοπτων" καταστάσεων (**try...catch** κτλ), **όπως θα έπρεπε**! Επίσης δεν υπάρχει έλεγχος αν τελικά έγινε το UPDATE (γραμμή 4, μεταβλητή **$result**).

Παράδειγμα SELECT

- Στο **σενάριο** του προηγούμενου slide, έστω θέλουμε **να εμφανίσουμε** σε μια σελίδα τα στοιχεία των ταινιών δεδομένης χρονιάς, π.χ. της χρονιάς: **$_POST['movieYear']**
- Στο αρχείο που παραλαμβάνει τα δεδομένα του POST θα έχουμε:

```
1  $pdoObject = new PDO("mysql:host=$dbhost;dbname=$dbname;", $dbuser, $dbpass);
2  $sql = "SELECT * FROM movies WHERE movieYear=:myMovieYear";
3  $statement = $pdoObject->prepare($sql);
   //πέρασμα παραμέτρων και εκτέλεση ερωτήματος
4  $statement->execute( array(':myMovieYear'=>$_POST['movieYear'] ) );
   //loop "κατανάλωσης" αποτελεσμάτων ερωτήματος
5  while ( $record = $statement -> fetch() )  {
6      echo $record['movieID'].' - '.$record['movieYear'].' - '.$record['movieTitle'].'<br/>';
7  }   //όταν εξαντληθούν οι εγγραφές, $record=false οπότε έχουμε έξοδο από το while loop
8  $statement->closeCursor();
9  $pdoObject = null;
```

- **Σημείωση**: για λόγους απλότητας, δεν υπάρχει κώδικας παγίδευσης "απρόοπτων" καταστάσεων (**try...catch** κτλ), **όπως θα έπρεπε**!
  - Σημεία ελέγχου μπορεί να είναι η 4 ή/και η 5 γραμμές που επιστρέφουν false σε αποτυχία.
  - Τα σημεία προσοχής για "απρόοπτα" είναι κύρια οι γραμμές 1 και 3 που δημιουργούν objects στη μνήμη της PHP.
- Πιθανό αποτέλεσμα του κώδικα φαίνεται δεξιά:

```
1 - 2009 - Avatar
2 - 2009 - District 9
3 - 2009 - The Reader
```

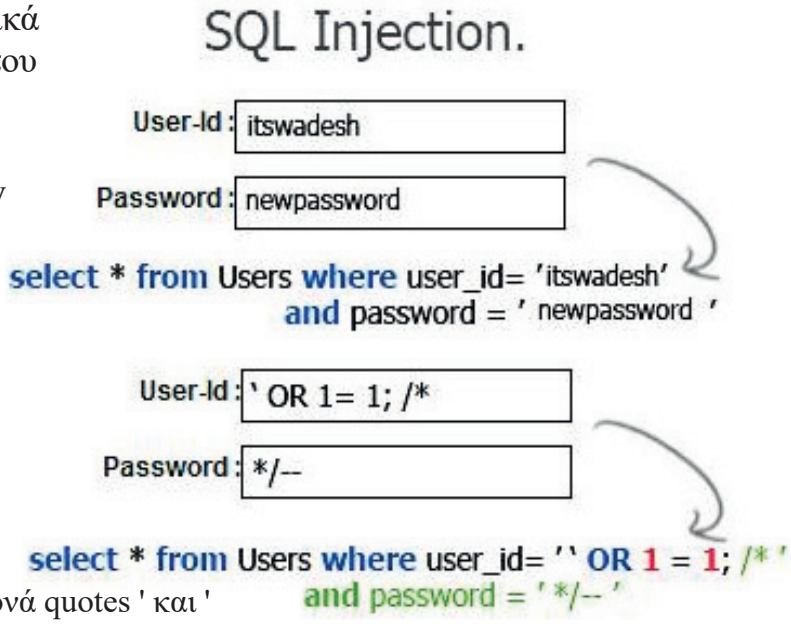
B. Βασικό Database Security

- Η εφαρμογή σας ΔΕΝ πρέπει να συνδέεται στην database με το λογαριασμό κάποιου superuser (π.χ. root – ο default admin του MySQL) ή του database owner.
  - Αν κάποιος πάρει τον έλεγχο τότε μπορεί, το λιγότερο, να "καταστρέψει" τον server.
- Χρησιμοποιήστε/φτιάξτε database users που έχουν ακριβώς τα δικαιώματα που απαιτεί η εφαρμογή – τίποτα λιγότερο – τίποτα περισσότερο!
  - Έτσι, αν κάποιος αποκτήσει πρόσβαση στη ΒΔ μέσω των username/password της εφαρμογής (της PHP δηλαδή), στη χειρότερη περίπτωση θα κάνει ότι κάνει και η εφαρμογή!

