jinyPHP 2권 - 003. 연결

**03. 연결**

====================

데이터베이스 시스템은 복잡한 데이터를 운영관리하기에 필요한 다양한 처리 기능들을 제공해 줍니다. 그리고 데이터를 저장하고 조작할 수 있는 SQL언어와 콘솔 인터페이스도 지원을 하고 있습니다. 하지만 단순히 데이터의 저장과 관리는 정보 데이터를 100% 효율적으로 활용을 한다고는 볼 수 없습니다.

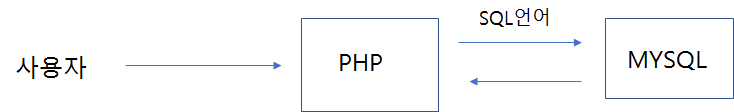
축적 및 보관된 데이터는 별도의 프로그램 및 정보 서비스를 통하여 더 가치가 있는 데이터로 거듭날 것입니다. 그래서 대부분의 데이터베이스 시스템은 외부 프로그램들과 접속하여 데이터를 처리를 할 수 있도록 다양한 연결 인터페이스를 제공하고 있습니다.

**03.1 DB연동**

====================

MYSQL은 가장 많이 사용을 하는 관계 데이터 베이스 시스템입니다. 또한, 수많은 언어들과 연결 인터페이스를 통하여 데이터 처리 작업을 할 수 있습니다. 그중 MYSQL와 가장 밀접하게 긴 시간동안 궁합을 맞추어 왔던 언어가 바로 PHP 입니다. PHP는 가장 많은 MYSQL 접속처리 환경과 기능들을 보유하고 있습니다. MYSQL 또한 웹활성화와 PHP언어의 성장세의 힘입어 지금의 유명세를 얻는데 많은 도움을 받았습니다.

PHP를 이용한 데이터베이스 연동 프로그램들은 3단계의 접속 및 처리 구조를 가지고 있습니다. 먼저 사용자는 PHP 프로그램으로 접속을 합니다.



두번째로 접속된 PHP는 데이터를 처리하기 위해서 MYSQL 시스템에 연결하며, 데이터 처리를 위한 SQL 명령문을 전달합니다. MYSQL은 PHP로부터 전달받은 SQL 명령문을 해석합니다. 만일 반환되는 데이터가 있을 경우에, 데이터를 PHP로 전달합니다. 세번째로 PHP는 MYSQL과 연동 및 처리가 완료된 데이터를 사용자에게 보기 좋게 출력을 가공하여 사용자에게 화면을 반환합니다.

기존의 MYSQL의 콘솔작업들은 직접 DB 시스템에 접속하여 SQL 실행 명령을 처리하는 것이었습니다. 하지만 이러한 직접적인 데이터처리는 일반적으로 관리자가 오류 데이터를 검색하거나 직접 수정을 할 때를 제외하고는 사용하지 않습니다. 데이터베이스 작업들은 PHP와 같은 외부 언어들을 통하여 프로그래밍 되고, 접근하여 데이터를 처리 반환합니다.

데이터베이스의 SQL 명령어 또한 PHP를 통하여 동적 생성하게 하고, 이를 MYSQL 데이터베이스에 전달하여 실행을 하게 됩니다. 이러한 연동작업들을 DB프로그래밍이라고 합니다.

**03.2 연결접속**

====================

PHP언어는 타 언어와 달리 MYSQL과 DB연동을 하기위한 다양한 접속방법을 제공합니다. 무려 3가지의 접속방법을 제공합니다. PHP 5.x 이상에서는 3가지 방법으로 코드상 MYSQL에 접속을 할 수 있는데, 객체지향 방식 과 절차적방식 PDO 방식입니다.

* Mysql
* MySQLi extension ("i"는 향샹된 기능이라는 의미입니다.)
* PDO (PHP Data Objects)

하지만 위 3가지중 보안상의 이유로 PHP 7.x 에서 기존에 사용하던 mysql\_\*\*\* 함수들은 제거되었습니다. 따라서 최신 버전의 PHP에서는 2가지 방법만 사용이 가능하다고 보시면 됩니다.

