Databases Connectivity

Introduction

- Most applications involve with using some sort of a data storage container. Usually, that container is a database.
- PHP enables us to work with different types of databases, mostly relational in their nature.

Tables Model

id	first_name	last_name	average
1239187	Moshe	Israeli	88
4321233	Tami	Tintin	98
2312232	Jane	Speedo	87
3241232	Goni	Mardo	78

The SQL SELECT Statement

- The SELECT statement is used to select data from a database. The result of the SELECT statement is a result table, also known as the result set.

```
SELECT column_name, column_name...
FROM table_name
```

```
SELECT * FROM  table_name
```

The WHERE Clause

- The WHERE clause is used to extract only those records that fulfill a specific criteria.

```
SELECT column_name, column_name...
FROM table_name
WHERE column_name operator value
```

The ORDER BY Keyword

- The ORDER BY keyword is used to sort the result set by a specific column.

```
SELECT column_name, column_name...
FROM table_name
WHERE column_name operator value
ORDER BY column_name
```

The DESC Keyword

- Adding 'DESC' keyword we can get the data sorted in a backward order.

```
SELECT column_name, column_name...
FROM table_name
WHERE column_name operator value
ORDER BY column_name DESC
```

The INSERT INTO Statement

- The INSERT INTO statement is used to insert new rows into our tables.

```
INSERT INTO table_name
VALUES (value1, value2, ...)
```

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...)
```

The UPDATE Statement

- The UPDATE statement is used to update existing records in a table.

```
UPDATE table_name
SET column1=value1, column2=value2,...
WHERE column_name=specific_value
```

The DELETE Statement

- The DELETE statement is used to delete existing records in a table.