- Ελέγξτε στο server όσο γίνεται καλύτερα ότι το input σε φόρμες είναι του αναμενόμενου τύπου.
  - Το validation στον client είναι για χρηστικούς κυρίως λόγους και αποσυμφόρηση του server από ελέγχους - μπορεί εύκολα να παρακαμφθεί από κακόβουλο χρήστη φτιάχνοντας μια άλλη φόρμα που κάνει POST στην ίδια σελίδα με την αρχική, αλλά χωρίς ελέγχους!
- Στο τελικό site, ρυθμίστε server/PHP (στο php.ini) να ΜΗΝ βγάζουν αναλυτικά μηνύματα σφαλμάτων γιατί μπορεί να αποκαλυφθούν ονόματα πεδίων, πινάκων, κτλ.
- Αποφεύγετε όσο γίνετε την υλοποίηση της λογικής της εφαρμογής (business logic) μέσα στις σελίδες (π.χ. κατασκευή SQL ερωτημάτων)
  - Προτιμείστε τη χρήση queries/views, stored procedures, triggers κτλ μέσα στο RDBMS.
- Εγκαταστήστε το RDBMS σε υποδίκτυο (εικονικό ή πραγματικό) μη προσβάσιμο εξωτερικά!

SQL Injection

- **ΚΑΚΟΒΟΥΛΗ** τεχνική για παράκαμψη διαδικασιών πιστοποίησης (authentication checks) και ελέγχων άδειας (authorization checks), ή ακόμη και απόκτηση πρόσβασης σε λειτουργίες του λειτουργικού συστήματος (OS compromise).
  - Συνήθως γίνεται με μεταβολή ενός αρχικά ασφαλούς SQL ερωτήματος με τρόπο που να θέτει σε κίνδυνο την ασφάλεια δεδομένων ή/και του server.
  - Η μεταβολή μπορεί να γίνει αν το query χτίζεται παραμετρικά (κάτι σύνηθες!).
  - **Παράδειγμα**:



- Για SQL Injection δείτε το τμήμα Database Security της PHP:
  - http://www.php.net/manual/en/security.database.php
    - στο SQL δεξιά, πριν το OR υπάρχουν 2 μονά quotes ' και '

Η PDO σε συνδυασμό με τη χρήση prepared ερωτημάτων καλύπτει από SQL Injection!

2ο Παράδειγμα SQL Injection

```php
<?php
// We didn't check $_POST
['password'], it could be anything the user wanted! For example:
$_POST['username'] = 'aidan';
$_POST['password'] = "' OR ''='";

// Query database to check if there are any matching users
$query = "SELECT * FROM users WHERE user='{$_POST
['username']}' AND password='{$_POST['password']}'";
mysql_query($query);

// This means the query sent to MySQL would be:
echo $query;
?>
```

The query sent to MySQL:

```
SELECT * FROM users WHERE user='aidan' AND password='' OR ''=''
```

- Το τελευταίο δίνει πρόσβαση στον οποιοδήποτε γνωρίζει ΜΟΝΟ ένα username!!!
- **Σημείωση**: χρησιμοποιήθηκε η παλιά βιβλιοθήκη (όχι κάποια από τις SQLi ή PDO)

Γ. Κρυπτογράφηση Ευαίσθητων Δεδομένων (1/2)

- Σε εφαρμογές που διακινούνται πολύτιμα δεδομένα (ό,τι και αν σημαίνει αυτό) κάποια δεδομένα ΔΕΝ πρέπει να αποθηκεύονται στην πρωταρχική τους μορφή.
  - π.χ. passwords
- Η ενδεικνυόμενη λύση είναι να αποθηκεύονται στην database αφού πρώτα κρυπτογραφηθούν.
  - Η χρήση του ασφαλούς https πρωτοκόλλου δεν καταργεί αυτή την ανάγκη καθώς κάποια δεδομένα πρέπει να είναι ασφαλή ΚΑΙ μέσα στην database και όχι μόνο όταν διακινούνται πάνω στο δίκτυο.
    - π.χ. τα passwords δεν πρέπει να μπορεί να τα δει ούτε καν ο διαχειριστής του RDBMS!
- Η γλώσσα PHP παρέχει συναρτήσεις κρυπτογράφησης όπως η **md5( )** και η **sha1( )**, πλην όμως επειδή είναι γρήγορες συναρτήσεις **ευνοούν επιθέσεις brute force**.
  - Θεωρητικά, με περίσσια υπολογιστικής ισχύος και αρκετό χρόνο στη διάθεσή του, ένας κακόβουλος χρήστης μπορεί να μαντέψει το αρχικό password που όταν κρυπτογραφηθεί με δεδομένο τρόπο παράγει κρυπτογραφημένο password (hash) ίδιο με το αποθηκευμένο στην database – δεν είναι γενικά εύκολο, είναι όμως εφικτό.

