```php
<?php
    /*
    Reference:
    http://code.tutsplus.com/tutorials/why-you-should-be-using-phps-pdo-for-database-access--net-12059
    */
    function connect(){
    /*
    No matter what error mode you set,
    an error connecting will always produce an exception,
    and creating a connection should always
    be contained in a try/catch block.
    */
    try {
    # MS SQL Server and Sybase with PDO_DBLIB
    $DBH = new PDO("mssql:host=$host;dbname=$dbname, $user, $pass");
    $DBH = new PDO("sybase:host=$host;dbname=$dbname, $user, $pass");
    # MySQL with PDO_MYSQL
    $DBH = new PDO("mysql:host=$host;dbname=$dbname", $user, $pass);
    # SQLite Database
    $DBH = new PDO("sqlite:my/database/path/database.db");
    }
    catch(PDOException $e) {
    echo $e->getMessage();
    }
    }
    function error_reporting_mode(){
    $DBH->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_SILENT ); # default
    $DBH->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_WARNING );
    $DBH->setAttribute( PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION );
    }
    function prepare_statements(){
    # no placeholders - ripe for SQL Injection!
    $STH = $DBH->("INSERT INTO folks (name, addr, city) values ($name, $addr, $city)");
    # unnamed placeholders
    $STH = $DBH->("INSERT INTO folks (name, addr, city) values (?, ?, ?)");
    # named placeholders
    $STH = $DBH->("INSERT INTO folks (name, addr, city) value (:name, :addr, :city)");
    }
    function unnamed_placeholders(){
    ### Alternative 1:
    # assign variables to each place holder, indexed 1-3
    $STH->bindParam(1, $name);
    $STH->bindParam(2, $addr);
    $STH->bindParam(3, $city);
    # insert one row
    $name = "Daniel"
    $addr = "1 Wicked Way";
    $city = "Arlington Heights";
    $STH->execute();
    # insert another row with different values
    $name = "Steve"
    $addr = "5 Circle Drive";
    $city = "Schaumburg";
    $STH->execute();
```

```
### Alternative 2:
# the data we want to insert
$data = array('Cathy', '9 Dark and Twisty Road', 'Cardiff');
$STH = $DBH->("INSERT INTO folks (name, addr, city) values (?, ?, ?)");
$STH->execute($data);
}
function named_placeholders(){
### Alternative 1:
# the first argument is the named placeholder name - notice named
# placeholders always start with a colon.
$STH->bindParam(':name', $name);
### Alternative 2:
# the data we want to insert
$data = array( 'name' => 'Cathy', 'addr' => '9 Dark and Twisty', 'city' => 'Cardiff' );
# the shortcut!
$STH = $DBH->("INSERT INTO folks (name, addr, city) value (:name, :addr, :city)");
$STH->execute($data);
### Alternative 3:
# ability to insert objects directly into your database, assuming the properties match the named fields.
# a simple object
class person {
public $name;
public $addr;
public $city;
function __construct($n,$a,$c) {
$this->name = $n;
$this->addr = $a;
$this->city = $c;
}
# etc ...
}
$cathy = new person('Cathy','9 Dark and Twisty','Cardiff');
# here's the fun part:
$STH = $DBH->("INSERT INTO folks (name, addr, city) value (:name, :addr, :city)");
$STH->execute((array)$cathy);
}
function select_data_modes(){
/*
PDO::FETCH_ASSOC: returns an array indexed by column name
PDO::FETCH_BOTH (default): returns an array indexed by both column name and number
PDO::FETCH_BOUND: Assigns the values of your columns to the variables set with the ->bindColumn() method
PDO::FETCH_CLASS: Assigns the values of your columns to properties of the named class. It will create the properties if matching
properties do not exist
PDO::FETCH_INTO: Updates an existing instance of the named class
PDO::FETCH_LAZY: Combines PDO::FETCH_BOTH/PDO::FETCH_OBJ, creating the object variable names as they are used
PDO::FETCH_NUM: returns an array indexed by column number
PDO::FETCH_OBJ: returns an anonymous object with property names that correspond to the column names
*/
/*
These three which will cover most situations: FETCH_ASSOC, FETCH_CLASS, and FETCH_OBJ.