```
DELETE FROM table_name
WHERE column_name = spcific_value
```

Join Statement

- Using '`join`' we can create a single record set based on data combined from multiple tables.
- The 'join' sql statement creates a link between two tables based on a common set of columns (keys).
- There are two types of join statements:

  Inner Join

  Outer Join

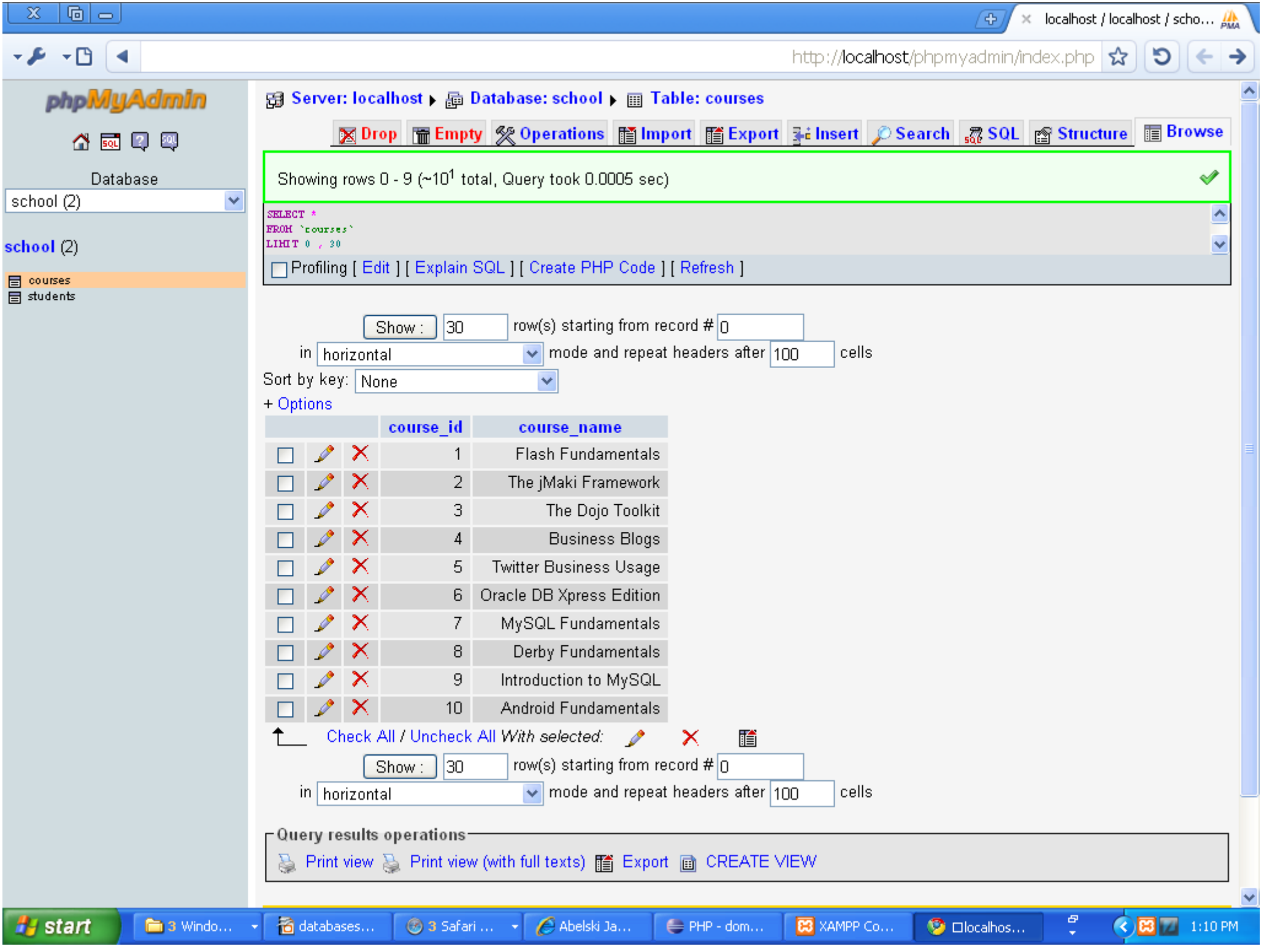
Inner Joins

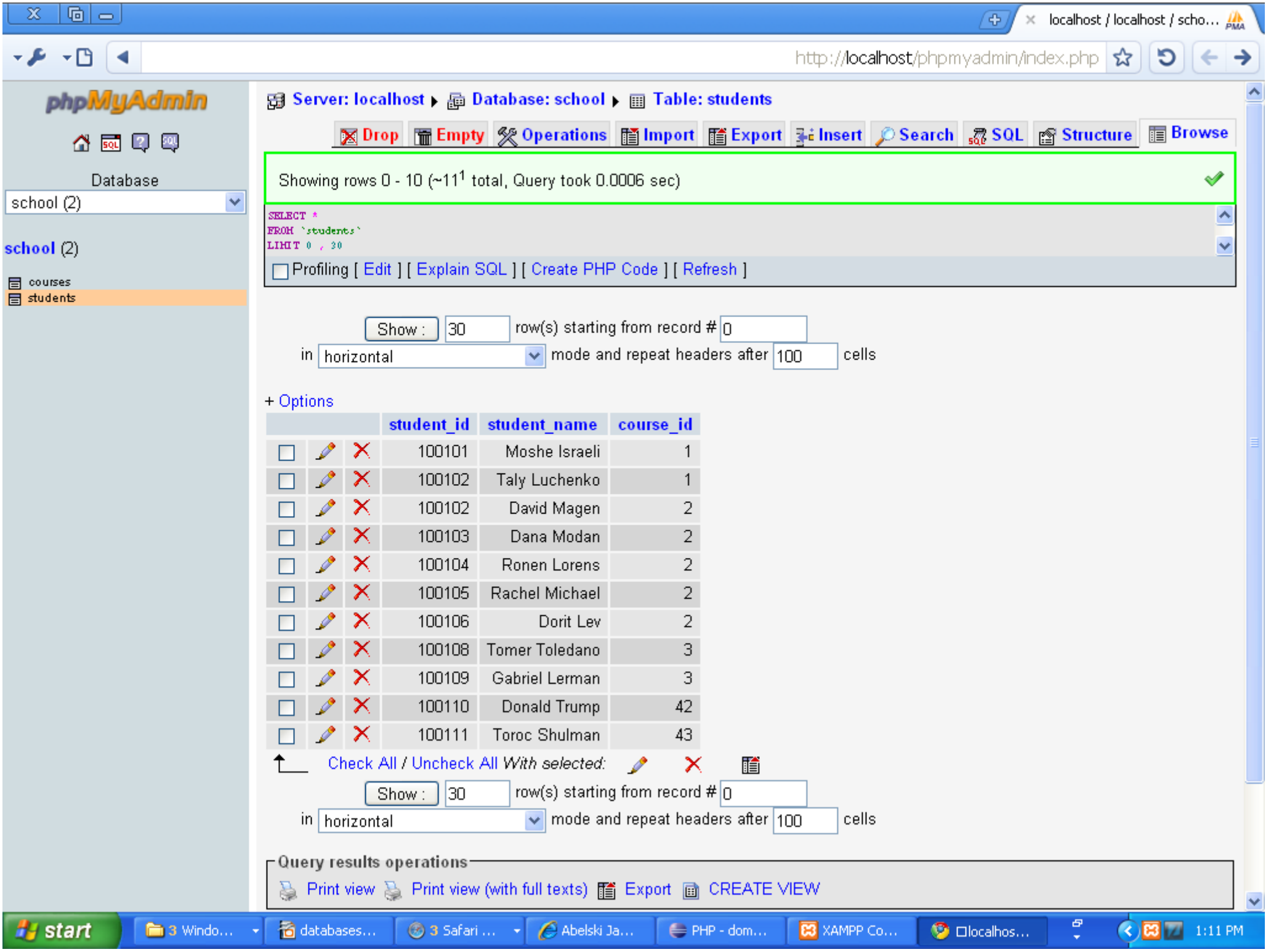
- The inner join returns rows from both tables only if it succeeds to find keys from both tables that satisfy the join condition.

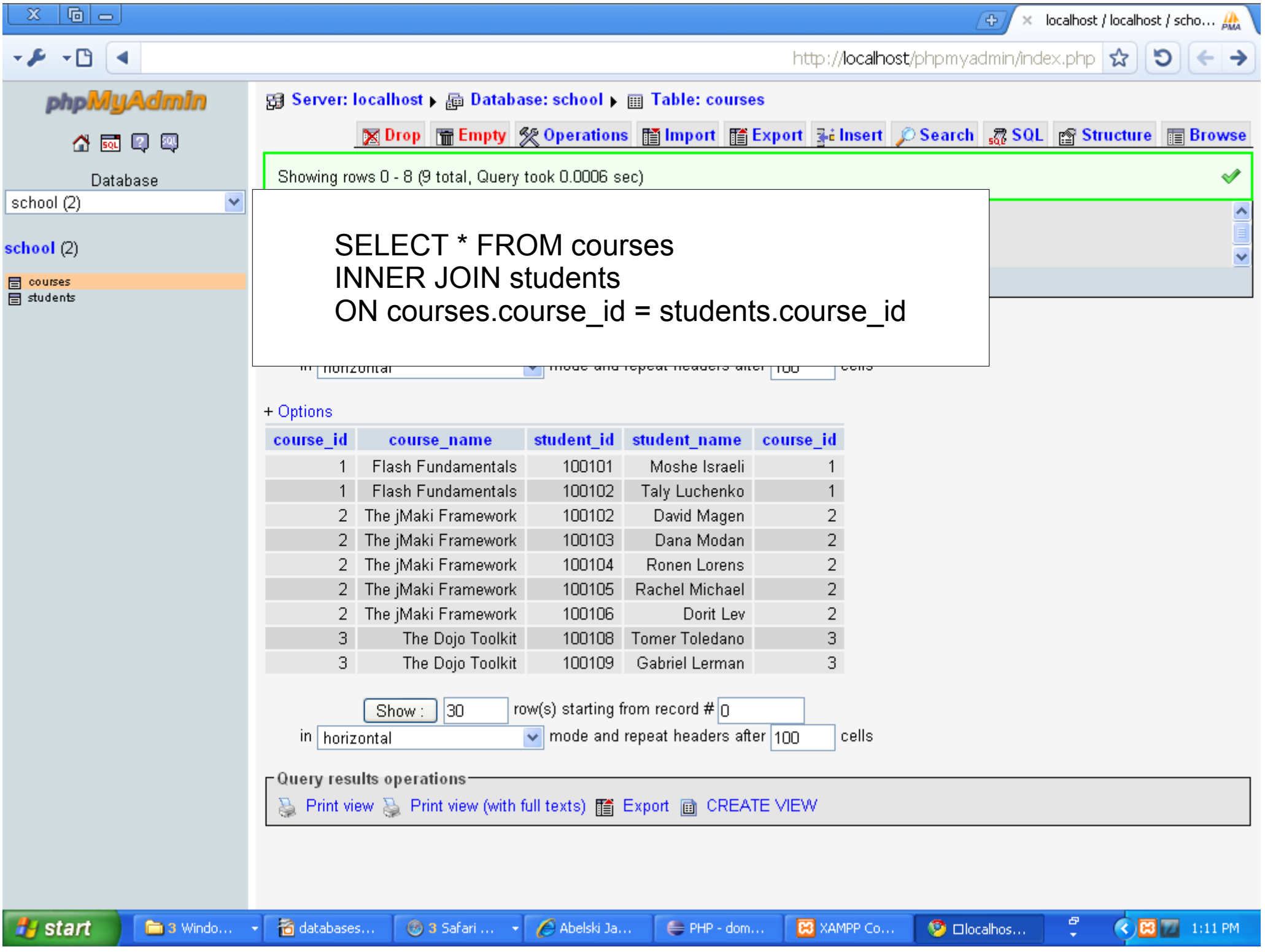
```
SELECT * FROM courses INNER JOIN students
ON courses.course_id = students.course_id
```

In this code sample a link is created between the 'students' and the 'courses' tables. This query returns the details of those courses have registered students.

- Inner join statements work well with assertive conditions only. Trying a negative condition (e.g. <>) often returns strange results.


localhost / localhost / scho...
http://localhost/phpmyadmin/index.php
phpMyAdmin
Database
school (2)
school (2)
courses
students
Server: localhost ▸ Database: school ▸ Table: courses
Drop Empty Operations Import Export Insert Search SQL Structure Browse
Showing rows 0 - 9 (~10[1] total, Query took 0.0005 sec)
SELECT *
FROM `courses`
LIMIT 0 , 30
Profiling [Edit] [Explain SQL] [Create PHP Code] [Refresh]
Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
Sort by key: None
+ Options
course_id course_name
1 Flash Fundamentals
2 The jMaki Framework
3 The Dojo Toolkit
4 Business Blogs
5 Twitter Business Usage
6 Oracle DB Xpress Edition
7 MySQL Fundamentals
8 Derby Fundamentals
9 Introduction to MySQL
10 Android Fundamentals
Check All / Uncheck All With selected:
Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
Query results operations
Print view Print view (with full texts) Export CREATE VIEW
start
3 Windo...
databases...
3 Safari ...
Abelski Ja...
PHP - dom...
XAMPP Co...
localhos...
1:10 PM