MYSQL과 접속한 PHP는 코드상에서 SQL 쿼리 명령을 동적으로 생성을 하여 DB 시스템에 전송을 할 수 있습니다. 또한 전송된 데이터를 배열 또는 객체 형태로 읽어 올 수 있습니다.

**03.2.1 MYSQLi**

====================

기존에 가장 많이 사용하는 DB 접속 함수로는 mysql\_\*\*\* 이 있었습니다. Mysql\_ 함수들은 쿼리의 실행화 결과 반환 값을 배열 형태로 받습니다. 하지만 이 함수들은 시간이 지날수록 보완적인 문제점들이 하나씩 발견되기 시작되었습니다.

기존 mysql\_\*\*\* 함수들의 보완적인 문제점과 보다 향상된 기능 처리를 위해서 고안된 방법이 객체지향형 mysqli\_\*\*\* 클래스 입니다. mysqli\_\*\*\* 클래스는 MySql과 관련된 기능들을 클래스 객체화 하여 접근 사용할 수 있습니다. 또한 반환 값도 객체 형태로 반환 받아 처리를 할 수 있습니다.

**MySQLi 설치 :** 도우/리눅스 모두 PHP 5.x 이상에서는 대부분 MySQLi 확장기능은 자동적으로 설치됩니다. 보다 자세한 내용은 http://php.net/manual/en/mysqli.installation.php 에서 확인이 가능합니다.

**03.2.2 PDO**

====================

PHP의 확장 클래스MYSQLi 만으로도 충분히 모든 데이터를 처리할 수 있습니다. 하지만 이 클래스는 MySql 데이터베이스 시스템을 위주로 개발된 클래스 입니다.

PHP는 MySql 데이터베이스 시스템 이외에도 다양한 데이터베이스 시스템을 이용할 수 있습니다. PHP는 MySql을 통하여서 여러 데이터베이스를 접속하고 처리를 할 수 있는 PDO 라는 방식을 하나 더 제공을 합니다.

PDO 방식은 MYSQL이외에 12가지의 다양한 데이터베이스를 접속 및 코딩을 작성할 수 있습니다. 반면에 MYSQLi는 MYSQL 에 특화된 클래스 입니다. 현재 개발하는 프로젝트가 MYSQL이외에 다양한 데이터베이스를 지원해야 한다면 PDO 방식을 추천합니다. PDO 방식은 간단한 쿼리변경으로 처리를 할 수 있습니다.

**PDO 설치 :** PDO 설치에 관련한 자세한 사항은 http://php.net/manual/en/pdo.installation.php 을 통해서 확인이 가능합니다.

**03.3 PHP 연결**

====================

기존 콘솔의 MySql 접속과 PHP 연동 접속에 대해서 차이점을 한번 살펴 보도록 하겠습니다.

MySQL데이터베이스에 쿼리를 전송하기 위해서는 먼저 데이터베이스 서버에 접속을 하여야 합니다. 기존 콘솔방법으로 접속할때는 **mysql -u계정명 -p패스워드** 처럼 계정명과 패스워드를 입력하였습니다.

하지만 콘솔이 아닌 PHP언어서는 콘솔 클라이언트 프로그램을 사용할 수는 없습니다. PHP 코드상에서는 직접 접속을 위한 클래스 생성, 매소드 인자를 호출하여 DB에 접속을 해야 합니다.

**03.3.1 접속계정**

====================

PHP에서도 MYSQL 데이터베이스에 접속을 하기 위해서는 기본적인 계정정보들이 필요로 합니다. 콘솔상에서는 아이디와 패스워드만 있어도 되었지만, PHP 언어를 통하여 DB에 접속을 하기 위해서는 완전한 접속정보 4가지가 필요로 합니다.

**첫번째 인자값 :** 호스트 주소

첫번째 인자 값으로 mysql 서버 IP접속 정보를 설정합니다. 동일한 서버일때는 localhost를 사용하면 됩니다. 만일 외부의 MYSQL 서버로 접속을 할 때는 도메인명 또는 주소를 입력하면 됩니다.

