CTIS 256 Web Technologies II

Notes # 8

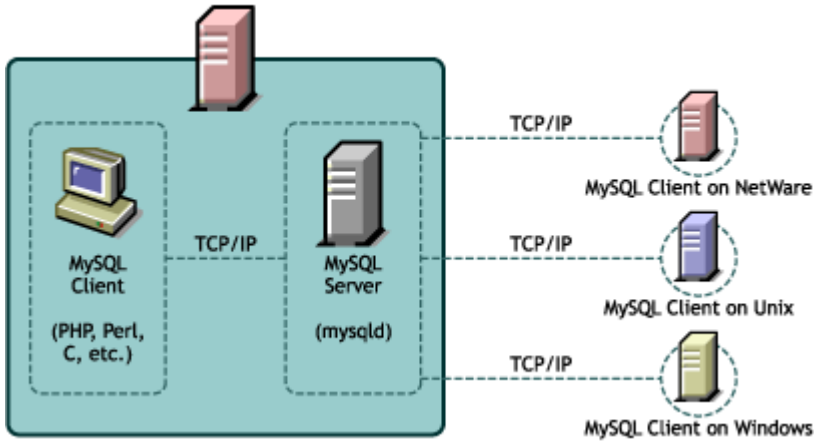
Database, MySQL, PDO Interface

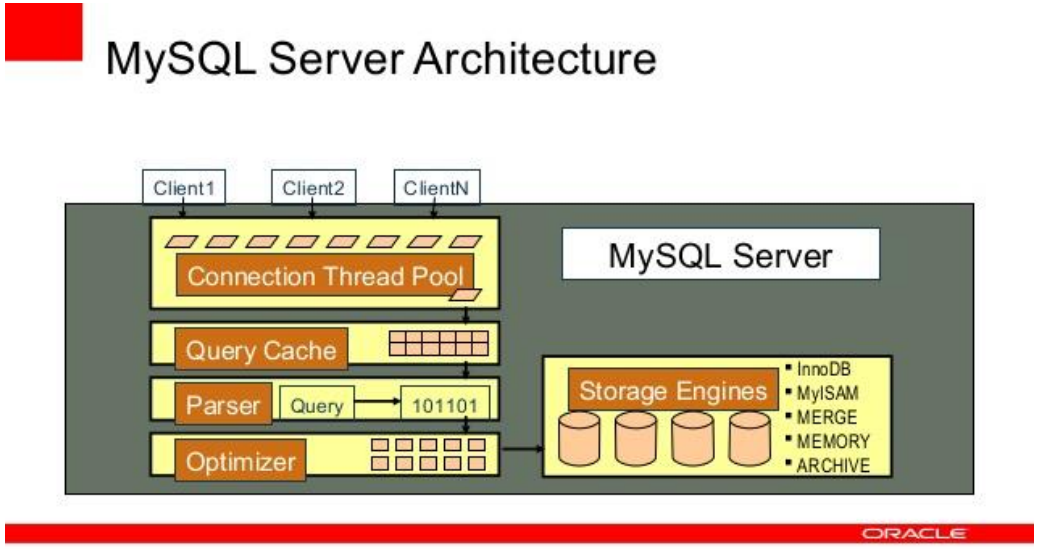
Serkan GENÇ

Relational Database

- Persistent data/records storage in files.
- Database is a collection of files physically, and collection of tables logically. A table is composed of rows (records), and each record has predefined fields/attributes.
- Database Management System (DBMS) is the software that manages Databases.
- Relational (MySQL, Oracle, SQLite) and NoSQL
- Relational and OO database is used for structured data, and NoSQL is for unstructured data.
- **Benefits**:
  - Abstracts data as tables, rows, and fields, easy to organize data. (Database like folder, Table like file)
  - Fast CRUD (Create, Read, Update, Delete) Operations through indexing, sorting.
  - Secure : Authentication and Authorization
  - Remote Data Access : usually client-server architecture.
  - Concurrency/multiuser : data corruption due to concurrent access to the same data.
  - Transactions : commit and rollback
  - SQL : Rich set queries.
  - Database Tools to replicate, import, export, backup, monitoring.