localhost / localhost / scho...
http://localhost/phpmyadmin/index.php
phpMyAdmin
Database
school (2)
school (2)
courses
students
Server: localhost ▸ Database: school ▸ Table: students
Drop Empty Operations Import Export Insert Search SQL Structure Browse
Showing rows 0 - 10 (~11[1] total, Query took 0.0006 sec)
SELECT *
FROM `students`
LIMIT 0 , 30
Profiling [Edit] [Explain SQL] [Create PHP Code] [Refresh]
Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
+ Options
student_id student_name course_id
100101 Moshe Israeli 1
100102 Taly Luchenko 1
100102 David Magen 2
100103 Dana Modan 2
100104 Ronen Lorens 2
100105 Rachel Michael 2
100106 Dorit Lev 2
100108 Tomer Toledano 3
100109 Gabriel Lerman 3
100110 Donald Trump 42
100111 Toroc Shulman 43
Check All / Uncheck All With selected:
Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
Query results operations
Print view Print view (with full texts) Export CREATE VIEW
start
3 Windo...
databases...
3 Safari ...
Abelski Ja...
PHP - dom...
XAMPP Co...
localhos...
1:11 PM

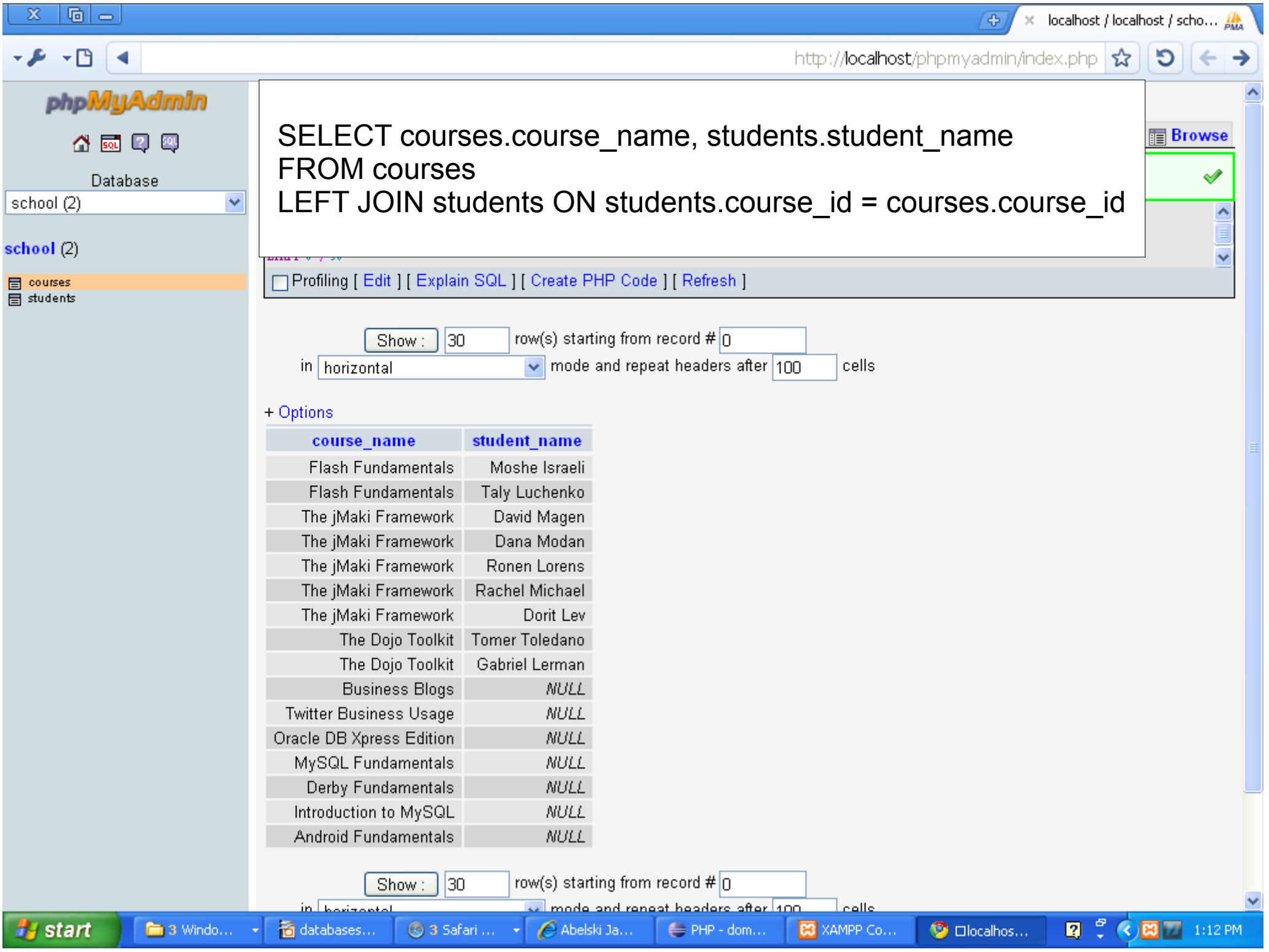

localhost / localhost / scho...
http://localhost/phpmyadmin/index.php
phpMyAdmin
Database
school (2)
school (2)
courses
students
Server: localhost ▸ Database: school ▸ Table: courses
Drop Empty Operations Import Export Insert Search SQL Structure Browse
Showing rows 0 - 8 (9 total, Query took 0.0006 sec)
SELECT * FROM courses
INNER JOIN students
ON courses.course_id = students.course_id
+ Options
course_id course_name student_id student_name course_id
1 Flash Fundamentals 100101 Moshe Israeli 1
1 Flash Fundamentals 100102 Taly Luchenko 1
2 The jMaki Framework 100102 David Magen 2
2 The jMaki Framework 100103 Dana Modan 2
2 The jMaki Framework 100104 Ronen Lorens 2
2 The jMaki Framework 100105 Rachel Michael 2
2 The jMaki Framework 100106 Dorit Lev 2
3 The Dojo Toolkit 100108 Tomer Toledano 3
3 The Dojo Toolkit 100109 Gabriel Lerman 3
Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
Query results operations
Print view Print view (with full texts) Export CREATE VIEW