- Οπότε, **πώς κρυπτογραφούμε τα passwords;**

Κρυπτογράφηση Ευαίσθητων Δεδομένων (2/2)

- Χρήση συναρτήσεων `crypt()` ή `hash()`:
  - `crypt()`: υποστηρίζει αρκετούς αλγορίθμους κρυπτογράφησης και εγγυάται τη διαθεσιμότητά τους καθώς υπάρχουν native υλοποιήσεις στην php (v.5.3 και μετά).
  - `hash()`: παρέχει περισσότερους αλγορίθμους κρυπτογράφησης από την `crypt()` αλλά όχι όλους όσους παρέχει η `crypt()` και επιπλέον είναι υλοποιημένη σε επέκταση της php και όχι στον πυρήνα της όπως η `crypt()`.
  - Αμφότερες, με κατάλληλο αλγόριθμο, είναι **υπολογιστικά πολύπλοκες** (δεν ευνοούν brute force attacks) και επιπλέον **υποστηρίζουν cryptographic salt**.
    - **cryptographic salt**: τυχαίο αλφαριθμητικό που προστίθεται στο password με σκοπό την αποτροπή εύρεσής του κοιτώντας σε πίνακες (rainbow tables) με προϋπολογισμένα ζευγάρια password και κρυπτογραφημένου password (κάτι που επιτρέπει πολλές δοκιμές σε μικρό χρόνο)
- **Για κρυπτογραφημένη αποθήκευση password (και γενικότερα κειμένου):**
  - παράγουμε κατάλληλο salt (SOS – επηρεάζει το ποιος αλγόριθμος θα χρησιμοποιηθεί!)
  - κρυπτογραφούμε το password με το salt (π.χ. με την συνάρτηση **crypt**)
  - αποθηκεύουμε στην database το **username** και το **κρυπτογραφημένο password**
    - το salt υπάρχει ως πρόθεμα στο αποτέλεσμα της κρυπτογράφησης – δεν χρειάζεται αποθήκευση

- **Για επαλήθευση password (διαπίστευση χρήστη):**
  - Με βάση το username που δίνει ο χρήστης, βρίσκουμε το salt που αποθηκεύσαμε για αυτόν στην database, κρυπτογραφούμε με ίδιο τρόπο το password και το συγκρίνουμε με το κρυπτογραφημένο password της ΒΔ. (Υπάρχει και Β τρόπος – θα το δούμε στο τέλος).

Επιλογή Αλγόριθμου Κρυπτογράφησης (1/2)

- Τα παρακάτω ισχύουν για την PHP έκδοσης τουλάχιστον 5.3 (Μάιος 2013). Η PHP έκδοση 5.5 έχει πρόσθετες συναρτήσεις (password_hash και password_verify).
  - Για v.5.5 δείτε: http://www.php.net/manual/en/ref.password.php
- Η σύνταξη της συνάρτησης crypt στην PHP είναι: `crypt($password,$salt)`
  - Οι δύο παράμετροι είναι αλφαριθμητικά. Η 2η παράμετρος είναι προαιρετική. Αν δεν την παρέχουμε χρησιμοποιείται MD5 αν είναι διαθέσιμος, διαφορετικά ο Standard DES. Η συνάρτηση επιστρέφει το κρυπτογραφημένο password.
  - Ανάλογα με το σύστημα στο οποίο τρέχει η PHP, υποστηρίζονται διάφοροι αλγόριθμοι κρυπτογράφησης. Αυτό μπορούμε να το μάθουμε ελέγχοντας την τιμή κάποιων σταθερών. Αν είναι 1 υποστηρίζεται ο αλγόριθμος, διαφορετικά δεν υποστηρίζεται. Οι σταθερές είναι:
    - `CRYPT_STD_DES` για Standard DES
    - `CRYPT_EXT_DES` για Extended DES
    - `CRYPT_MD5` για MD5
    - `CRYPT_BLOWFISH` για Blowfish
    - `CRYPT_SHA256` για SHA-256
    - `CRYPT_SHA512` για SHA-512
  - Μπορούμε να καθορίσουμε τον αλγόριθμο κρυπτογράφησης της `crypt()` μέσω του salt που θα δώσουμε! Πλήρης τεκμηρίωση υπάρχει στην διεύθυνση:
    - http://php.net/manual/en/function.crypt.php

Επιλογή Αλγόριθμου Κρυπτογράφησης (2/2)

Η δομή του salt καθορίζει τον αλγόριθμο κρυπτογράφησης!

ΠΡΟΣΟΧΗ! Να έχετε επαρκές μήκος στην database για την αποθήκευση του παραγόμενου hash

Στο παράδειγμα, κρυπτογραφούμε το αλφαριθμητικό 'myusername'.

- **Standard DES**: salt δύο χαρακτήρων από τους "**./0-9A-Za-z**"

```
if (CRYPT_STD_DES == 1) {
    echo 'Standard DES: ' . crypt('myusername', 'ko');
} else echo 'Standard DES is not supported';     //koBey1NE0LjtE     (μήκος 13)
```

- **Extended DES**: "salt" μήκους 9 χαρακτήρων με πρώτο χαρακτήρα το _ ακολουθούμενο από **4 bytes** για το πλήθος iteration και **4 bytes** για το salt. Περισσότερα on-line

```
if (CRYPT_EXT_DES == 1) {
    echo 'Extended DES: ' . crypt('myusername', '_J9..kokk');
} else echo ''Extended DES is not supported';    //_J9..kokku2/6LhSJExc     (μήκος 20)
```