**두번째 인자값 :** 데이터베이스명 설정

두번째 인자 값으로 mysql의 데이터베이스명을 입력합니다. MYSQL은 다수의 DB를 관리할 수 있습니다. 접속 기본이 되는 데이터베이스 이름을 입력합니다.

**세번째 인자값 :** 사용자 아이디

세번째 인자 값으로 MYSQL 사용자 아이디를 입력합니다. MYSQL은 루트(root)사용자를 포함하여 다수의 사용자 계정을 만들어 서비스를 할 수 있습니다. 생성된 사용자 계정 아이디를 입력하시면 됩니다.

**네번째 인자값 :** 사용자 패스워드

네번째 인자 값으로 MYSQL의 사용자 패스워드를 입력합니다.

**03.3.2 설정값**

====================

MySql 데이터베이스에 접속하기 위한 4개의 인자값을 직접 코드상에서 일일이 적어 주기에는 불편합니다. 접속 정보들은 언제 든지 변경이 될 수 있기 때문입니다. 또한 보완상 직접 작성을 하지도 않습니다.

편리하게 접속정보를 관리하는 방법은 PHP변수에 저장하여 별도의 외부 파일로 분리를 해 놓는 것입니다. 이렇게 저장해 놓은 환결파일은 include 하여 실제적인 PHP 소스에 사입하여 사용하면 편리합니다.

**예제파일) dbconf.php**

|  |
| --- |
| <?php  $mysql\_host = "localhost";  $mysql\_user = "root";  $mysql\_password = "root123";  $mysql\_database = "jiny";  ?> |

위의 예처럼 MYSQL접속에 관련된 값을 변수나 상수로 지정하여 사용을 하시면 편리합니다.

**03.3.3 mysqli 개체지향적**

====================

Mysqli는 PHP의 MYSQL 전용 확장 클래스 입니다. 또한 가장 많이 일반적으로 사용하는 PHP & Mysql 접속 방법입니다.

**예제파일) sql-01.php**

|  |
| --- |
| <?php  include "dbinfo.php";  // ++ Mysqli DB 연결.  $mysqli = new mysqli($mysql\_host, $mysql\_user, $mysql\_password, $mysql\_database);    /\* check connection \*/  if($mysqli->connect\_errno) {  die('DB Connect Error'.$mysqli->connect\_error);  } else {  echo "mysql connect<br>";  }  ?> |

화면출력)

mysql connect

위의 예제는 mysqli 방식을 통하여 MySQL 데이터베이스에 접속하는 방법입니다. 먼저 접속에 필요한 mysqli 인스턴스를 생성해야 합니다. new 키워드를 통하여 mysqli 클래스 타입으로 객체를 생성하면 됩니다.

mysqli 접속 인스턴스 객체를 생성할 때 접속에 필요한 계정정보를 초기 인자값으로 같이 전달 합니다. mysqli 클래스는 함께 전달받은 계정정보로 DB에 접속을 합니다. 만일 접속을 성공하면 정상적인 객체를 반환을 하고, 실패를 하면 $mysqli->connect\_errno 프로퍼티를 true로 설정을 합니다.

항상 DB를 접속할 때 접속의 성공여부를 확인하는 조건문을 삽입하는 것을 추천합니다. 다른 외부적인 환경들로 인하여 DB접속이 실패할 수 도 있기 때문입니다. 접속실패 상태에서 쿼리를 실행하는 것은 또다른 오류를 발생할 수도 있습니다.

**03.3.4 mysqli 절차적**

====================

Mysqli를 클래스 방식으로 인스턴스를 생성하여 사용하는 것 외에 기존에 mysql\_\*\*\* 처럼 절차적인 함수 형태로도 사용을 할 수 있습니다.

