I'm not a bot

Before using database we need to connect it first. This process is common in all types of database irrespective of language we use. We will learn here the code for connecting to MySQL database using PDO class. PHP Data Object PDO installation or enable and creating connection string to manage MySQL database It is always a good practice to keep the database connection details at a common place and call it from different pages where it is required. This helps in maintaining the code when the database connection details changes or the script is shifted to different server with different connection details. If we are not using a common file then to change the database details we need not change the connection string in all the files. By keeping all details at one place if we update the details then all the files will use the same (updated) details. We can keep the connection details in config.php file or any other file name you find suitable. All the codes inside this file are to be within the PHP code block, so it will not be exposed if it is opened directly in browser. Let us start with the basic connection code in PDO for PHP & MySQL (PHP MySQL Connection code for old system is here) config.php The code below is kept inside config.php file and can be connected from any other file in the script. More on SQLite connection Common file config.php to keep all database connection details and connected from all script pages. Here the variables $host_name is the name of the MySQL server host or ip address ('localhost' or '127.0.0.1') To get the database connection we have to keep these two lines inside any file and start using the database require "config.php"; Example 2: Connecting to a PostgreSQL Database Using PHP PDO Connecting to a PostgreSQL database with PDO is similar to MySQL but requires a different DSN. Heres an example: Example 3: Setting Connection Timeout in PDO Its possible to define a connection timeout by setting the ATTR_TIMEOUT attribute: Example 4: Using Persistent Connections in PDO Persistent connections improve performance by reusing the same connection rather than opening a new one each time: Handling Connection Errors Gracefully Handling connection errors in a structured way helps improve user experience and debugging: Google Cloud & MySQL connection using PDO Details of MySQL database setup More on Managing MySQL database at Google cloud Download Zip file to test your PHP PDO script PDO References PDO Fetch record Last update on August 19 2022 21:51:13 (UTC/GMT +8 hours) Due to its simplicity and ease of use, PHP is a widely-used open source general-purpose scripting language. PHP is used for creating interactive and dynamic web pages quickly and can access a wide range of relational database management systems such as MySQL, PostgreSQL, and SQLite. Many of us already access MySQL databases by using either the MySQL or MySQLi extensions. As of version 5.1 PHP provides new database connection abstraction library, PHP Data Objects (PDO). What is PDO? PDO - PHP Data Object. A set of PHP extensions that provide a core PDO class and database specific drivers. Provides a vendor-neutral lightweight data-access abstraction layer. Focus on data access abstraction rather than database abstraction. PDO requires the new object oriented features in the core of PHP 5, therefore it will not run with earlier versions of PHP. Installing PDO PDO is dividing into two components: - Core which provides the interface. - Drivers to access particular driver. Installing PDO on Unix systems: -- PDO (Core) and the PDO_SQLITE driver (SQLITE driver) is enabled by default as of PHP 5.1.0. To access other databases you must enable the PDO driver. -- To install PDO as a shared module the php.ini needs to be updated so that the PDO extension will be loaded automatically when PHP runs. You also need to enable other database specific drivers and they must be listed after the pdo.so line, as PDO must be initialized before the database-specific extensions can be loaded. If you built PDO and the database-specific extensions statically, you can skip this step: extension=pdo.so Installing PDO on Windows systems: -- PDO and all the major drivers ship with PHP as shared extensions, and simply need to be activated by editing the php.ini file : extension=php_pdo.dll. This step is not necessary for PHP 5.3 and above, as a DLL is no longer required for PDO. -- Next, choose the other database-specific DLL files and either use dl() to load them at runtime, or enable them in php.ini below php_pdo.dll. To get the effect of a new configuration in php.ini file you will need to restart PHP. Predefined Constants Supported Database PDO interface is available in the following drivers: Database name Driver name Cubrid PDO_CUBRID FreeTDS / Microsoft SQL Server / Sybase PDO_DBLIB Firebird/Interbase 6 PDO_FIREBIRD IBM DB2 PDO_IBM IBM Informix Dynamic Server PDO_INFORMIX MySQL 3.x/4.x/5.x PDO_MYSQL Oracle Call Interface PDO_OCI ODBC v3 (IBM DB2, unixODBC and win32 ODBC) PDO_ODBC PostgreSQL PDO_PGSQL SQLite 3 and SQLite 2 PDO_SQLITE Microsoft SQL Server / SQL Azure PDO_SQLSRV 4D PDO_4D Sample database, table, table structure, table records for various examples MySQL: Database Name: hr Host Name: localhost Database user: root Password: ' ' Structure of the table: user_details Records of the table: user_details PostgreSQL: Date base Name: postgres Host Name: localhost Database user: postgres Password: abc123 Structure of the table: user_details Records of the table: user_details The PDO class The class represents a connection between PHP and a database server. Syntax: PDO { __construct (string $dsn [, string $username [, string $password [, array $driver_options]]]) bool beginTransaction (void) bool commit (void) mixed errorCode (void) array errorInfo (void) int exec (string $statement) mixed getAttribute (int $attribute) static array getAvailableDrivers (void) bool inTransaction (void) string lastInsertId ([string $name = NULL]) PDOStatement prepare (string $statement [, array $driver_options = array()]) PDOStatement query (string $statement) string quote (string $string [, int $parameter_type = PDO::PARAM_STR]) bool rollBack (void) bool setAttribute (int $attribute , mixed $value) }mp($var_name);} Details of the PDO class methods : PDO::__construct Creates a PDO instance representing a connection to a database. Syntax: PDO::__construct() (string $dsn [, string $username [, string $password [, array $driver_options]]]) Parameters : dsn - The Data Source Name, or DSN, contains the information required to connect to the database. The string contains the prefix name (e.g. pgsql for PostgreSQL database), a colon, and the server keyword. username - A string that contains the user's name. This parameter is optional for some PDO drivers. password - A string that contains the user's password. This parameter is optional for some PDO drivers. driver_options - Optional. A key=>value array of driver-specific connection options. Return Value: Returns a PDO object on success. If failure, returns a PDOException object. Database Connections Connections are established by creating instances of the PDO base class. It doesn't matter which driver you want to use; you always use the PDO class name. The constructor accepts parameters for specifying the database source (known as the DSN) and optionally for the username and password (if any). MySQL connection PostgreSQL connection Handling connection errors If there are any connection errors, a PDOException object will be thrown. You may catch the exception if you want to handle the error condition, or you can leave it to global exception handler which can be set up via set_exception_handler(). MySQL: Here the user id is wrong. Output: Error : SQLSTATE[28000] [1045] Access denied for user 'rootT'@'localhost' (using password: NO) PostgreSQL: Here the database name is wrong. Output: Error: SQLSTATE[08006] [7] could not connect to server: Connection refused (0x0000274D/10061) Is the server running on host "localhost" (::1) and accepting TCP/IP connections on port 5432? FATAL: password authentication failed for user "postgress" Closing a connection Persistent connections Many web applications will benefit from making persistent connections to database servers. Persistent connections are not closed at the end of the script but are cached and re-used when another script requests a connection using the same credentials. The persistent connection cache allows you to avoid the overhead of establishing a new connection every time a script needs to talk to a database, resulting in a faster web application. MySQL: PostgreSQL: PDO::beginTransaction Turns off auto-commit mode and begins a transaction. The transaction begins with PDO::beginTransaction and will end when PDO::commit or PDO::rollback is called. Syntax: bool PDO::beginTransaction (void) Return Value: Returns TRUE on success or FALSE on failure. Example: The following example a MySQL database called hr and table called user_details have used. It starts a transaction and then executes a command to add one row into the table user_details. The command is sent to the database and the transaction is explicitly ended with PDO::commit. Output: PDO::ATTR_AUTOCOMMIT: PDO::ATTR_ERRMODE: 0 PDO::ATTR_CASE: 0 PDO::ATTR_CLIENT_VERSION: 8.3.6 PDO::ATTR_CONNECTION_STATUS: Connection OK; waiting to send. PDO::ATTR_ORACLE_NULLS: 0 PDO::ATTR_PERSISTENT: PDO::ATTR_PREFETCH: PDO::ATTR_SERVER_INFO: PID: 5940; Client Encoding: UTF8; Is Superuser: on; Session Authorization: postgres; Date Style: ISO, MDY PDO::ATTR_SERVER_VERSION: 9.1.3 PDO::ATTR_TIMEOUT: PDO::getAvailableDrivers Return an array of available PDO drivers in your PHP installation. Syntax: array PDO::getAvailableDrivers (); Return Value: An array with the list of PDO drivers. Example: The following example returns an array of available PDO driver names. Output: Array ([0] => mysql [1] => sqlite) PDO::inTransaction Checks if a transaction is currently active within the driver. This method only works for database drivers that support transactions. Syntax: bool PDO::inTransaction (void) Return Value: Returns TRUE if a transaction is currently active, and FALSE if not. PDO::lastInsertId Returns the identifier of the last inserted row or sequence value into a table in the database. Syntax: string PDO::lastInsertId ([string $name = NULL]) Return Value: If a sequence name was not specified for the name parameter, PDO::lastInsertId() returns a string representing the row ID of the last row that was inserted into the database. If a sequence name was specified for the name parameter, PDO::lastInsertId() returns a string representing the last value retrieved from the specified sequence object. If the PDO driver does not support this capability, PDO::lastInsertId() triggers an IM001 SQLSTATE. Example: The following example (PostgreSQL database is used) returns the ID of the last inserted row or sequence value. Output: Deleted 4 number of rows PDOStatement::setAttribute Set a statement attribute. Currently, no generic attributes are set but only driver specific: Syntax: bool PDOStatement::setAttribute (int $attribute , mixed $value) Return Value: Returns TRUE on success or FALSE on failure. PDOStatement::setFetchMode Set the default fetch mode for this statement. Syntax: public bool PDOStatement::setFetchMode (int $mode) public bool PDOStatement::setFetchMode (int $PDO::FETCH_COLUMN , int $colno) public bool PDOStatement::setFetchMode (int $PDO::FETCH_CLASS , string $classname , array $ctorargs) public bool PDOStatement::setFetchMode (int $PDO::FETCH_INTO , object $object) Parameters: Name Description Type mode The fetch mode must be one of the PDO::FETCH_* constants. See the constants list. mixed colno Column number. int classname Class name. string ctorargs Constructor arguments. array object Object. object Return Value: Returns TRUE on success or FALSE on failure. Example - 1: The following example demonstrates the uses of PDO::FETCH_ASSOC, PDO::FETCH_NUM, PDO::FETCH_BOTH, PDO::FETCH_LAZY, PDO::FETCH_OBJ constants.