start
1:11 PM

Outer Joins

- Unlike the inner join statements that restrict the results returned to those that match records in both tables, when using the outer join statements we get all records from one of the two tables only, depending on whether we queried using 'LEFT JOIN' or ' RIGHT JOIN'.

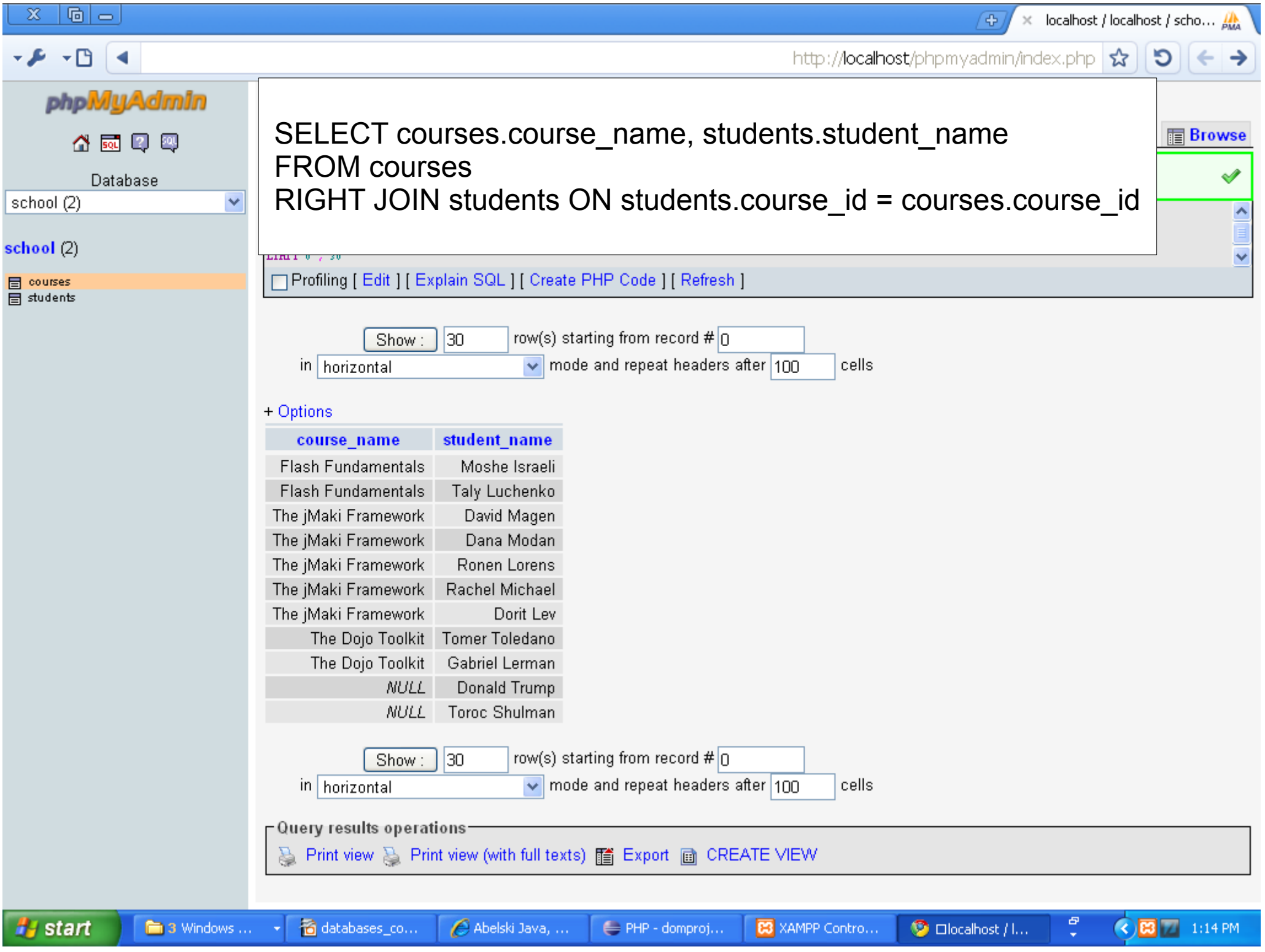
Outer Joins

- Left Joins are a type of outer join in which every record in the left table that matches the WHERE clause (if exists) will be returned regardless whether a match takes place in the ON clause of the right table.


localhost / localhost / scho...
http://localhost/phpmyadmin/index.php
phpMyAdmin
Database
school (2)
school (2)
courses
students
SELECT courses.course_name, students.student_name
FROM courses
LEFT JOIN students ON students.course_id = courses.course_id
Browse
Profiling [Edit] [Explain SQL] [Create PHP Code] [Refresh]
Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
+ Options
course_name | student_name
Flash Fundamentals | Moshe Israeli
Flash Fundamentals | Taly Luchenko
The jMaki Framework | David Magen
The jMaki Framework | Dana Modan
The jMaki Framework | Ronen Lorens
The jMaki Framework | Rachel Michael
The jMaki Framework | Dorit Lev
The Dojo Toolkit | Tomer Toledano
The Dojo Toolkit | Gabriel Lerman
Business Blogs | NULL
Twitter Business Usage | NULL
Oracle DB Xpress Edition | NULL
MySQL Fundamentals | NULL
Derby Fundamentals | NULL
Introduction to MySQL | NULL
Android Fundamentals | NULL
Show : 30 row(s) starting from record # 0
start
1:12 PM

Outer Joins

- Right Joins are a type of outer join in which every record in the right table that matches the WHERE clause (if exists) will be returned regardless whether a match takes place in the ON clause of the left table.


localhost / localhost / scho...
http://localhost/phpmyadmin/index.php
phpMyAdmin
Database
school (2)
school (2)
courses
students
SELECT courses.course_name, students.student_name
FROM courses
RIGHT JOIN students ON students.course_id = courses.course_id
Browse
