SQLite PHP: Querying Data

Summary: this tutorial shows you how various way to query data from SQLite table using PHP PDO.

To query data from a table, you use the following steps:

1. Connect to the SQLite database (https://www.sqlitetutorial.net/sqlite-php/connect/) using the PDO object.
2. Use the `query()` method of the PDO object to execute the SELECT (https://www.sqlitetutorial.net/sqlite-select/) statement. The `query()` method returns a result set as a `PDOStatement` object. If you want to pass values to the `SELECT` statement, you create the `PDOStatement` object by calling the `prepare()` method of the PDO object, bind values using the `bindValue()` method of the `PDOStatement` object, and call the `execute()` method to execute the statement.
3. Loop through the result set using the `fetch()` method of the `PDOStatement` object and process each row individually.

See the following `getProjects()` method.

```
    /**
     * Get all projects
     * @return type
     */
    public function getProjects() {
        $stmt = $this->pdo->query('SELECT project_id, project_name '
                . 'FROM projects');
        $projects = [];
        while ($row = $stmt->fetch(\PDO::FETCH_ASSOC)) {
            $projects[] = [
                'project_id' => $row['project_id'],
                'project_name' => $row['project_name']
            ];
        }
        return $projects;
    }
```

This method retrieves all projects from the `projects` table using the following SELECT statement.

```
SELECT project_id,
       project_name
  FROM projects;
```

First, we called the `query()` method of the PDO object to query the data from the `projects` table. The `query()` method returns PDOStatement object, which is `$stmt.`

Second, we called the fetch() (http://php.net/manual/en/pdostatement.fetch.php) method of the PDOStatement object to retrieve the next row from the result set. We passed the following value to the `fetch_style` parameter of the `fetch()` method.

```
\PDO::FETCH_ASSOC
```

The `fetch_style` parameter determines how the row returned to the caller. The `FETCH_ASSOC` means that the `fetch()` method will return an array indexed by column name.

Third, we collected data inside the while-loop and returned the result as an associative array of projects.

In case you want the fetch() method returns the row in the result set as an object you can use the `\PDO::FETCH_OBJ` or you can use the `fetchObject()` method.

The following `getProjectObjectList()` method returns a list of project objects.

```
    /**
    * Get the project as an object list
    * @return an array of Project objects
    */
   public function getProjectObjectList() {
       $stmt = $this->pdo->query('SELECT project_id, project_name '
               . 'FROM projects');

       $projects = [];
       while ($project = $stmt->fetchObject()) {
           $projects[] = $project;
       }

       return $projects;
   }
```

Note that the property names of the object correspond to the column names in the result set. For example, you can access the property names of the `project` object as:

```
$project->project_id;
$project->project_name;
```

See the following `getTasks()` method.

```
  /**
   * Get tasks by the project id
   * @param int $projectId
   * @return an array of tasks in a specified project
   */
  public function getTaskByProject($projectId) {
      // prepare SELECT statement
```

```
        $stmt = $this->pdo->prepare('SELECT task_id,
                                            task_name,
                                            start_date,
                                            completed_date,
                                            completed,
                                            project_id
                                        FROM tasks
                                       WHERE project_id = :project_id;');

        $stmt->execute([':project_id' => $projectId]);

        // for storing tasks
        $tasks = [];

        while ($row = $stmt->fetch(\PDO::FETCH_ASSOC)) {
            $tasks[] = [
                'task_id' => $row['task_id'],
                'task_name' => $row['task_name'],
                'start_date' => $row['start_date'],
                'completed_date' => $row['completed_date'],
                'completed' => $row['completed'],
                'project_id' => $row['project_id'],
            ];
        }

        return $tasks;
    }
```

In this method, we get all tasks associated with a project therefore we need to pass the project id to the `SELECT` statement.

To do so, we use the `prepare()` method to prepare the `SELECT` statement for execution and pass the project id to the statement using the `execute()` method.

If the `SELECT` statement returns one value e.g., when we use an aggregate function such as COUNT (https://www.sqlitetutorial.net/sqlite-count-function/) , AVG (https://www.sqlitetutorial.net/sqlite-avg/) , SUM (https://www.sqlitetutorial.net/sqlite-sum/) , MIN (https://www.sqlitetutorial.net/sqlite-min/) , MAX (https://www.sqlitetutorial.net/sqlite-max/) , etc. in the query.

To get the value, you use the `fetchColumn()` method that returns a single column from the next row in a result set.

See the following `getTaskCountByProject()` method that returns the number of tasks in a specified project.

```
/**
 * Get the number of tasks in a project
 * @param int $projectId
 * @return int
 */
public function getTaskCountByProject($projectId) {

    $stmt = $this->db->prepare('SELECT COUNT(*)
                                 FROM tasks
                                WHERE project_id = :project_id;');
    $stmt->bindParam(':project_id', $projectId);
    $stmt->execute();
    return $stmt->fetchColumn();
}
```

In this tutorial, we have shown various ways to query data in the SQLite database using PHP PDO.