**예제파일) sql-02.php**

|  |
| --- |
| <?php  include "dbinfo.php";    // ++ Mysqli DB 절차적 연결  $conn = mysqli\_connect($mysql\_host, $mysql\_user, $mysql\_password, $mysql\_database);    // Check connection  if (!$conn) {  die("Connection failed: " . mysqli\_connect\_error());  } else {  echo "mysql 절차적 connect<br>";  }    mysqli\_close($conn);  ?> |

화면출력)

mysql 절차적 connect

사용법은 mysql\_\*\*\* 기존과 거의 유사하지만 함수의 이름이 mysqli\_\*\*\*로 i 글자가 하나 더 추가된 점이 있습니다.

**03.3.5 PDO**

====================

PDO는 또하나의 PHP의 데이터 접속 방법중의 하나 입니다. PDO는 MySql 이외에 다양한 데이터베이스에 접속하여 처리를 할 수 있다는 장점이 있습니다.

**예제파일) sql-03.php**

|  |
| --- |
| <?php  include "dbinfo.php";    // ++ PDO DB 연결  try {  $conn = new PDO("mysql:host=$mysql\_host;dbname=$mysql\_database", $mysql\_user, $mysql\_password);  // set the PDO error mode to exception  $conn->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  echo "mysql PDO connect";  }  catch (PDOException $e){  echo $sql . "<br>" . $e->getMessage();  }    $conn = null;  ?> |

화면출력)

mysql PDO connect

PDO는 DB 접속 및 처리를 하는데 예외 오류를 발생할 수 있습니다. 따라서 오류로 인하여 시스템이 중단되는 것을 방지 하기 위해서 예외처리 루틴이 try ~ cath{}를 적용하여 사용해야 합니다.

**03.4 연결종료**

====================

PHP가 데이터베이스 처리 작업을 모두 한 후에는 연결한 DB 를 종료해 주어야 합니다. 물론

스크립트가 종료되면 연결된 DB는 자동으로 종료 됩니다. 하지만 정상적인 완료를 위해서 종료처리를 해주도록 합니다.

**MySQLi 객체지향**

|  |
| --- |
| $conn->close(); |

**MySQLi 절차적**

|  |
| --- |
| mysqli\_close($conn); |

**PDO**

|  |
| --- |
| $conn = null; |

DB의 연결 종류는 접속한 방식에 따라서 약간씩 차이가 있습니다.

**03.5 언어셋트**

====================

데이터베이스는 다양한 종류의 데이터를 저장 관리하게 됩니다. 또한, 시스템을 사용하는 환경과 지역 언어별로 특화된 문자셋등이 처리되어야 하는 경우도 발생을 합니다.

MYSQL 환경 설정에서 언어 설정을 할 수도 있지만, 쿼리 명령을 통하여도 언어 설정을 지정할 수 있습니다.

**쿼리문법)**

|  |
| --- |
| SET NAMES UTF8; |

SQL 쿼리를 통하여 문자 인코딩 세트를 변경할 수 있습니다. 예전에는 국가별 언어셋을 설정하여 사용하는 경우가 많았습니다. 한국의 경우 EUC-KR 를 많이 사용했습니다. 하지만 요즘들어 다국어 환경이 일반화 되면서 대부분의 언어셋은 UTF-8 코드셋을 많이 사용합니다.

**03.6 클래스 만들기**

====================

단순하게 SQL언어 문법으로 실습과 결과를 확인하는 것은 실제적인 PHP 연동 코딩을 하기에는 추상적인 면이 있을 수 있습니다. 따라서 이 책에서는 직접 SQL 명령과 이를 적용한 소스코드를 같이 작성을 하여 실제적인 동작 결과를 확인할 수 이도록 구체적인 예제를 많이 작성해 보도록 하겠습니다.

DB연동 작업은 먼저 DB 접속연결과 SQL명령어 실행 두개의 코드로 구분을 할 수 있습니다. 매번 실행 코드에 DB를 접속하는 설명과 지면을 낭비하는 것은 불필요 합니다. 따라서 DB접속 및 처리를 위한 별도의 클래스 예제를 하나 만들어서 이를 재사용 하고자 합니다.

그리고 각각의 예제들은 관련 클래스 메서드를 추가하여 사용으로 나중에 필요한 DB처리 클래스 하나를 완성해 보는 것으로 합니다.