Profiling [Edit] [Explain SQL] [Create PHP Code] [Refresh]
Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
+ Options
course_name
student_name
Flash Fundamentals
Moshe Israeli
Flash Fundamentals
Taly Luchenko
The jMaki Framework
David Magen
The jMaki Framework
Dana Modan
The jMaki Framework
Ronen Lorens
The jMaki Framework
Rachel Michael
The jMaki Framework
Dorit Lev
The Dojo Toolkit
Tomer Toledano
The Dojo Toolkit
Gabriel Lerman
NULL
Donald Trump
NULL
Toroc Shulman
Show : 30 row(s) starting from record # 0
in horizontal mode and repeat headers after 100 cells
Query results operations
Print view
Print view (with full texts)
Export
CREATE VIEW
start
3 Windows ...
databases_co...
Abelski Java, ...
PHP - domproj...
XAMPP Contro...
localhost / l...
1:14 PM

Transactions

- Transaction is a group of operations that are committed (or discarded) as if they were an atomic unit.

  One of the best examples that assists understanding the Transactions concept is a bank transaction through which money is being transferred from one account to another. Such operation involves with updating more than a few tables. We want this series of operations to be considered as an atomic one.

Transactions

- Databases that support transactions are considered to be ACID compliant for offering atomicity (A), consistency (C), isolation (I) and durability (D).

  Atomicity (A)

  Either all tasks of the transaction are performed or none of them.

  Consistency (C)

  The database remains in a consistent state before the transaction starts and after it ends, whether successfully or not.

Transactions

Isolation (I)

This refers to the requirement that other operations cannot access or see the data in the intermediate state during the transaction.

Durability (D)

Once the user has been notified of the success the transaction will persist and won't be undone.

Transactions

- Working directly with the data base server we start a transaction using 'START TRANSACITON'. We complete it using 'COMMIT'. We undo using 'ROLLBACK'.