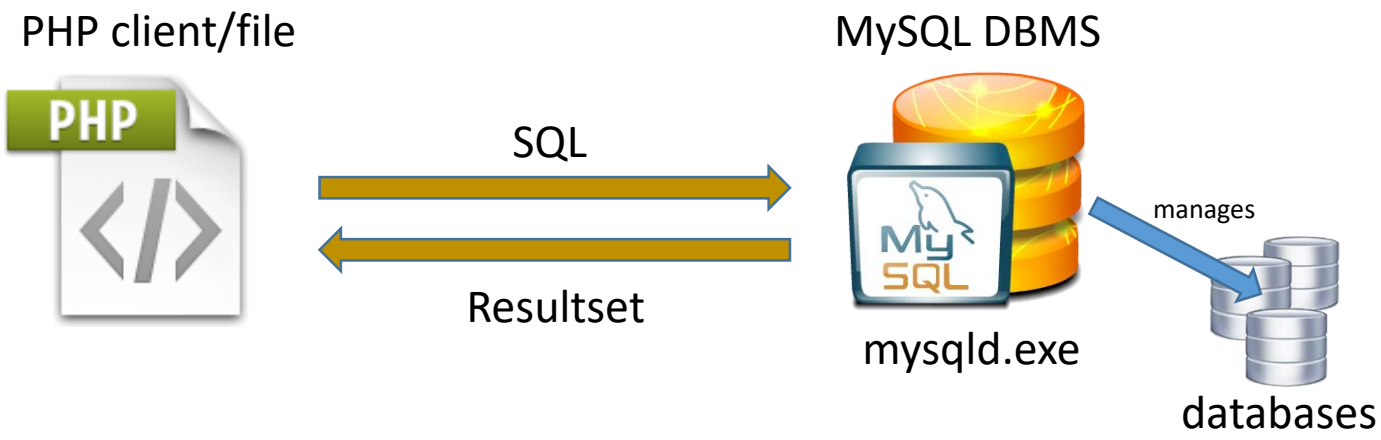
- an open-source Relational Database Management System.
- owned by Oracle System.
- support several storage engines: myISAM, InnoDB, Memory
- easy to use (install, SQL), inexpensive, fast (written in C and C++), multiplatform, scalable (up to 48 CPU), interfaces (JDBC, ODBC, PDO), fully transactional (InnoDB),
- Rich tools such as **phpmyadmin** (web-based mysql interface), **mysql workbench** and so on.





SQL

- Standard Query Language to access and send commands to DBMS.
- It is about the same for all DBMS (minor language dialects)



Basic SQL – MySQL Data types

- **NUMERIC TYPES (SIGNED and UNSIGNED)**
  - **BIT** : 0 or 1
  - **TINYINT** : 1 byte (-128 - +127) or (0 – 255)
  - **SMALLINT** : 2 bytes (-32768 - +32767) or (0 – 65535)
  - **INT** : 4 bytes
  - **BIGINT** : 8 bytes
  - **SERIAL** : BIGINT UNSIGNED, NOT NULL, AUTO_INCR, UNIQUE
  - **DECIMAL(M,D)** : M digits totally, D digits after decimal point, it stores data in decimal instead of binary. It stores exact representation, not approximate. It is useful for financial data such as money. Its calculation is slower comparing to other floating point formats.
  - **FLOAT(M,D)**: 4 bytes, binary floating point, **DOUBLE(M,D)**: 8 bytes. They represent an approximate floating number. However, it is faster in terms of calculation since they are supported by hardware.