- **MD5**: salt μήκος 12 χαρακτήρων που ξεκινά με **1** και τελειώνει με **$**

```
if (CRYPT_MD5 == 1) {
    echo 'MD5: ' . crypt('myusername', '$1$kokkoras$');
} else echo ' MD5 is not supported';     //$1$kokkoras$93f8HVhV2uelJp.dzs4fe1     (μήκος 34)
```

- **Blowfish**: "salt" που ξεκινά με "**$2a$**" (για php < v5.3.7) ή "**$2y$**" (για php ≥v5.3.7), ακολουθούμενο από διψήφια παράμετρο κόστους μεταξύ 04 και 31, ακολουθούμενο από **$**, ακολουθούμενο από 22 χαρακτήρες από τους "**./0-9A-Za-z**"


```
if (CRYPT_BLOWFISH == 1) {
    echo 'Blowfish: ' . crypt('myusername', '$2y$07$use.some.silly.string2');
} else echo ''Blowfish is not supported';
    //$2y$07$use.some.silly.stringu5pd8Tq3/OFWw7ouIACVgweD4K.cxqzO     (μήκος 60)
```

- **Παρατήρηση**: Για κάποιο λόγο το hash δεν ξεκινά με το salt, όπως γίνεται συνήθως! Χάνει τον τελευταίο χαρακτήρα του salt. Ίσως είναι κάποιο bug…

- **SHA-256**: Το "salt" ξεκινάει με **5** ακολουθούμενο από **16** χαρακτήρες. Αν πριν από τους 16 χαρακτήρες βάλουμε το '**rounds=N$**' (όπου **N** ακέραιος μεταξύ 1000 και 999999999) τότε ο αλγόριθμος θα τρέξει το loop κρυπτογράφησης **N** φορές. Εξ ορισμού (by default) το loop κρυπτογράφησης τρέχει 5000 φορές.
  - Το N είναι το αντίστοιχο της παραμέτρου cost του Blowfish.

```
if (CRYPT_SHA256 == 1) {
    echo 'SHA-256: ' . crypt('myusername', '$5$rounds=6000$usesomesillystri');
} else echo ''SHA-256 is not supported';
//$5$rounds=6000$usesomesillystri$lqHyBAkcHU7P0kTRbIutp976BnfKIYclnFMrte8agJ1 (μήκος 75)
```

- **SHA-512:** Το salt ορίζεται όπως στον **SHA-256** αλλά με **6** στην αρχή.

```
if (CRYPT_SHA512 == 1) {
    echo 'SHA-512: ' . crypt('myusername', '$6$usesomesillystri');      //5000 rounds
} else echo ' SHA-512 is not supported';                                (μήκος 106)
$6$usesomesillystri$2/bfsiEX0ltg1uKM9K7EoPTUAxNO8hxBxI8qaiSuywM3Qj4saoTaIfuhy6FHyXI944DctqEfN.tQtYKSDLJ4x/
```

- **Σε όλες τις περιπτώσεις, μεγαλύτερο salt από το ζητούμενο, ΔΕΝ επηρεάζει το hash.**

Παράδειγμα Κρυπτογράφισης (με PDO)

- Έστω πίνακας (table) **users** με τα πεδία (fields): **username**, **password**
- Έστω τα δεδομένα προς αποθήκευση: **$myUsername**, **$myPassword**

```
    //εκκίνηση PDO (πρέπει να έχουν οριστεί πριν οι 4 παράμετροι/μεταβλητές)
1   $pdoObject = new PDO("mysql:host=$dbhost; dbname=$dbname;", $dbuser, $dbpass);
    //διαμόρφωση παραμετρικού ερωτήματος
2   $sql = 'INSERT INTO users VALUES (:username, :password)';
    //παραγωγή τυχαίου οκταψήφιου αριθμού, ως salt για MD5 (καλύτερα να περιέχει και γράμματα)
3   $salt = '$1$'.rand(10000000,99999999).'$';
    //κρυπτογράφηση password
4   $encryptedPass = crypt($myPassword, $salt);  //encrypt the password
    //compile ερωτήματος σε PDO
5   $statement = $pdoObject -> prepare($sql);
    //πέρασμα τιμών στις παραμέτρους του ερωτήματος και εκτέλεση
6   $statement->execute( array(':username' =>$myUsername, ':password'=>$encryptedPass) );
    //απελευθέρωση πόρων που δέσμευσε η PDO
7   $statement->closeCursor();
8   $pdoObject=null;
```

- Με επιφύλαξη για την περίπτωση του Blowfish (βλ. σχετική παρατήρηση)
- Τα ίδια βήματα ακολουθούνται και στην περίπτωση χρήσης της βιβλιοθήκης **SQLi**.
  - Προφανώς οι εντολές προσαρμόζονται στην "συνταγή" της **SQLi** **(βλ. επόμενο slide)**.

