```php
<?php
class Database{

    /**
     * database connection object
     * @var \PDO
     */
    protected $pdo;

    /**
     * Connect to the database
     */
    public function __construct(\PDO $pdo)
    {
        $this->pdo = $pdo;
        $this->pdo->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
    }

    /**
     * Return the pdo connection
     */
    public function getPdo()
    {
        return $this->pdo;
    }

    /**
     * Changes a camelCase table or field name to lowercase,
     * underscore spaced name
     *
     * @param  string $string camelCase string
     * @return string underscore_space string
     */
    protected function camelCaseToUnderscore($string)
    {
        return strtolower(preg_replace('/([a-z])([A-Z])/', '$1_$2', $string));
    }

    /**
     * Returns the ID of the last inserted row or sequence value
     *
     * @param  string $param Name of the sequence object from which the ID should
be returned.
     * @return string representing the row ID of the last row that was inserted
into the database.
     */
    public function lastInsertId($param = null)
    {
        return $this->pdo->lastInsertId($param);
    }
```

```php

    /**
     * handler for dynamic CRUD methods
     *
     * Format for dynamic methods names -
     * Create:  insertTableName($arrData)
     * Retrieve: getTableNameByFieldName($value)
     * Update: updateTableNameByFieldName($value, $arrUpdate)
     * Delete: deleteTableNameByFieldName($value)
     *
     * @param  string     $function
     * @param  array      $arrParams
     * @return array|bool
     */
    public function __call($function, array $params = array())
    {
        if (! preg_match('/^(get|update|insert|delete)(.*)$/', $function, $matches)) {
            throw new \BadMethodCallException($function.' is an invalid method Call');
        }

        if ('insert' == $matches[1]) {
            if (! is_array($params[0]) || count($params[0]) < 1) {
                throw new \InvalidArgumentException('insert values must be an array');
            }
            return $this->insert($this->camelCaseToUnderscore($matches[2]), $params[0]);
        }

        list($tableName, $fieldName) = explode('By', $matches[2], 2);
        if (! isset($tableName, $fieldName)) {
            throw new \BadMethodCallException($function.' is an invalid method Call');
        }

        if ('update' == $matches[1]) {
            if (! is_array($params[1]) || count($params[1]) < 1) {
                throw new \InvalidArgumentException('update fields must be an array');
            }
            return $this->update(
                $this->camelCaseToUnderscore($tableName),
                $params[1],
                array($this->camelCaseToUnderscore($fieldName) => $params[0])
            );
        }

        //select and delete method
```

```php
        return $this->{$matches[1]}(
            $this->camelCaseToUnderscore($tableName),
            array($this->camelCaseToUnderscore($fieldName) => $params[0])
        );
    }

    /**
     * Record retrieval method
     *
     * @param  string     $tableName name of the table
     * @param  array      $where     (key is field name)
     * @return array|bool (associative array for single records, multidim array for multiple records)
     */
    public function get($tableName,  $whereAnd  =   array(), $whereOr   = array(), $whereLike =   array())
    {
    $cond   =   '';
    $s=1;
    $params =   array();
    foreach($whereAnd as $key => $val)
    {
        $cond   .=  " And ".$key." = :a".$s;
        $params['a'.$s] = $val;
        $s++;
    }
    foreach($whereOr as $key => $val)
    {
        $cond   .=  " OR ".$key." = :a".$s;
        $params['a'.$s] = $val;
        $s++;
    }
    foreach($whereLike as $key => $val)
    {
        $cond   .=  " OR ".$key." like '% :a".$s."%'";
        $params['a'.$s] = $val;
        $s++;
    }
    $stmt = $this->pdo->prepare("SELECT  $tableName.* FROM $tableName WHERE 1 ".$cond);
        try {
            $stmt->execute($params);
            $res = $stmt->fetchAll();

            if (! $res || count($res) != 1) {
               return $res;
            }
            return $res;
        } catch (\PDOException $e) {
            throw new \RuntimeException("[".$e->getCode()."] : ".
```

```
$e->getMessage());
        }
    }

    public function getAllRecords($tableName, $fields='*', $cond='', $orderBy='', $limit='')
    {
        //echo "SELECT  $tableName.$fields FROM $tableName WHERE 1 ".$cond." ".$orderBy." ".$limit;
        //print "<br>SELECT $fields FROM $tableName WHERE 1 ".$cond." ".$orderBy." ".$limit;
        $stmt = $this->pdo->prepare("SELECT $fields FROM $tableName WHERE 1 ".$cond." ".$orderBy." ".$limit);
        //print "SELECT $fields FROM $tableName WHERE 1 ".$cond." ".$orderBy." ";
        $stmt->execute();
        $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
        return $rows;
    }

    public function getRecFrmQry($query)
    {
        //echo $query;
        $stmt = $this->pdo->prepare($query);
        $stmt->execute();
        $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
        return $rows;
    }

    public function getRecFrmQryStr($query)
    {
        //echo $query;
        $stmt = $this->pdo->prepare($query);
        $stmt->execute();
        return array();
    }
    public function getQueryCount($tableName, $field, $cond='')
    {
        $stmt = $this->pdo->prepare("SELECT count($field) as total FROM $tableName WHERE 1 ".$cond);
        try {
            $stmt->execute();
            $res = $stmt->fetchAll(PDO::FETCH_ASSOC);

            if (! $res || count($res) != 1) {
               return $res;
            }
            return $res;
        } catch (\PDOException $e) {
            throw new \RuntimeException("[".$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(',', $arrSet).' WHERE '.
…  key($where). '=:'. key($where) . 'Field'
205            );
206
207           foreach ($set as $field => $value) {
208               $stmt->bindValue(':'.$field, $value);
209           }
210           $stmt->bindValue(':'.key($where) . 'Field', current($where));
211           try {
212               $stmt->execute();
213
214               return $stmt->rowCount();
215           } catch (\PDOException $e) {
216               throw new \RuntimeException("[".$e->getCode()."] : ".
…  $e->getMessage());
217           }
218       }
219
220       /**
221        * Delete Method
222        *
223        * @param  string $tableName
224        * @param  array  $where     (associative where key is field name)
225        * @return int    number of affected rows
226        */
227       public function delete($tableName, array $where)
228       {
229           $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>";
```

```php
    }
    /**
     * Maker Model Name Method
     *
     * @param  Int make id
     * @param  Int name id
     */
    public function getModelMake($makeID,$nameID){
        $vehMakeData    =   self::getRecFrmQry('SELECT veh_make_id,veh_make_name FROM tb_vehicle_make WHERE veh_make_id="'.$makeID.'"');
        $vehNameData    =   self::getRecFrmQry('SELECT veh_name_id,veh_name FROM tb_vehicle_name WHERE veh_name_id="'.$nameID.'"');
        return $vehMakeData[0]['veh_make_name'].' '.$vehNameData[0]['veh_name'];
    }
    /**
     * Cache Method
     *
     * @param  string QUERY
     * @param  Int Time default 0 set
     */
    public function getCache($sql,$cache_min=0) {
      $f = 'cache/'.md5($sql);
      if ( $cache_min!=0 and file_exists($f) and ( (time()-filemtime($f))/60 < $cache_min ) ) {
        $arr = unserialize(file_get_contents($f));
      }
      else {
        unlink($f);
        $arr = self::getRecFrmQry($sql);
        if ($cache_min!=0) {
          $fp = fopen($f,'w');
          fwrite($fp,serialize($arr));
          fclose($fp);
        }
      }
      return $arr;
    }


}
?>
```