- **Non-Binary STRING TYPES : *charset* and *collation***
  - **CHAR(N)**: Fixed length, max 255 chars.
  - **VARCHAR(N)**: Variable length, max 65535 bytes. (name, surname, location, phone number, email, etc.), good for indexing and fast retrieval.
  - **TEXT**: Variable length, max 65535 (message, document, html page, etc)
  - **MEDIUMTEXT** : max 16MB, **LONGTEXT**: max 4 GB
- **Binary Data (Images, pdf doc, video)**
  - **BLOB**: max 64KB, **MEDIUMBLOB**: max 16MB, **LONGBLOB**: 4GB.
- **Date and Time:**
  - **DATE :** YYYY-MM-DD (birthday, etc)
  - **DATETIME** : YYYY-MM-DD HH:MM:SS
  - **TIMESTAMP** : # of seconds after 1970-01-01 00:00:00 UTC, a point in time, used for record changes.
  - **TIME** : HH:MM:SS

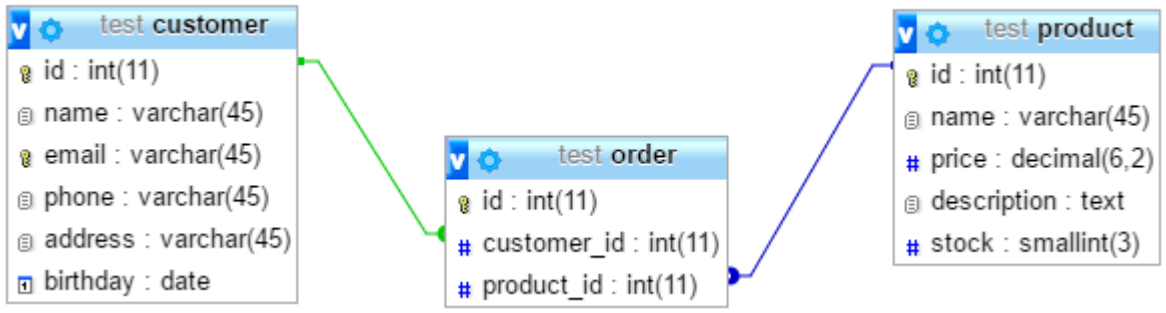
Basic SQL – Create Tables

```
-- -----------------------------------------------------
-- Table `test`.`customer`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `test`.`customer` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) CHARACTER SET 'utf8' NOT NULL,
  `email` VARCHAR(45) CHARACTER SET 'utf8' NOT NULL,
  `phone` VARCHAR(45) CHARACTER SET 'utf8' NOT NULL,
  `address` VARCHAR(45) CHARACTER SET 'utf8' NOT NULL,
  `birthday` DATE NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE INDEX `email_UNIQUE` (`email` ASC))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_turkish_ci;
```

```
-- -----------------------------------------------------
-- Table `test`.`order`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `test`.`order` (
  `id` INT(11) NOT NULL,
  `customer_id` INT(11) NOT NULL,
  `product_id` INT(11) NOT NULL,
  PRIMARY KEY (`id`),
  FOREIGN KEY (`customer_id`) REFERENCES `test`.`customer` (`id`),
  FOREIGN KEY (`product_id`) REFERENCES `test`.`product` (`id`)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_turkish_ci;
```

```
-- -----------------------------------------------------
-- Table `test`.`product`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `test`.`product` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `name` VARCHAR(45) CHARACTER SET 'utf8' NOT NULL,
  `price` DECIMAL(6,2) NOT NULL,
  `description` TEXT CHARACTER SET 'utf8' NOT NULL,
  `stock` SMALLINT(3) NOT NULL,
  PRIMARY KEY (`id`),
  INDEX `price` (`price` ASC))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8
COLLATE = utf8_turkish_ci;
```

ER – Diagram/Design



- **Keywords:**
  - **NOT NULL**: can not be empty/null. There must be a value.
  - **AUTO_INCREMENT**: MySQL automatically assign a unique number if you do not assign a value.
  - **DEFAULT**: if you do not give a value, mysql gives this value.
  - **PRIMARY KEY**: a constraint that uniquely identifies each row, cannot be NULL, indexed by default, at most one *primary key* in a table.
  - **UNIQUE**: can be NULL, many *unique* columns in a table.
  - **FOREIGN KEY**: a constraint that a table can reference a primary key of another table.
  - **INDEX**: for a column, it indexes the values to speed up select queries, but it slows down insert, delete operations.