*/
$STH->setFetchMode(PDO::FETCH_ASSOC);
}
function fetch_assoc(){
#This fetch type creates an associative array, indexed by column name.
```

```
# using the shortcut ->query() method here since there are no variable
# values in the select statement.
$STH = $DBH->query('SELECT name, addr, city from folks');
# setting the fetch mode
$STH->setFetchMode(PDO::FETCH_ASSOC);
while($row = $STH->fetch()) {
echo $row['name'] . "\n";
echo $row['addr'] . "\n";
echo $row['city'] . "\n";
}
}
function fetch_obj(){
#This fetch type creates an object of std class for each row of fetched data.
# creating the statement
$STH = $DBH->query('SELECT name, addr, city from folks');
# setting the fetch mode
$STH->setFetchMode(PDO::FETCH_OBJ);
# showing the results
while($row = $STH->fetch()) {
echo $row->name . "\n";
echo $row->addr . "\n";
echo $row->city . "\n";
}
}
function fetch_class(){
#This fetch method allows you to fetch data directly into a class of your choosing.
#the properties of your object are set BEFORE the constructor is called.
#If properties matching the column names don't exist, those properties will be created (as public) for you.
#So if your data needs any transformation after it comes out of the database,
# it can be done automatically by your object as each object is created. (via __consruct() method )
class secret_person {
public $name;
public $addr;
public $city;
public $other_data;
function __construct($other = '') {
// will be called after object has valid data in its properties
$this->address = preg_replace('/[a-z]/', 'x', $this->address);
$this->other_data = $other;
}
}
$STH = $DBH->query('SELECT name, addr, city from folks');
$STH->setFetchMode(PDO::FETCH_CLASS, 'secret_person');
while($obj = $STH->fetch()) {
echo $obj->addr;
}
##### Alternaive: Late properties fetch
# construct() will be invoked before intantiating properties
$STH->setFetchMode(PDO::FETCH_CLASS | PDO::FETCH_PROPS_LATE, 'secret_person');
##### Alternative
# you can pass arguments to the constructor when fetching data into objects
$STH->setFetchMode(PDO::FETCH_CLASS, 'secret_person', array('stuff'));
##### Alternative:
# If you need to pass different data to the constructor for each object,
# you can set the fetch mode inside the fetch method:
```

```php
$i = 0;
while($rowObj = $STH->fetch(PDO::FETCH_CLASS, 'secret_person', array($i))) {
// do stuff
$i++
}
}
function some_other_useful_methods(){
# Transaction Processing
$DBH->beginTransaction();
$DBH->commit();
$DBH->rollBack();
# retrieves the id field for the last insert query
# (should be called inside transaction, if there is any transaction in place)
$DBH->lastInsertId();
# exec() for operations that can not return data other then the affected rows.
$DBH->exec('DELETE FROM folks WHERE 1');
$DBH->exec("SET time_zone = '-8:00'");
# quotes strings so they are safe to use in queries.
# This is your fallback if you're not using prepared statements.!!!
$safe = $DBH->quote($unsafe);
# returns an integer indicating the number of rows affected by an operation.
$rows_affected = $STH->rowCount();
# in a known version of PDO, rowCount() doesn't work for select statements,
# if so, you can use following code instead:
$sql = "SELECT COUNT(*) FROM folks";
if ($STH = $DBH->query($sql)) {
# check the row count
if ($STH->fetchColumn() > 0) {
# issue a real select here, because there's data!
}
else {
echo "No rows matched the query.";
}
}
}
function close_connection(){
# close the connection
$DBH = null;
}
```