**예제파일) mysql.class.php**

|  |
| --- |
| <?php  class JinyMysql  {  public $dbcon;    // 데이터베이스 초기 연결  function \_\_construct()  {    $this->dbcon = mysqli\_connect(  $GLOBALS['mysql\_host'],  $GLOBALS['mysql\_user'],  $GLOBALS['mysql\_password'],  $GLOBALS['mysql\_database']);  if (mysqli\_connect\_errno()) {  printf("Connect failed: %s\n", mysqli\_connect\_error());  exit();  } else {    $this->msgEcho("mysql connected!");  mysqli\_query($this->dbcon, "set names utf8");    }  }    public function msgEcho($msg)  {  echo $msg."<br>";  }  }  ?> |

위의 예제 클래스는 앞으로 확장해 나갈 기본 클래스 뼈대 입니다. 이곳에 추가적으로 기능을 하나씩 덧 붙여 가면서 실습해 보도록 하겠습니다.

예제 클래스를 삽입하여 DB를 접속하고 테스트 해보는 예제코드를 만들어 보겠습니다.

**예제파일) sql-04.php**

|  |
| --- |
| <?php  include "dbinfo.php";  include "mysql.class.php";  // ++ Mysqli DB 연결.  $db = new JinyMysql();  ?> |

화면출력)

mysql connected!

처음 DB접속에 대한 환경 소스파일을 삽입을 합니다. 두번째 줄에는 작성한 클래스 소스 파일을 삽입을 합니다.

실제적인 DB 접속 인스턴스를 new 키워드를 통하여 생성합니다.

**03.7 mysqli 함수**

====================

mysqli 는 MySql을 접속하고 처리할 수 있는 다양한 PHP 내부 함수들을 제공하고 있습니다. 몇가지 주요한 함수들만 학습하고 넘어가도록 하겠습니다.

**03.7.1 쿼리전송**

====================

mysqli\_query() 함수는 입력된 쿼리값을 MYSQL로 전송하여 실행을 합니다. 쿼리를 생성하고 전송할 때 가장 많이 쓰는 함수일 것입니다.

|  |
| --- |
| mysqli\_query($this->dbcon, "set names utf8"); |

**03.7.2 반환갯수**

====================

SELECT 쿼리와 같이 반환값이 있는 경우에는 mysqli\_num\_rows() 함수를 통하여 데이터의 반환 데이터 갯수를 읽어 올 수 있습니다.

|  |
| --- |
| mysqli\_num\_rows($result) |

다수의 데이터를 반환을 받을 때 for문관 같은 반복문을 사용합니다. 반복문 사용시 반복 횟수를 지정하기 위해서 이 함수는 본격적으로 반환 데이터를 읽기 전에 한번 호출하여 데이터의 개수 유무를 파악하는 것이 좋습니다.

**03.7.3 데이터읽기**

====================

실행한 쿼리의 반환값이 있을 경우, 반환 데이터를 읽어 옵니다.

|  |
| --- |
| $row = mysqli\_fetch\_object($result); |

**03.6.4 메모리 해제**

====================

쿼리를 실행후에 반환데이터가 있을 경우 반환되는 객체를 통하여 데이터를 반복적으로 읽어오게 됩니다. 데이터의 조회등의 반환 객체는 이와 관련된 메모리를 할당 받게 됩니다. 이는 시스템의 리소스 자원에 효율적으로 사용하는데 방해가 되기 때문에 정상적으로 모든 데이터처리가 이루어진 후에는 리소스를 해제하여 메모리를 절약 하도록 합니다.

|  |
| --- |
| $result->free(); |

MYSQL 쿼리실행후 넘어오는 객체 메모리를 해제합니다.

**03.8 잠금**

====================

데이터베이스 시스템이 파일시스템 보다 효율적이고 향상된 점은 다수의 사용자가 동시에 접속하여 데이터를 처리할 수 있다는 것입니다. 하지만, 아무리 좋은 데이터베이스 관리 시스템 이어도 순간적으로 여러 사람들이 동일한 데이터를 접근 할 때는 문제가 발생될 수 있습니다.