Basic SQL – Insert/Update/Delete

INSERT Rows :

```
INSERT INTO `test`.`customer` (`name`, `email`, `phone`, `address`, `birthday`) VALUES ('Hakan Kural', 'hakan@gmail.com', '03124534343', 'Bilkent Ankara', '1983-12-24');
INSERT INTO `test`.`customer` (`name`, `email`, `phone`, `address`, `birthday`) VALUES ('Mustafa Atik', 'mustafa@hotmail.com', '05432342323', 'Küçükesat', '1993-1-23');
INSERT INTO `test`.`customer` (`name`, `email`, `phone`, `address`, `birthday`) VALUES ('Ayşegül Yılmaz', 'agul@bilkent.edu.tr', '05443434423', 'Dikmen', '1990-3-13');
INSERT INTO `test`.`customer` (`name`, `email`, `phone`, `address`, `birthday`) VALUES ('Oya Ateş', 'oya@msn.com', '05326545656', 'Keçiören', '2001-4-5');
```

```
INSERT INTO `test`.`product` (`name`, `price`, `description`, `stock`) VALUES ('Samsung UE46ES700', '2300', 'super clear', '30');
INSERT INTO `test`.`product` (`name`, `price`, `description`, `stock`) VALUES ('iPhone 7', '3600', 'piano black', '15');
INSERT INTO `test`.`product` (`name`, `price`, `description`, `stock`) VALUES ('HP LaserJET P1100W', '890', 'free cartridge', '45');
```

```
INSERT INTO `test`.`order` (`customer_id`, `product_id`) VALUES ('1', '2');
INSERT INTO `test`.`order` (`customer_id`, `product_id`) VALUES ('3', '3');
INSERT INTO `test`.`order` (`customer_id`, `product_id`) VALUES ('3', '1');
INSERT INTO `test`.`order` (`customer_id`, `product_id`) VALUES ('4', '2');
```

UPDATE Rows :

```
UPDATE `test`.`customer` SET `email`='hakankural@gmail.com', `phone`='05453213232' WHERE `id`='1';
```

```
UPDATE `test`.`product` SET `price`='3750.00' WHERE `id`='2';
```

DELETE Rows:

```
DELETE FROM `test`.`order` WHERE `id`='21';
```

```
DELETE FROM test.order WHERE product_id = 1 ;
```

```
DELETE FROM test.order WHERE product_id = 1 AND customer_id = 2;
```

Basic SQL – Select/Read Rows

```sql
SELECT * FROM test.customer;
```

id	name	email	phone	address	birthday
1	Hakan Kural	hakankural@gmail.com	05453213232	Bilkent Ankara	1983-12-24
2	Mustafa Atik	mustafa@hotmail.com	05432342323	Küçükesat	1993-01-23
3	Ayşegül Yılmaz	agul@bilkent.edu.tr	05443434423	Dikmen	1990-03-13
4	Oya Ateş	oya@msn.com	05326545656	Keçiören	2001-04-05

```sql
SELECT * FROM test.product;
```

id	name	price	description	stock
1	Samsung UE46ES700	2300.00	super clear	30
2	iPhone 7	3750.00	piano black	15
3	HP LaserJET P1100W	890.00	free cartridge	45

```sql
SELECT * FROM test.`order`;
```