```
START TRANSACTION;
DELETE FROM courses WHERE course_id = 2;
UPDATE courses SET course_id=999 WHERE course_id=1;
ROLLBACK;
```

Indices

- Most databases were designed to allow better performance when reading the data while having certain amount of efficiency be sacrificed when data is written.
- This efficiency is achieved by using the indices mechanism. Indices should be created on those columns that most likely will be used more than others in your queries.

Indices

- Too many indices will cause extra work and damage their effectiveness. On the other hand, a relatively small number of indices might cause the engine more often to ignore them.

```
CREATE INDEX student_id ON students (id);
```

- The primary key is a sort of index. Specifying a column as a primary key functions as index.

PHP & MySQL

- PHP supports MySQL since PHP 2.0.
- The PHP and MySQL technologies are highly related with each other allowing to use MySQL within PHP code in a natural way.

MySQLi

- With the release of PHP 5.0 a new MySQL extension was released, known as MySQL Improved (AKA 'MySQLi').

  This new extension takes advantage of many of the new features introduced by MySQL (e.g. prepared statements, advanced connection options, security improvements and others.). In addition, MySQLi provides a native object oriented interface allowing us to extend it in accordance with our needs.

- MySQL and MySQLi extensions share many similarities.

  That allows PHP programmers to smoothly upgrade their code to use the MySQLi extension instead of the well known procedural oriented.

MySQLi Major Enhancements

- MySQLi allows us to use many of the new capabilities available with the recently released MySQL extensions.
- MySQLi allows us to enjoy the following enhancements:

  Prepared Statements

  Object Oriented Programming Interface

  Support for Embedded Server

  Enhanced Debugging Capabilities

  Support for Transactions

Users Privileges

- As with any other programming language interface MySQL supports, a PHP script connecting with MySQL must first connect to the MySQL server and must select the database it wants to interact with.
- Doing so as well as interacting with the database performing various SQL statements can be carried out only by a user that has the required privileges.

  When the user connects with the database those privileges are communicated and verified by the MySQL server.

Connecting MySQL Database

- Interacting with MySQL database starts with setting up the connection and ends with closing it.
- The first step will be instantiating mysqli class with the parameters describing the connection we want to set.