Παράδειγμα Κρυπτογράφισης (με SQLi)

(αλλά και παράδειγμα για αντιπαραβολή στη σύνταξη αντικειμενοστραφούς SQLi με PDO προηγούμενου slide)

- Αφορά στο παράδειγμα του προηγούμενου slide.

```
    //εκκίνηση SQLi (πρέπει να έχουν οριστεί πριν οι 4 παράμετροι/μεταβλητές)
1   $mysqli = new mysqli($dbhost, $dbuser, $dbpass, $dbname);
    //διαμόρφωση παραμετρικού ερωτήματος (για τον πίνακα του προηγούμενου slide)
2   $sql = 'INSERT INTO users VALUES (?, ?)';
    //παραγωγή τυχαίου οκταψήφιου αριθμού, ως salt για MD5 (καλύτερα να περιέχει και γράμματα)
3   $salt = '$1$'.rand(10000000,99999999).'$';
    //κρυπτογράφηση password
4   $encryptedPass = crypt($myPassword,$salt);
    //compile ερωτήματος σε SQLi
5   $statement = $sqli -> prepare($sql);
    //πέρασμα τιμών στις παραμέτρους του ερωτήματος
6   $statement->bind_param("ss", $myUsername, $encryptedPass);  //"ss": 2 αλφαριθμητικά
    //εκτέλεση ερωτήματος
7   $statement->execute();
    //απελευθέρωση πόρων που δέσμευσε η SQLi
8   $statement->close();
9   $sqli->close();
```

- Στο παραπάνω παράδειγμα ακολουθήθηκε η αντικειμενοστραφής σύνταξη της **SQLi**.
- Σε αμφότερα τα παραδείγματα και για λόγους απλότητας, δεν υπάρχει κώδικας παγίδευσης "απρόοπτων" καταστάσεων (**try {…} catch{…}**), **όπως θα όφειλε**.

Επαλήθευση password

- Κατά την ταυτοποίηση, ο χρήστης δίνει username και password. Στη συνέχεια:
  - Με βάση το **username** του χρήστη βρίσκουμε στην database (με SELECT) το **κρυπτογραφημένο password** (έστω ότι είναι `$hash`).
  - Ο χρήστης ταυτοποιείται αν: `crypt($password,$hash)==$hash`
- Παρατηρήστε ότι **ΔΕΝ** χρειάζεται αποθήκευση του **salt** της κρυπτογράφησης καθώς:
  - το salt βρίσκεται πάντα στην αρχή του hash
  - στην αρχή του salt, οι πρώτοι χαρακτήρες ορίζουν τον αλγόριθμο κρυπτογράφησης, άρα είναι γνωστό πόσο μεγάλο αρχικό τμήμα του hash είναι το salt
  - μεγαλύτερο salt από το ζητούμενο, ΔΕΝ λαμβάνεται υπόψη

- Άρα τελικά, παρόλο που βάλαμε `$hash` στη 2η παράμετρο της crypt, αυτό που τελικά χρησιμοποιεί η crypt είναι το τμήμα που αντιστοιχεί στο salt!
- **ΥΠΕΝΘΥΜΙΣΗ**: η αποθήκευση κρυπτογραφημένου password είναι το μισό της υπόθεσης "ασφαλής διαπίστευση χρήστη". Στις φόρμες εγγραφής-επαλήθευσης-login πρέπει επιπλέον να χρησιμοποιείται το πρωτόκολλο **https** που διακινεί τα δεδομένα κρυπτογραφημένα.
- **ΠΡΟΣΟΧΗ**: Στα δύο παραδείγματα κρυπτογράφησης στα slides #20 και #21, **για απλοποίηση** χρησιμοποιήσαμε salt **8 ψηφίων** και όχι **8 χαρακτήρων** γενικότερα.

Δ. Περιορισμός Αποτελεσμάτων SQL Ερωτήματος

- Μέσω της παραμέτρου **Limit N, K** (στη MySQL) μπορούμε να επιλέξουμε υποσύνολο των αποτελεσμάτων ενός SQL ερωτήματος, συγκεκριμένα τις πρώτες **K** εγγραφές αρχίζοντας από αυτή με **δείκτη** N.
  - Υπενθυμίζεται ότι η πρώτη εγγραφή έχει δείκτη 0, η δεύτερη 1, κ.ο.κ.
- **Παραδείγματα:**

```
SELECT * FROM someTable LIMIT 0, 10
```

  - Θα επιστρέψει τις 10 πρώτες εγγραφές από τα αποτελέσματα (αρχίζοντας από αυτή με δείκτη 0 (την πρώτη δηλαδή) μέχρι την εγγραφή με δείκτη 9 (δηλαδή την 10η)

```
SELECT * FROM someTable LIMIT 4, 5
```

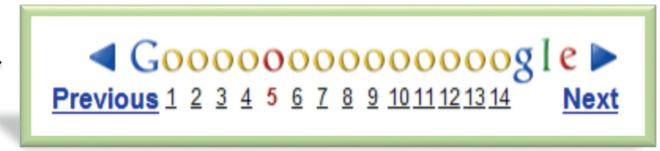
  - Θα επιστρέψει 5 εγγραφές, αρχίζοντας από αυτή με δείκτη 4, δηλαδή τις εγγραφές με δείκτη 4, 5, 6, 7 και 8 (που αντιστοιχούν στις 5η, 6η, ..., 9η πραγματικές εγγραφές).