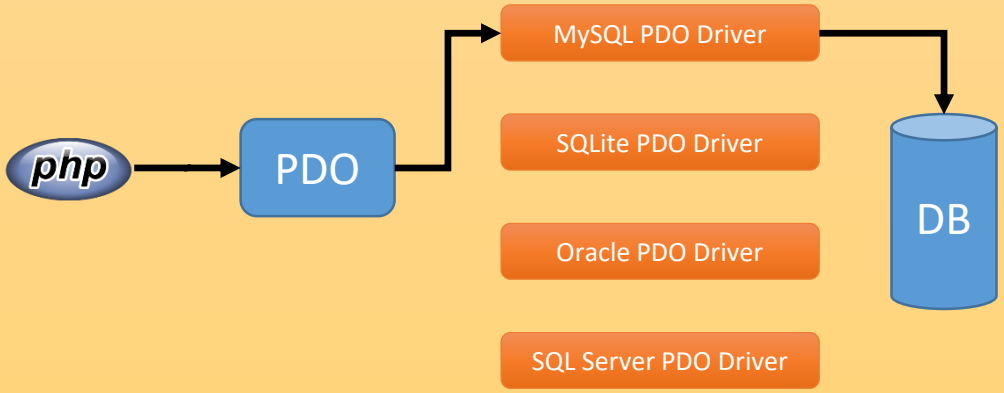
id	customer_id	product_id
0	1	1
19	1	3
20	1	2
22	3	1
23	4	2
24	2	2
25	3	3
26	3	2
27	4	3
28	4	1

```sql
SELECT * FROM test.customer LIMIT 0,2 ;
SELECT * FROM test.customer LIMIT 3,1 ;
SELECT * FROM test.product WHERE stock > 20;
SELECT count(*) total FROM test.customer ;
SELECT * FROM test.customer WHERE phone like '054%' ;
SELECT * FROM test.product WHERE name LIKE '%samsung%' AND price > 1000;
SELECT * FROM test.customer ORDER BY birthday;
SELECT customer_id, COUNT(*) amount FROM test.`order` GROUP BY customer_id;
SELECT name, year(birthday) FROM test.customer WHERE year(birthday) > 1990 ;
SELECT name, datediff(NOW(), birthday)/365 age FROM test.customer ORDER BY birthday;

SELECT P.name, COUNT(*) as amount FROM test.order O
        INNER JOIN test.product P ON O.product_id = P.id
        GROUP BY P.name ;

SELECT * FROM test.order O
        INNER JOIN test.customer C ON O.customer_id = C.id
        INNER JOIN test.product P ON O.product_id = P.id
        ORDER BY C.name;
```

PHP and MySQL

- There are several ways to access and send queries to MySQL DBMS from PHP Applications.

1. **Legacy mysql** interface: deprecated.
2. **mysqli** (mysql improved) : two different interfaces
   - Procedural mysqli interface : similar to old mysql interface, but reimplemented from the scratch.
   - Object Oriented interface: specific to mysql database only
3. **PDO** (PHP Database Object) : Object Oriented, database neutral, as fast as mysqli OO interface.



Legacy mysql interface

```php
<?php
 function logError($str) {
     print "$str" ; // log error here.
     header("Location: not_available.html") ;
     exit ;
 }
 $conn = mysql_connect("localhost", "std", "ctis256") ;
 if ( !$conn ) {
     logError ("LOG: Connection Error") ;
 }
 mysql_select_db("test", $conn ) OR logError("LOG: DB not available") ;
$rs = mysql_query('SELECT * FROM product', $conn) ;
 if ( !$rs ) {
     logError(mysql_error()) ;
 }
$rowCount = mysql_num_rows($rs) ;
print "<p>Total Rows : $rowCount</p>";
while ( $row = mysql_fetch_assoc($rs)) {
     print "<p>" . $row['name'] . " Stock Amount: " . $row['stock'] . "</p>";
 }

 mysql_close($conn) ;
```

PDO Interface for CRUD

- **DSN (Data Source Name)**
  - Required for connection.
  - Only part that is specific to PDO database driver
  - For mysql driver : “mysql:host=IP;dbname=DB;charset=cs;port=#”
  - Port number is optional. Default value is 3306 for mysql server.
  - Example : “**mysql:host=localhost;dbname=test;charset=utf8mb4**”
  - Example : “mysql:host=139.179.10.1;dbname=stars;charset=utf8mb4”
  - For sqlite driver : “sqlite:C:/app/db/users.db”

- **Exception**
  - If something wrong with DSN, user or password, PDO object throws an exception, this is the only way to handle the error.
  - Usually, log the error and try different mysql servers to connect within exception handler, but if there is no option, then exit or redirect to an error page.

