

```
1  <?php
2  class Database{
3
4      /**
5       * database connection object
6       * @var \PDO
7       */
8      protected $pdo;
9
10     /**
11      * Connect to the database
12      */
13     public function __construct(\PDO $pdo)
14     {
15         $this->pdo = $pdo;
16         $this->pdo->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
17     }
18
19     /**
20      * Return the pdo connection
21      */
22     public function getPdo()
23     {
24         return $this->pdo;
25     }
26
27     /**
28      * Changes a camelCase table or field name to lowercase,
29      * underscore spaced name
30      *
31      * @param string $string camelCase string
32      * @return string underscore_space string
33      */
34     protected function camelCaseToUnderscore($string)
35     {
36         return strtolower(preg_replace('/([a-z])([A-Z])/','_$1_$2', $string));
37     }
38
39     /**
40      * Returns the ID of the last inserted row or sequence value
41      *
42      * @param string $param Name of the sequence object from which the ID should
43      ... be returned.
44      * @return string representing the row ID of the last row that was inserted
45      ... into the database.
46      */
47     public function lastInsertId($param = null)
48     {
49         return $this->pdo->lastInsertId($param);
50     }
51 }
```

```
49
50 /**
51  * handler for dynamic CRUD methods
52  *
53  * Format for dynamic methods names -
54  * Create: insertTableName($arrData)
55  * Retrieve: getTableNameByFieldName($value)
56  * Update: updateTableNameByFieldName($value, $arrUpdate)
57  * Delete: deleteTableNameByFieldName($value)
58  *
59  * @param string      $function
60  * @param array       $arrParams
61  * @return array|bool
62  */
63 public function __call($function, array $params = array())
64 {
65     if (! preg_match('/^(get|update|insert|delete)(.*)$/', $function,
... $matches)) {
66         throw new \BadMethodCallException($function.' is an invalid method
... Call');
67     }
68
69     if ('insert' == $matches[1]) {
70         if (! is_array($params[0]) || count($params[0]) < 1) {
71             throw new \InvalidArgumentException('insert values must be an
... array');
72         }
73         return $this->insert($this->camelCaseToUnderscore($matches[2]),
... $params[0]);
74     }
75
76     list($tableName, $fieldName) = explode('By', $matches[2], 2);
77     if (! isset($tableName, $fieldName)) {
78         throw new \BadMethodCallException($function.' is an invalid method
... Call');
79     }
80
81     if ('update' == $matches[1]) {
82         if (! is_array($params[1]) || count($params[1]) < 1) {
83             throw new \InvalidArgumentException('update fields must be an
... array');
84         }
85         return $this->update(
86             $this->camelCaseToUnderscore($tableName),
87             $params[1],
88             array($this->camelCaseToUnderscore($fieldName) => $params[0])
89         );
90     }
91
92     //select and delete method
```

```
93         return $this->{$matches[1]}(
94             $this->camelCaseToUnderscore($tableName),
95             array($this->camelCaseToUnderscore($fieldName) => $params[0])
96         );
97     }
98
99     /**
100     * Record retrieval method
101     *
102     * @param string      $tableName name of the table
103     * @param array       $where      (key is field name)
104     * @return array|bool (associative array for single records, multidim array
105     ... for multiple records)
106     */
107     public function get($tableName, $whereAnd = array(), $whereOr =
108     ... array(), $whereLike = array())
109     {
110         $cond = '';
111         $s=1;
112         $params = array();
113         foreach($whereAnd as $key => $val)
114         {
115             $cond .= " And ".$key." = :a".$s;
116             $params['a'.$s] = $val;
117             $s++;
118         }
119         foreach($whereOr as $key => $val)
120         {
121             $cond .= " OR ".$key." = :a".$s;
122             $params['a'.$s] = $val;
123             $s++;
124         }
125         foreach($whereLike as $key => $val)
126         {
127             $cond .= " OR ".$key." like '% :a".$s."%'";
128             $params['a'.$s] = $val;
129             $s++;
130         }
131         $stmt = $this->pdo->prepare("SELECT $tableName.* FROM $tableName WHERE 1
132     ... ".$cond);
133         try {
134             $stmt->execute($params);
135             $res = $stmt->fetchAll();
136
137             if (! $res || count($res) != 1) {
138                 return $res;
139             }
140             return $res;
141         } catch (\PDOException $e) {
142             throw new \RuntimeException("[ ".$e->getCode()."] : ").
```

```
139... $e->getMessage());
140     }
141 }
142
143 public function getAllRecords($tableName, $fields='*', $cond='', $orderBy='',
... $limit='')
144 {
145     //echo "SELECT $tableName.$fields FROM $tableName WHERE 1 ".$cond."
... ".$orderBy." ".$limit;
146     //print "<br>SELECT $fields FROM $tableName WHERE 1 ".$cond."