  - Αν μετά το `LIMIT` υπάρχει ένας ακέραιος `K`, τότε σημαίνει το πλήθος εγγραφών που θέλουμε, ξεκινώντας από την 1η εγγραφή (δηλ. `K` εγγραφές αρχίζοντας από την πρώτη).

Ε. Σελιδοποίηση Εγγραφών ΒΔ με PHP (1/2)

❖ **Ζητούμενο**: Να μην εμφανίζονται πολλά δεδομένα εγγραφών σε μία "ατέλειωτη" σελίδα (π.χ. σε σελίδες προβολής αποτελεσμάτων αναζήτησης).

- ❑ **Λύση**: Σελιδοποίηση των αποτελεσμάτων
- ❑ **Παράδειγμα**: Google Search Results



❖ **Τι χρειάζεται για να φτιάξουμε το query;**

- ❑ Το πλήθος αποτελεσμάτων που θέλουμε σε κάθε σελίδα. Έστω **k=10**.
- ❑ Μια μεταβλητή στο URL που να μας λέει ποιας σελίδας τα αποτελέσματα θα δείξουμε. Έστω ότι είναι να δείξουμε την 3η σελίδα (έστω **p=3**).
  - Δηλ. θα δείξουμε τις εγγραφές με δείκτη **20** ως **29** (η αρίθμηση των δεικτών ξεκινάει από το 0) ή γενικότερα, τις **k** εγγραφές αρχίζοντας από την εγγραφή με δείκτη **(p-1)*k**
- ❑ Προσθέτουμε παράμετρο `LIMIT N,K` στο τέλος του ερωτήματος που θα φέρει τα δεδομένα:
  - Το **N** = **(p-1)*k** ορίζει το δείκτη της πρώτης εγγραφή από τη δεκάδα εγγραφών που μας ενδιαφέρει, εδώ 20.
  - Το K ορίζει πόσες εγγραφές θέλουμε (εδώ 10)
- ❑ Άρα γίνεται: SELECT * FROM myDATA LIMIT 20,10

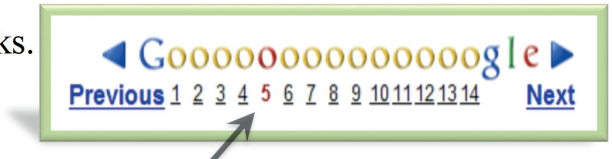
Σελιδοποίηση Εγγραφών ΒΔ με PHP (2/2)

❖ **Τι άλλη πληροφορία χρειάζεται;**

- ❑ Το συνολικό πλήθος των αποτελεσμάτων (όχι της σελίδας – αυτό είναι **K**) για να μπορέσουμε να υπολογίσουμε το πλήθος των σελίδων.
- ❑ Αυτό υπολογίζεται εκτελώντας παρόμοιο query αλλά χωρίς το LIMIT στο τέλος. Π.χ.:
  - `SELECT count(*) AS counter FROM myDATA`
  - Αν το αρχικό ερώτημα έχει conditions (`WHERE...`) αυτά πρέπει να κρατηθούν και στο ερώτημα υπολογισμού του πλήθους των αποτελεσμάτων!
- ❑ Σε MySQL, μπορεί να φανεί χρήσιμη και η μέθοδος `rowCount()` του PDO, που επιστρέφει το πλήθος των εγγραφών που "επηρέασε" (επέλεξε, διέγραψε, μετέβαλε, κτλ) το query.
  - σε PDO/MySQL: `$statement->rowCount();`

❖ **Τι χρειάζεται στη σελίδα;**

- ❑ Σε κατάλληλο μέρος της σελίδας πρέπει να βάλουμε λίστα με links για πλοήγηση στις σελίδες των αποτελεσμάτων.
- ❑ Γίνεται με ένα loop μέσα στο οποίο "γράφουμε" links.
- ❑ Για τα "όρια" του loop πρέπει να ξέρουμε:
  - το **σύνολο σελίδων** (βλ. παραπάνω)
  - **τις εγγραφές ανά σελίδα** (τις έχουμε ήδη προαποφασίσει – βλ. **K** σε προηγούμενο slide)
  - την **τρέχουσα σελίδα** (για να μην την κάνουμε link – βλ. πχ το 5 στην Google πλοήγηση)



Παράδειγμα Σελιδοποίησης (1/2)