접근의 우선 순위에 따라서 데이터가 순간적으로 변경이 되거나, 데이터의 값에 오류가 발생할 수 있습니다. 특히, 데이터를 변경할 때 동시접속에 의한 문제가 발생될 수 있습니다.

**03.8.1 잠금**

====================

데이블의 데이터를 조작할 때 권한을 통하여 일부 동작들을 제한할 수 있습니다. 테이블을 잠금을 설정합니다.

**쿼리문법)**

|  |
| --- |
| LOCK TABLES 테이블명 옵션 |

옵션종류 :

* READ : SELECT 명령만 가능합니다. 다른 작업들의 명령은 할 수 없습니다.
* READ\_LOCAL : 로컬만 읽기 가능합니다.
* WRITE : LOCK을 실행한 이외의 사용자는 제한합니다.

**콘솔작업 예제)**

|  |
| --- |
| mysql> lock tables members read;  Query OK, 0 rows affected (0.00 sec) |

**PHP 예제)**

mysql.class.php 파일에 메서드 예제를 추가합니다.

|  |
| --- |
| // 테이블 잠금  // 매개인자: 데이블명, 옵션 = 옵션이 없을 경우 read로 설정합니다.  // 반한값 : 객체 this를 반환 합니다.  public function tableLock($tableName, $option="read")  {    $queryString = "LOCK TABLES ";  if ($tableName) {    $queryString .= $tableName;    // 옵션이 없는 경우에는 READ로 기본설정 합니다.  $queryString .= " ".$option;    // 쿼리를 전송합니다.  if (mysqli\_query($this->dbcon, $queryString)=== TRUE) {  $this->msgEcho("쿼리성공] ".$queryString);  $this->msgEcho($tableName." LOCK!");    } else {  $this->msgEcho("Error] ".$queryString);  }    // 객체 반환, 매서드체인  return $this;    }    } |

**03.6.2 해제**

====================

향후 정상적인 데이터 처리를 위해서 반드시 잠금된 테이블은 사용후에 해제를 해주어야 합니다. 잠금 해제 쿼리는 잠금 설정과 달리 별도의 테이블을 선택하지 않습니다. 해제 쿼리는 잠금 설정된 모든 테이블을 동시에 해제 하게 됩니다.

**쿼리문법)**

|  |
| --- |
| UNLOCK TABLES |

**콘솔작업 예제)**

|  |
| --- |
| mysql> unlock tables;  Query OK, 0 rows affected (0.00 sec) |

**PHP 예제)**

mysql.class.php 파일에 메서드 예제를 추가합니다.

|  |
| --- |
| // 테이블 잠금해제  public function tableUnLock()  {    $queryString = "UNLOCK TABLES ";    // 쿼리를 전송합니다.  if (mysqli\_query($this->dbcon, $queryString)=== TRUE) {  $this->msgEcho("쿼리성공] ".$queryString);  $this->msgEcho(" TABLE UNLOCK!");  } else {  $this->msgEcho("Error] ".$queryString);  }    // 객체 반환, 매서드체인  return $this;  } |

테이블 잠금 및 해제 관련 클래스를 추가하고 이를 이용한 예제를 만들어 실험해 보도록 하겠습니다.

**예제파일) sql-05.php**

|  |
| --- |
| <?php  include "dbinfo.php";  include "mysql.class.php";    // ++ Mysqli DB 연결.  $db = new JinyMysql();  // 테이블을 잠금설정 합니다.  $db->tableLock("members");  // 잠금된 테이블을 해제 합니다.  $db->tableUnLock();  ?> |

화면출력)

mysql connected!

members LOCK!

TABLE UNLOCK!

항상 데이터를 삽입하거나 변경할 때 테이블을 잠금 설정하는 것은 안전한 데이터의 조작을 하는데 유용합니다. 또한 모든 조작처리 후에는 반드시 테이블을 다시 해제 하도록 합니다.