PHP – PDO Extension

PDO is an acronym for PHP Data Objects. PHP can interact with most of the relational as well as NOSQL databases. The default PHP installation comes with vendor-specific database extensions already installed and enabled. In addition to such database drivers specific to a certain type of database, such as the mysqli extension for MySQL, PHP also supports abstraction layers such as PDO and ODBC.

The PDO extension defines a lightweight, consistent interface for accessing databases in PHP. The functionality of each vendor-specific extension varies from the other. As a result, if you intend to change the backend database of a certain PHP application, say from PostGreSql to MySQL, you need to make a lot of changes to the code. The PDO API on the other hand doesn't require any changes apart from specifying the URL and the credentials of the new database to be used.

Your current PHP installation must have the corresponding PDO driver available to be able to work with. Currently the following databases are supported with the corresponding PDO interfaces –

Driver Name	Supported Databases
PDO_CUBRID	Cubrid
PDO_DBLIB	FreeTDS / Microsoft SQL Server / Sybase
PDO_FIREBIRD	Firebird
PDO_IBM	IBM DB2
PDO_INFORMIX	IBM Informix Dynamic Server
PDO_MYSQL	MySQL 3.x/4.x/5.x/8.x
PDO_OCI	Oracle Call Interface
PDO_ODBC	ODBC v3 (IBM DB2, unixODBC and win32 ODBC)
PDO_PGSQL	PostgreSQL
PDO_SQLITE	SQLite 3 and SQLite 2
PDO_SQLSRV	Microsoft SQL Server / SQL Azure

By default, the PDO_SQLITE driver is enabled in the settings of php.ini, so if you wi interact with a MySQL database with PDO, make sure that the following line is uncommented by removing the leading semicolon.

```
extension=pdo_mysql
```

You can obtain the list of currently available PDO drivers by calling PDO::getAvailableDrivers() static function in PDO class.

PDO Connection

An instance of PDO base class represents a database connection. The constructor accepts parameters for specifying the database source (known as the DSN) and optionally for the username and password (if any).

The following snippet is a typical way of establishing connection with a MySQL database –

```
<?php
   $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
?>
```

If there is any connection error, a PDOException object will be thrown.

Example

Take a look at the following example –

```
<?php
   $dsn="localhost";
   $dbName="myDB";
   $username="root";
   $password="";
   try{
      $dbConn= new PDO("mysql:host=$dsn;dbname=$dbName",$username,$password);
      Echo "Successfully connected with $dbName database";
   } catch(Exception $e){
      echo "Connection failed" . $e->getMessage();
   }
?>
```

It will produce the following **output** –

```
Successfully connected with myDB database
```

In case of error –

```
Connection failedSQLSTATE[HY000] [1049] Unknown database 'mydb'
```

PDO Class Methods

The PDO class defines the following static methods −

PDO::beginTransaction

After obtaining the connection object, you should call this method to that initiates a transaction.

```
public PDO::beginTransaction(): bool
```

This method turns off autocommit mode. Hence, you need to call commit() method to make persistent changes to the database Calling rollBack() will roll back all changes to the database and return the connection to autocommit mode.This method returns true on success or false on failure.

PDO::commit

The commit() method commits a transaction.

```
public PDO::commit(): bool
```

Since the BeginTransaction disables the autocommit mode, you should call this method after a transaction. It commits a transaction, returning the database connection to autocommit mode until the next call to PDO::beginTransaction() starts a new transaction. This method returns true on success or false on failure.

PDO::exec

The exec() method executes an SQL statement and return the number of affected rows

```
public PDO::exec(string $statement): int|false
```

The exec() method executes an SQL statement in a single function call, returning the number of rows affected by the statement.

Note that it does not return results from a SELECT statement. If you have a SELECT statement that is to be executed only once during your program, consider issuing PDO::query().

On the other hand For a statement that you need to issue multiple times, prepare a PDOStatement object with PDO::prepare() and issue the statement with PDOStatement::execute().

The exec() method need a string parameter that represents a SQL statement to prepare and execute, and returns the number of rows that were modified or deleted by the SQL statement you issued. If no rows were affected, PDO::exec() returns 0.

PDO::query

The query() method prepares and executes an SQL statement without placeholders

```
public PDO::query(string $query, ?int $fetchMode = null): PDOStatement|false
```

This method prepares and executes an SQL statement in a single function call, returning the statement as a PDOStatement object.

PDO::rollBack

The rollback() method rolls back a transaction as initiated by PDO::beginTransaction().

```
public PDO::rollBack(): bool
```

If the database was set to autocommit mode, this function will restore autocommit mode after it has rolled back the transaction.

Note that some databases, including MySQL, automatically issue an implicit COMMIT when a DDL statement such as DROP TABLE or CREATE TABLE is issued within a transaction, and hence it will prevent you from rolling back any other changes within the transaction boundary. This method returns true on success or false on failure.

Example

The following code creates a student table in the myDB database on a MySQL server.

```
<?php
   $dsn="localhost";
   $dbName="myDB";
   $username="root";
   $password="";
   try{
      $conn= new PDO("mysql:host=$dsn;dbname=$dbName",$username,$password);
      Echo "Successfully connected with $dbName database";
      $qry = <<<STRING
      CREATE TABLE IF NOT EXISTS STUDENT (
         student_id INT AUTO_INCREMENT,
         name VARCHAR(255) NOT NULL,
         marks INTEGER(3),
         PRIMARY KEY (student_id)
      );
      STRING;
      echo $qry . PHP_EOL;
      $conn->exec($qry);
      $conn->commit();
      echo "Table created\n";
   }
   catch(Exception $e){
      echo "Connection failed : " . $e->getMessage();
   }
?>
```

Example

Use the following code to insert a new record in student table created in the above example −

```
<?php
   $dsn="localhost";
   $dbName="myDB";
   $username="root";
   $password="";
   try {
      $conn= new PDO("mysql:host=$dsn;dbname=$dbName",$username,$password);
      echo "Successfully connected with $dbName database";

      $sql = "INSERT INTO STUDENT values(1, 'Raju', 60)";
      $conn->exec($sql);
      $conn->commit();
      echo "A record inserted\n";
   } catch(Exception $e){
      echo "Connection failed : " . $e->getMessage();
   }
?>
```

Example

The following PHP script fetches all the records in the student table –

```
<?php
   $dsn="localhost";
   $dbName="myDB";
   $username="root";
   $password="";
   try {
      $conn= new PDO("mysql:host=$dsn;dbname=$dbName",$username,$password);
      echo "Successfully connected with $dbName database";
      $sql = "SELECT * from student";
      $statement = $conn->query($sql);
      $rows = $statement->fetchAll(PDO::FETCH_ASSOC);

      foreach ($rows as $row) {
         var_dump($row);
      }
   } catch(Exception $e){
      echo "Connection failed : " . $e->getMessage();
   }
?>
```