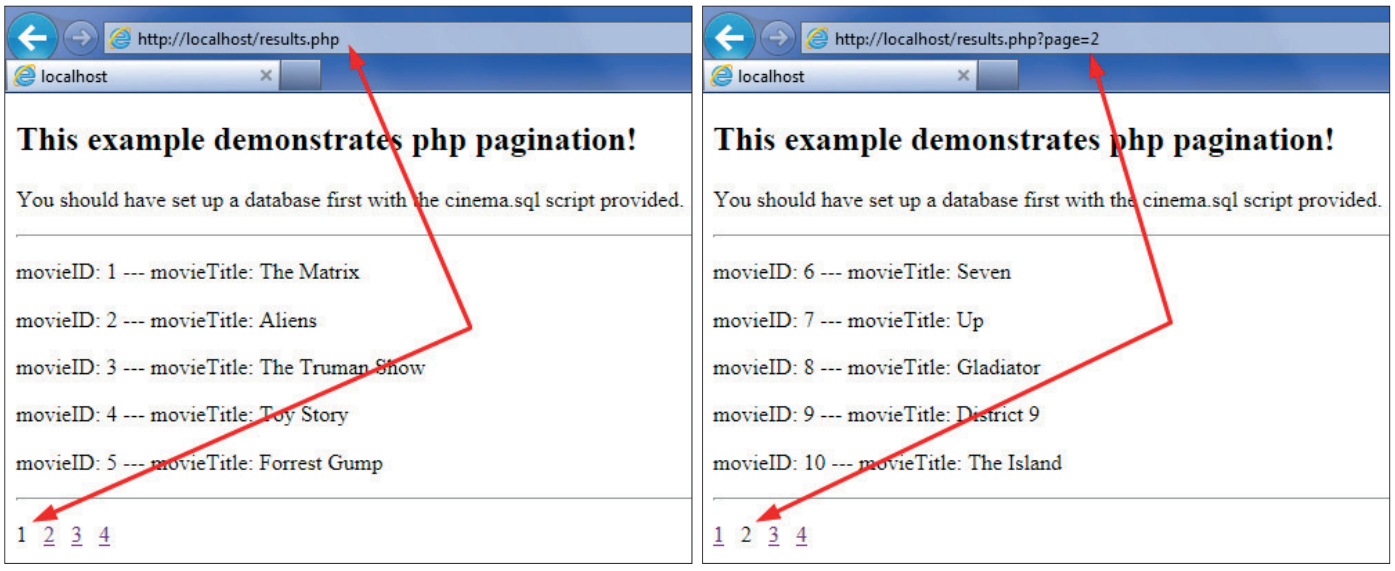
(με βάση το αρχείο pagination_example.zip)

- Έστω η database cinema που περιέχει ένα πίνακα movies με πεδία **movieID** και **movieTitle** και 19 καταχωρήσεις συνολικά (δημιουργήστε τη στο Workbench με το sql script cinema.sql).
- Βάλτε το αρχείο results.php στον φάκελο htdocs του XAMP σας και με τον browser ζητήστε τη σελίδα http://localhost/results.php
- Δείτε τα σχόλια στο results.php. Οι βασικές παράμετροι της σελιδοποίησης είναι:
  - `$recordsPerPage`: πόσα αποτελέσματα ανά σελίδα θέλουμε να δείχνουμε
  - `$curPage`: ποια σελίδα αποτελεσμάτων φτιάχνουμε (τροφοδοτείται από παράμετρο στο URL – αν δεν υπάρχει παράμετρος είμαστε στην πρώτη σελίδα)
  - `$pages`: πόσες σελίδες χρειάζονται συνολικά τα αποτελέσματα που θα τυπώσουμε
- **Παρατήρηση 1**: Στον κώδικα του παραδείγματος, τα queries υπολογισμού του πλήθους των απαντήσεων και εύρεσης των σχετικών εγγραφών, είναι ενσωματωμένα στο results.php. Αν το ερώτημα σχηματίζεται δυναμικά, με βάση κάποια φόρμα αναζήτησης, τότε είναι βολικότερο το πλήθος των αποτελεσμάτων και το query αναζήτησης να αποθηκευτούν σε session μεταβλητές πριν κληθεί το results.php και ο κώδικας του results.php να τα διαβάζει από εκεί. Αναλογιστείτε ότι τα δεδομένα του POST δεν θα είναι διαθέσιμα στις επόμενες σελίδες αποτελεσμάτων!

Παράδειγμα Σελιδοποίησης (2/2)

- **Παρατήρηση 2**: Στο results.php δεν γίνεται χρήση prepared ερωτημάτων καθώς όλες οι εμπλεκόμενες παράμετροι είναι ορισμένες από εμάς. Επιπλέον, όλα τα queries εκτελούνται μόνο μία φορά.

- Ακολουθούν οι δύο πρώτες σελίδες αποτελεσμάτων του παραδείγματος.
  - Προσέξτε την παράμετρο **page** στο URL στην εικόνα δεξιά.



ΣΤ. Αποστολή email μέσω PHP (1/2)

- Η απλή αποστολή email μέσω PHP είναι θέμα μιας εντολή:
  - `$result = mail($to, $subject , $message);`
    - `$to` το email του παραλήπτη, `$subject` το θέμα του email, `$message` το σώμα του μηνύματος και `$result` το αποτέλεσμα εκτέλεσης της εντολής (1/true αν γίνει η αποστολή).
  - Η εντολή έχει και 4η προαιρετική παράμετρο για κοινοποίηση (cc), bcc, κτλ.
    - Δείτε πλήρη τεκμηρίωση: http://php.net/manual/en/function.mail.php
  - Για αλλαγή γραμμής στο σώμα του κειμένου, γράψτε `"\r\n"`
    - ισχύει και γενικότερα, όταν π.χ. γράφετε text σε αρχείο (θα το δούμε ξανά σε PHP & XML)