- **ResultSet**
  - For read queries such as SELECT and SHOW, **query()** returns either a resultset as a *PDOStatement* object or *FALSE* in failure.
  - Resultset is a list of rows as a result of the query and stored in a PDOStatement object.
  - PDOStatement object contains row related methods such as fetch(), fetchAll(), fetchColumn(), rowCount(), and so on.

```php
<?php
$dsn = 'mysql:host=localhost;dbname=test;charset=utf8mb4' ;
$user = 'std' ;
$pass = 'ctis256' ;

// 1. Create a Connection to MySQL Server.
try {
  $db = new PDO( $dsn, $user, $pass ) ;
} catch( Exception $ex) {
    print "<p>Connection Error : " . $ex->getMessage() . "</p>";
    exit;
}

// 2. Prepare your SELECT sql query and retrieve the result in resultset object
$sql = 'SELECT id, name, price, stock FROM test.product ORDER BY price DESC' ;
$rs = $db->query($sql) ;
// returns ResultSet object that contains both rows and some methods such as
// fetch(), fetchAll(), rowCount() and so on.
// or if it returns FALSE, there is something wrong with query itself.
if ( !$rs) {
    print "<p>Query Error: " . print_r( $db->errorInfo(), true) . "</p>";
    exit ;
}

// 3. Process/Iterate ResultSet Object.
$numRow = $rs->rowCount();
print "<p>Total : $numRow</p>";
// In foreach loop, we can iterate resultset object.
foreach( $rs as $row) {
    print "<pre>" . print_r($row, true) . "</pre>";
}
```

Retrieve Records from a ResultSet

- **FETCHING RESULT: (for Statement Object)**
  - **Foreach** : Using query() within foreach()
  - **fetch()** : Retrieve row by row at a time using fetch() method, it returns each record in two arrays; sequential and associative array. To return only associative array give a parameter as PDO::FETCH_ASSOC, for sequential array use PDO::FETCH_NUM, default value is PDO::FETCH_BOTH. Fetch() method retrieves a record from resultset and forwards the built-in cursor or iterator.
  - **fetchAll()** : Retrieve all rows at once, it converts all records to PHP array format. This is useful if you use the resultset many times in the same file, or random access to the returned records.
  - **fetchColumn()** : instead of whole record, you can access only the specified column. It takes a numeric parameter for the column index. Default value is 0 or the first column.
  - **fetchObject()** : return a record in a given object format. It calls object constructor with the returned data.
- Result Size:
  - **rowCount()** : method returns the number of records from a select query or the affected rows update, delete and insert.

Check if the query returns a row

```php
// Check if a row exists.
$row = $rs->fetch() ;
if ( !$row ) {
    print "Query does not return any record.";
}
```

Fetching Records

```php
// Alternative #1 - use query() in foreach()
foreach( $db->query($sql) as $row) {
    print "<p>" . $row["name"] . " Price : " . $row["price"] . "</p>";
}

// Alternative #2 - Using fetch()
while( $row = $rs->fetch(PDO::FETCH_ASSOC)) {
    print "<p>" . $row["name"] . " Price : " . $row["price"] . "</p>";
}

// Alternative #3 - Using fetchAll()
$rows = $rs->fetchAll(PDO::FETCH_ASSOC) ;
// rows is a PHP array. It is more flexible to traverse, maybe in reverse order.
// or if you use the resultset many times in the same file, this is better.
foreach ( $rows as $row) {
    print "<p>" . $row["name"] . " Price : " . $row["price"] . "</p>";
}

// Alternative #4 - retrieve only a column instead of a complete row.
while( $column = $rs->fetchColumn(2)) {
    print "<p>Price : " . $column . "</p>" ;
}
```

Insert a Record

(with placeholder)

```php
$sql =  "INSERT INTO product  (name, price, description, stock) ";
$sql .= "VALUES (?, ?, ?, ?)" ;
$stmt = $db->prepare($sql) ;
$affected = $stmt->execute(array('JBL Pulse 2', '550', 'Colorful', '22')) ;
if ( !$affected) {
    print "<p>Insert Error : " . print_r($db->errorInfo(), true) . "<p>";
    exit ;
}
print "<p>$affected row added. Last Product ID : " . $db->lastInsertId() . "</p>" ;
```