... ".$orderBy." ".$limit;
147     $stmt = $this->pdo->prepare("SELECT $fields FROM $tableName WHERE 1
... ".$cond." ".$orderBy." ".$limit);
148     //print "SELECT $fields FROM $tableName WHERE 1 ".$cond." ".$orderBy." "
... ;
149     $stmt->execute();
150     $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
151     return $rows;
152 }
153
154 public function getRecFrmQry($query)
155 {
156     //echo $query;
157     $stmt = $this->pdo->prepare($query);
158     $stmt->execute();
159     $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
160     return $rows;
161 }
162
163 public function getRecFrmQryStr($query)
164 {
165     //echo $query;
166     $stmt = $this->pdo->prepare($query);
167     $stmt->execute();
168     return array();
169 }
170 public function getQueryCount($tableName, $field, $cond='')
171 {
172     $stmt = $this->pdo->prepare("SELECT count($field) as total FROM
... $tableName WHERE 1 ".$cond);
173     try {
174         $stmt->execute();
175         $res = $stmt->fetchAll(PDO::FETCH_ASSOC);
176
177         if (! $res || count($res) != 1) {
178             return $res;
179         }
180         return $res;
181     } catch (\PDOException $e) {
182         throw new \RuntimeException("[".$e->getCode()."] : ").
```

```
182... $e->getMessage());
183     }
184 }
185
186 /**
187  * Update Method
188  *
189  * @param string $tableName
190  * @param array $set (associative where key is field name)
191  * @param array $where (associative where key is field name)
192  * @return int number of affected rows
193  */
194 public function update($tableName, array $set, array $where)
195 {
196     $arrSet = array_map(
197         function($value) {
198             return $value . '=: ' . $value;
199         },
200         array_keys($set)
201     );
202
203     $stmt = $this->pdo->prepare(
204         "UPDATE $tableName SET ". implode(', ', $arrSet).' WHERE '.
205         key($where). '=: '. key($where) . 'Field'
206     );
207
208     foreach ($set as $field => $value) {
209         $stmt->bindValue(':'.$field, $value);
210     }
211     $stmt->bindValue(':'.$key($where) . 'Field', current($where));
212     try {
213         $stmt->execute();
214
215         return $stmt->rowCount();
216     } catch (\PDOException $e) {
217         throw new \RuntimeException("[".$e->getCode()."] : ".
218         $e->getMessage());
219     }
220 }
221
222 /**
223  * Delete Method
224  *
225  * @param string $tableName
226  * @param array $where (associative where key is field name)
227  * @return int number of affected rows
228  */
229 public function delete($tableName, array $where)
230 {
231     $stmt = $this->pdo->prepare("DELETE FROM $tableName WHERE ".key($where) .
```

```
229... ' = '?'');
230     try {
231         $stmt->execute(array(current($where)));
232
233         return $stmt->rowCount();
234     } catch (\PDOException $e) {
235         throw new \RuntimeException("[ ".$e->getCode()."] : ".
... $e->getMessage());
236     }
237 }
238
239
240 public function deleteQry($query)
241 {
242     $stmt = $this->pdo->prepare($query);
243     $stmt->execute();
244 }
245
246
247 /**
248  * Insert Method
249  *
250  * @param string $tableName
251  * @param array $arrData (data to insert, associative where key is field
... name)
252  * @return int number of affected rows
253  */
254 public function insert($tableName, array $data)
255 {
256     $stmt = $this->pdo->prepare("INSERT INTO $tableName (".implode(', ',
... array_keys($data)).")
257         VALUES (".implode(', ', array_fill(0, count($data), '?')).")"
258     );
259     try{
260         $stmt->execute(array_values($data));
261         return $stmt->rowCount();
262     } catch (\PDOException $e) {
263         throw new \RuntimeException("[ ".$e->getCode()."] : ".
... $e->getMessage());
264     }
265 }
266 /**
267  * Print array Method
268  *
269  * @param array
270  */
271 public function arprint($array){
272     print"<pre>";
273     print_r($array);
274     print"</pre>";
```

```
275     }
276     /**
277      * Maker Model Name Method
278      *
279      * @param Int make id
280      * @param Int name id
281      */
282     public function getModelMake($makeID,$nameID){
283         $vehMakeData = self::getRecFrmQry('SELECT veh_make_id,veh_make_name
... FROM tb_vehicle_make WHERE veh_make_id="'.$makeID.'");
284         $vehNameData = self::getRecFrmQry('SELECT veh_name_id,veh_name FROM
... tb_vehicle_name WHERE veh_name_id="'.$nameID.'");
285         return $vehMakeData[0]['veh_make_name'].' '.$vehNameData[0]['veh_name'];
286     }
287     /**
288      * Cache Method
289      *
290      * @param string QUERY
291      * @param Int Time default 0 set
292      */
293     public function getCache($sql,$cache_min=0) {
294         $f = 'cache/'.md5($sql);
295         if ( $cache_min!=0 and file_exists($f) and ( (time()-filemtime($f))/60 <
... $cache_min ) ) {
296             $arr = unserialize(file_get_contents($f));
297         }
298         else {
299             unlink($f);
300             $arr = self::getRecFrmQry($sql);
301             if ($cache_min!=0) {
302                 $fp = fopen($f,'w');
303                 fwrite($fp,serialize($arr));
304                 fclose($fp);
305             }
306         }
307         return $arr;
308     }
309
310
311 }
312 ?>
```