SQLITE TUTORIAL

SQLite PHP: Transaction

Summary: in this tutorial, we will show you how to use the transaction features of PHP PDO to ensure the data integrity in the SQLite database.

Let's create a new table named `task_documents` that stores the relationships between a task and a document.

```
CREATE TABLE IF NOT EXISTS task_documents (
    task_id     INT NOT NULL,
    document_id INT NOT NULL,
    FOREIGN KEY (
        task_id
    )
    REFERENCES tasks (task_id) ON UPDATE CASCADE
                               ON DELETE CASCADE,
    FOREIGN KEY (
        document_id
    )
    REFERENCES documents (document_id) ON UPDATE CASCADE
                                       ON DELETE CASCADE
);
```

Basically, a task has multiple documents and a document may belong to many tasks. The relationship between a task and a document is many-to-many.

Whenever we add a new document to the `documents` table, we need to assign it to a specific task. We don't want to be in a situation that a document is inserted without belonging to a task.

To ensure this, we must perform both actions: insert a new document and assign it to a task in the all-or-nothing fashion. To achieve this, we use the PDO transaction feature.

Whenever we execute a statement in PDO, the database commits the operation by default. To wrap multiple operations inside a transaction, we call the `beginTransaction()` method of the PDO object as follows:

```
$pdo->beginTransaction();
```

To commit the transaction, you call the `commit()` method:

```
$pdo->commit();
```

In case something wrong happened, you can roll back all the operations using the `rollback()` method as follows:

```
$pdo->rollback();
```

SQLite PHP transaction example

We create a new class name `SQLiteTransaction` for the demonstration.

The following method inserts a new document into the `documents` table and returns the document id.

```
    /**
     * Insert blob data into the documents table
     * @param type $pathToFile
     * @return document id
     */
    public function insertDoc($mimeType, $pathToFile) {
```

```
        $sql = "INSERT INTO documents(mime_type,doc) "
                . "VALUES(:mime_type,:doc)";

        // read data from the file
        $fh = fopen($pathToFile, 'rb');

        $stmt = $this->pdo->prepare($sql);

        // pass values
        $stmt->bindParam(':mime_type', $mimeType);
        $stmt->bindParam(':doc', $fh, \PDO::PARAM_LOB);

        // execute the INSERT statement
        $stmt->execute();

        fclose($fh);

        // return the document id
        return $this->pdo->lastInsertId();
    }
```

The following method assigns a document to a task.

```
    /**
     * Assign a document to a task
     * @param int $taskId
     * @param int $documentId
     */
    private function assignDocToTask($taskId, $documentId) {
        $sql = "INSERT INTO task_documents(task_id,document_id) "
                . "VALUES(:task_id,:document_id)";

        $stmt = $this->pdo->prepare($sql);

        $stmt->bindParam(':task_id', $taskId);
        $stmt->bindParam(':document_id', $documentId);
```

```
        $stmt->execute();
    }
```

The following method inserts a document and assigns it to a task within a single transaction.

```
    /**
     * Add a task and associate a document to it
     * @param int $taskId
     * @param string $mimeType
     * @param string $pathToFile
     */
    public function attachDocToTask($taskId, $mimeType, $pathToFile) {
        try {

            // to make sure the foreign key constraint is ON
            $this->pdo->exec('PRAGMA foreign_keys = ON');

            // begin the transaction
            $this->pdo->beginTransaction();

            // insert a document first
            $documentId = $this->insertDoc($mimeType, $pathToFile);

            // associate document with the task
            $this->assignDocToTask($taskId, $documentId);

            // commit update
            $this->pdo->commit();
        } catch (\PDOException $e) {
            // rollback update
            $this->pdo->rollback();
            //
            throw $e;
        }
    }
```

Notice that you must execute the following statement to enable foreign key (https://www.sqlitetutorial.net/sqlite-foreign-key/) support in SQLite:

```
PRAGMA foreign_keys = ON;
```

Therefore, from the PHP application, we use the following statement:

```
$this->pdo->exec('PRAGMA foreign_keys = ON');
```

Let's create the `index.php` file to test the `SQLiteTransaction` class.

```
<?php

require 'vendor/autoload.php';

use App\SQLiteConnection;
use App\SQLiteTransaction;

$pdo = (new SQLiteConnection())->connect();
$sqlite = new SQLiteTransaction($pdo);

$taskId = 9999;

try {
// add a new task and associate a document
    $sqlite->attachDocToTask($taskId, 'application/pdf', 'assets/test.pdf
} catch (PDOException $e) {
    echo $e->getMessage();
}
```

We assigned an non-existent task_id `9999` to purposely violate the foreign key constraint when assigning the document to the task. As the result, PHP threw a PDO exception that caused all operations to be rolled back.

Now, if you change the task id to any valid value in the `tasks` table, a new document will be inserted into the `documents` table and also new entry is also inserted into the

`task_documents` table.

In this tutorial, we have shown you how to perform the transaction in SQLite using PHP PDO transaction API.