```
$mysqli = new mysqli(“server_name”,”username”,”password”,”db_name”);
```

Connecting MySQL Database

- We can alternatively instantiate mysqli class with an empty parameters constructor and set the connection parameters calling the connect() method.

```
$mysqli = new mysqli();
$mysqli -> connect("server_name","username","password","db_name");
```

Connecting MySQL Database

- Assuming we have already created the required mysqli connection object and we now want to switch to another database we can call the select_db method.

```
$mysqli = new mysqli();
$mysqli -> connect("server_name","username","password","db_name");
$mysqli -> select_db("other_db_name");
```

MySQL DB Connection Closing

- Once a database was successfully selected we can easily execute various queries, such as select, insert, update etc.
- Once a PHP script ends all open databases connections are automatically closed and their resources are automatically freed.
- Calling close() terminates a database connection.

```
$mysqli -> close();
```

Simple Queries

- Sending a query to the database is done by calling the `query()` method.

  This method was declared within the mysqli class. Therefore, we should call it using the mysqli object.

```
class mysqli
{
    ...
    mixed query(string query [, int resultmode])
    ...
}
```

Simple Queries

- Calling the `query` method we can modify its behavior passing one of the two available values for the `resultmode` parameter:

`MYSQLI_STORE_RESULT`

The result will be returned as a buffered set. The entire set will be available for an immediate browsing. This is the default setting. This setting has a performance price (it requires more memory).

`MYSQLI_USE_RESULT`

The result will be returned as a non buffered set. The set will be retrieved from the server on as-needed base. When the set is big we will enjoy a better performance.

Simple Query Sample

```
<?php


$mysqli = new mysqli("127.0.0.1","iuser","ipassword",
    "store_db");

$query = "SELECT product, id FROM products ORDER BY product";

$result = $mysqli->query($query,MYSQLI_STORE_RESULT);

while(list($name,$id) = $result->fetch_row())
{
    printf("%s,%s<br>",$name,$id);
}

$mysqli->close();

?>
```

MySQLi Prepared Statements

- A prepared statement is an object that represents SQL statement that is partially compiled.
- The prepared statement includes '?' marks representing missing values required to complete its execution.
- We can execute a prepared statement multiple times, each time with different values instead of the '?' marks.
- Using prepared statements is an excellent practice against SQL injections.

MySQLi Prepared Statements

- We can create a `MySQLi_STMT` object by calling the `$mysqli->prepare()` method.

```
$stmt = $mysqli->prepare("INSERT INTO students VALUES (?, ?, ?)");
```

- We can bind each one of the '?' marks with a specific variable using the `$stmt->bind_param()` method.

  Once the '?' are binded with specific variables we can execute the prepared statement, again and again, each time having other values within the binded variables.

MySQLi Prepared Statements

```
...
$stmt = $mysqli->prepare("INSERT INTO students VALUES (?, ?, ?, ?)");
$stmt->bind_param('issd',$id,$first_name,$last_name,$average);
$id = 123423123;
$first_name = 'Haim';
$last_name = 'Moshe';
$average = 88.2;
$stmt->execute();
...
```

MySQLi Prepared Statements

- Working with prepared statements it is also possible to bind the results we get from querying a database.
- We can bind the result with specific variables and use them when iterating the rows the result includes.
- Binding the result with specific variables is done by calling the `$stmt->bind_result()` method.

MySQLi Prepared Statements

```
...
$stmt = $mysqli->prepare("SELECT name,average FROM students");
$stmt->bind_result($name,$avg);
while($stmt->fetch())
{
    printf("%s %s\n",$name,$avg);
}
$stmt->close();
...
```

MySQLi Transactions

- The MySQLi extension implements the transactions functionality using the `commit()` and `rollback()` methods.
- By default, mysqli works in the auto commit mode, meaning that the each database statement is committed immediately. Calling the `autocommit()` method we can change that.
- Calling `commit()` will complete the execution of the current transaction. If it fails the returned value is FALSE. We can use that to call the `rollback()` method.

MySQLi Transactions

```
...
$mysqli->autocommit(FALSE);
$mysqli->query("INSERT INTO students (id,name) VALUES (12312,'John')");
$mysqli->query("INSERT INTO courses (id,title) VALUES (221,'Math')");
if(!mysqli->commit())
{
    $mysqli->rollback();
}
...
```

The 'real_escape_string' Method

- This method escapes special characters a string includes so we could use it as part of our SQL statement.

```
...
$statement = "SELECT * FROM books WHERE title='"
    .$mysqli->real_escape_string($title)."'";
...
```

The 'real_escape_string' Method

```
<?php
$mysqli = new mysqli("localhost", "my_user", "my_password", "school");

if (mysqli->connect_errno())
{
    printf("Connect failed: %s\n", mysqli->connect_error());
    exit();
}

$mysqli->query("CREATE TABLE students LIKE stud");

$name = "Big Mose's Friend";

/* this query will fail, cause we didn't escape $name */
if (!$mysqli->query("INSERT INTO students (nickname) VALUES ('$name')"))
{
    printf("Error: %s\n", $mysqli->sqlstate);
}
```

The 'real_escape_string' Method

```
$name = $mysqli->real_escape_string($name);

/* this query with escaped $name will work */
if ($mysqli->query("INSERT into students (nickname) VALUES ('$name')"))
{
    printf("%d Row inserted.\n", $mysqli->affected_rows);
}

$mysqli->close();
?>
```

The 'ctype_alpha' Function

- This function receives a string and returns true if each one of its characters is an alphabetic character from current locale.
- This function is useful for preventing SQL injections when filtering input coming from the user over the web.