- Για να λειτουργήσει η εντολή σε XAMP, χρειάζονται κάποιες ρυθμίσεις.
  - Παράδειγμα ρυθμίσεων σε συνδυασμό με έναν λογαριασμό `test1` σε GMail, που έχει password `mypass1`:

Στο αρχείο: `C:\xampp\sendmail\sendmail.ini`	Στο αρχείο: `C:\xampp\php\php.ini`

`smtp_server= smtp.gmail.com`
`smtp_port= 587`
`smtp_ssl= auto`
`auth_username= test1@gmail.com`
`auth_password= mypass1`
	`sendmail_path= "C:\xampp\sendmail\sendmail.exe"`
	
	έχει δοκιμαστεί με XAMPP – λειτουργεί ΟΚ

Αποστολή email μέσω PHP (2/2)

- ...συνέχεια από προηγούμενο slide:
  - Παράδειγμα ρυθμίσεων σε συνδυασμό με ένα λογαριασμό `test2` σε HotMail, που έχει password `mypass2`:

Στο αρχείο: `C:\xampp\sendmail\sendmail.ini`	Στο αρχείο: `C:\xampp\php\php.ini`

`smtp_server= smtp-mail.outlook.com`
`smtp_port= 587`
`smtp_ssl= auto`
`auth_username= test2@hotmail.com`
`auth_password= mypass2`
	`sendmail_path="C:\xampp\sendmail\sendmail.exe"`
	
	ΔΕΝ έχει δοκιμαστεί – αν έχετε hotmail δοκιμάστε το

- Βασικές προϋποθέσεις για να λειτουργήσει η εντολή σε GMail και Hotmail είναι:
  - να υπάρχει ένας ενεργός λογαριασμός email σε αυτές τις υπηρεσίες
  - ο SMTP server να επιτρέπει αποστολή από τρίτο domain
    - για παράδειγμα, με τον SMTP server του πρώην ΤΕΙ Θεσσαλίας (teilar.gr) λειτουργεί μόνο μέσα από το δίκτυο του campus – από το σπίτι δεν λειτουργεί!

- Επειδή υπάρχουν διαφοροποιήσεις ανάλογα και με το λειτουργικό σύστημα, δείτε τις επίσημες οδηγίες εδώ: http://php.net/manual/en/mail.configuration.php

Ζ. Ανέβασμα Αρχείων στο Server, από τον Client

- Μπορεί να γίνει με 2 τρόπους:
  - Το αρχείο αποθηκεύεται **στο file system του web server** και κάποια μεταδεδομένα (π.χ. όνομα αρχείου, τίτλος, σχόλια, μέγεθος, κτλ) στην database.
    - Πρέπει να επιτρέπεται από τις ρυθμίσεις στο php.ini .
    - Πρέπει να λάβουμε υπόψη το θέμα μεγέθους του αρχείου.
  - Αρχείο και μεταδεδομένα αποθηκεύεται **σε κάποιον πίνακα (table) της database** του site. Το αρχείο αποθηκεύεται σε πεδίο (field) τύπου BLOB (Binary Large OBject).
    - Δεν είναι ιδιαίτερα κομψή λύση καθώς:
      - ✓ τα Binary αρχεία είναι συνήθως μεγάλα σε μέγεθος και "φορτώνουμε" την database
      - ✓ απαιτεί περισσότερη εργασία σε επίπεδο κώδικα (κυρίως στην ανακατασκευή του αρχείου όταν κάποιος ζητάει να το "κατεβάσει".
    - Μερικές φορές όμως είναι "επιβεβλημένη" λύση:
      - ✓ π.χ. αποθήκευση PDF ως BLOB σε Microsoft SQL Server για αξιοποίηση της υποδομής indexing (ευρετηρίου) του SQL server ώστε να δίνεται δυνατότητα αναζήτησης μέσα στο περιεχόμενο των αρχείων PDF (full-text-search) ή για λόγους ασφάλειας.
- Και στην δύο περιπτώσεις, στην μεριά του client χρειαζόμαστε μια κατάλληλα στημένη φόρμα που θα επιτρέψει στον χρήστη να κάνει upload.
  - Βασική ιδιότητα φόρμας: `<form ... enctype="multipart/form-data" ... >`
  - Βασικό στοιχείο μέσα στη φόρμα: `<input ... type="file" ... />`
    - δίνει στον χρήστη της σελίδας δυνατότητα browsing στο file system του υπολογιστή του.

Διακίνηση Αρχείων Μεταξύ Client-Server

Τι περιλαμβάνει;

- Ρυθμίσεις σε PHP
- Υποδομή για τον client (κατάλληλη form)
- Υποδοχή αρχείου στον server
  - Που θα αποθηκευτεί;
- Ανάκτηση αποθηκευμένων αρχείων
- MIME types για αρχεία

Δείτε τα επόμενα slides (`06-PHP-&-MySQL-files.pdf`) καθώς και το παράδειγμα σε εργαστηριακή άσκηση.