Insert a Record

(with named-placeholder)

```php
$row = array('name' => 'JBL Pulse 2', 'price' => 555, 'desc' => 'Colorful', 'stock' => '22');
$sql =  "INSERT INTO product ( name, price, description, stock) ";
$sql .= "VALUES (:name, :price, :desc, :stock)" ;
$stmt = $db->prepare($sql) ;
$affected = $stmt->execute( $row ) ;
if ( !$affected) {
    print "<p>Insert Error : " . print_r($db->errorInfo(), true) . "<p>";
    exit ;
}
print "<p>$affected row added. Last Product ID : " . $db->lastInsertId() . "</p>" ;
```

Delete Rows

```php
$sql = 'DELETE FROM product WHERE id = ?' ;
$stmt = $db->prepare($sql) ;
$affected = $stmt->execute( array($_GET["id"]) ) ;
if ( !$affected) {
    print "<p>Delete Error : " . print_r($db->errorInfo(), true) . "<p>";
    exit ;
}
```

Update Rows

```
$newRow = array( 'name' => 'BENQ W1070', 'price' => 2750, 'id' => 13) ;
$sql = 'UPDATE product SET name=:name, price=:price WHERE id=:id' ;
$stmt = $db->prepare($sql) ;
$affected = $stmt->execute( $newRow ) ;
if ( !$affected) {
    print "<p>Update Error : " . print_r($db->errorInfo(), true) . "<p>";
    exit ;
}
```

Exception Handling

- By default, when error occurs in queries, we have to check if the return value is FALSE or not. If it is false, then check the error in detail by *errorInfo()* method that returns an array about the error.
- PDO constructor is an exception. This is why we used try/catch block for *new PDO()* statement.
- Exception handling code is separated from the main code which simplifies coding and maintenance.
- To support exception for queries, we have to change error mode of PDO object:
- **$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION)**

```php
<?php

$dsn = 'mysql:host=localhost;dbname=test;charset=utf8mb4' ;
$user = 'std' ;
$pass = 'ctis256' ;

// 1. Create a Connection to MySQL Server.
try {
  $db = new PDO( $dsn, $user, $pass ) ;
  $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION) ;

// 2. Update with Prepared Statement to prevent SQL Injection
$newRow = array( 'name' => 'BENQ W1070', 'price' => 2750, 'id' => 13) ;
$sql = 'UPDATE product SET name=:name, price=:price WHERE id=:id' ;
$stmt = $db->prepare($sql) ;
$affected = $stmt->execute( $newRow ) ;
} catch( Exception $ex) {
    print "<p>PDO Error : " . $ex->getMessage() . "</p>";
    exit;
}
```

Redirection

- Redirection is about directing a browser to another URL address.
- To do that, "Location" header must be inserted into HTTP Response Packet's header part, and *usually* HTTP Response Status is "301 Moved Permenantly"
- In PHP, header() function is a general function which is used to modifty HTTP Response Packet.
  - header("Content-type: application/json")
  - header("Cache-control: no-cache")
  - header("Expires: Tue, 1 Feb 2018 06:11:34 GMT")
  - header("Content-Type: text/html; charset=utf-8")
  - header("Content-Disposition:attachment;filename='report.pdf'")
  - **header("Location: main.php") :** it adds "Location" header and set status code as 301.

email.php

```php
<?php
    if ( isset($_POST['submit'])) {
        // validate email address
        if ( filter_var($_POST['email'], FILTER_VALIDATE_EMAIL) === false) {
            // Adds Location header to Response packet.
            // It also sets the status as 301,
            // when browser gets 301 status code, it checks out Location
            // header and it prepares another request packet for the new
            // URL address specified in "Location" header.
            header("Location: Error.html") ;

            // header command just puts a line into header, it continues with
            // the following php code.
            // To exit from the program, you need to call "exit" function.
            exit ;
        }
        echo "<p>Your email is " . $_POST['email'] . "</p>" ;
    }
?>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <title></title>
        <style></style>
    </head>
    <body>
        <form action="" method="post">
            Email: <input type="text" name="email">
            <input type="submit" value="submit" name="submit">
        </form>
    </body>
</html>
```