```
...
$title='';
if(ctype_alpha($_GET['title']))
{
    $title = $_GET['title'];
}
...
```

MySQL Native Driver

- The new MySQL native driver provides with an improved persistent connection, the function `mysqli_fetch_all()` and with performance statistics functions we can use in order to get more info about the performance of our application.
- In addition, as of PHP 5.3 MySQL native driver supports SSL and as of PHP 5.2 MySQL native driver supports the compressed client server protocol.

PHP Data Objects

- Since PHP 5.1, the PDO is included within the standard distribution of the PHP environment.
- The PDO (PHP Data Objects) allows us to use one unified interface for accessing all databases. The same code. Without changes. One single interface.

PHP Data Objects

- PDO provides a data access abstraction layer only. Regardless of which database we are using we will use the same functions. Working with PDO we still need to install the specific database PDO driver required for our work.
- Many database drivers already exist for PDO, including drivers that enable to access Microsoft SQL Server, MySQL, Oracle, PostgreSQL and ODBC.

PHP Data Objects

- Working with PDO switching to another database there is a need to pay attention to the fact that specific SQL statements that work on one database may not work on another.

PHP Data Objects

```
try
{
    $ob = new PDO('mysql:host=localhost;dbname=school,
              'my_user_name','my_password');
    $ob->setAttribute(  PDO::ATTR_EMULATE_PREPARES,
                        TRUE);
    $ob->setAttribute(  PDO::ATTR_ERRMODE,
                        PDO::ERRMODE_EXCEPTION);
    ...
}
catch(PDOException $e) {...}
```

PHP Data Objects

- The querying operation should be performed using the 'PDO::query()' method. It returns a PDOStatement object. Working with a PDOStatment object we can access each one of the columns as a property of the object.

```
...
$results = $ob->query($statement);
foreach($results as $row)
{
    echo $row['course_name'].' '.$row['course_id'];
}
...
```

PHP Data Objects

- The default fetch mode when calling the query method is PDO::FETCH_BOTH. Calling `PDO::query()` we get an array that its values can be accessed both using the numeric indexes and the associative keys.
- Calling the setFetchMode() method on our PDO object we can change the default fetch mode.

  Possible values we can pass this method when calling it include the following: PDO::FETCH_OBJ, PDO:FETCH_NUM, PDO:FETCH_BOTH and others.

PHP Data Objects

- The inserting, updating and deleting operations should be performed using the 'PDO::exec()' method. It returns the number of rows affected.

```
...
$num_affected_rows = $ob->exec('DELETE FROM courses');
echo 'number of rows being deleted is '.$num_affected_rows;
...
```

PHP Data Objects Sample

```
<?php
try
{
    $ob = new PDO('mysql:host=127.0.0.1;dbname=mystore',
        'iuser','ipassword');
    $ob->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
    $statement = "SELECT product, id FROM inventory ORDER BY product";
    $results = $ob->query($statement);
    printf("<table>");
    foreach($results as $row)
    {
        printf("<tr><td>%s</td><td>%s</td></tr>",
            $row['product'],$row['id']);
    }
    printf("</table>");
}
catch(Exception $e)
{
    echo $e->getMessage();
}
?>
```

PHP ORM (Object Relational Mapping)

- Object Relational Mapping is a relatively new approach for working with a database interacting with objects instead of interacting with the database tables.

  Ready to use frameworks are available to ease our work with databases. Propel is one of them.

- Instead of mapping the database tables with objects manually we can use ready-to-use frameworks that do the work for us.

  These frameworks may include built-in validation tests as well as any other operation common for mapping objects with a database.

Propel Framework

- Propel is a PHP framework that provides an ORM solution for objects persistence and querying, allowing you to work with a database indirectly via objects.
- Propel is based on Torque, an open source ORM project developed in Java and maintained by the Apache community.

Propel Framework

- Working with Propel, executing CRUD (Create, Read, Update & Delete) operations as well as validating forms becomes fairly easy. Propel can even generate the required SQL schema.

MySQL Native Driver

- The MySQL native driver replaces the MySQL client library (libmysql). The MySQL native driver is part of the official PHP sources as of PHP 5.3.
- As of PHP 5.3 the communication with the MySQL database, whether we use PDO, MySQL or MySQLi is done using the native driver.
- The MySQL native driver is written in C as an extension to the PHP execution environment.

MySQL Native Driver

- The new MySQL native driver provides with an improved persistent connection, the function `mysqli_fetch_all()` and with performance statistics functions we can use in order to get more info about the performance of our application.
- In addition, as of PHP 5.3 MySQL native driver supports SSL and as of PHP 5.2 MySQL native driver supports the compressed client server protocol.

MySQL Native Driver

- When developing for the windows platform the new MySQL native driver is enabled by default. When developing for linux there PHP needs to be built with the MySQL native driver support.