http://www.postgresqltutorial.com/postgresql-php/transaction/

PostgreSQL PHP: Transaction

?

Summary: in this tutorial, you will learn how to perform transactions in the PostgreSQL using PHP PDO.

A transaction is a series of operations performed as a single logical unit of work. A transaction has four characteristics: atomicity, consistency, isolation, and durability (ACID).

By default, PostgreSQL uses the auto-commit mode. It means that every statement that the application issues, PostgreSQL commits it automatically.

To turn off the auto-commit mode in PHP, you call the `beginTransaction()` method of the PDO object.

All the changes that you have made are committed only when you call the `commit()` method of the PDO object.

If there is any exception or error happens, you can cancel the changes using the `rollback()` method of the PDO object.

The typical usage of the transaction in PHP PDO is as follows:

```
<?php

try {
    $pdo->beginTransaction();

    $pdo->query("SELECT * FROM table");

    $stmt = $pdo->prepare("UPDATE QUERY");
    $stmt->execute();

    $stmt = $pdo->prepare("ANOTHER UPADTE QUERY");
    $stmt->execute();

    $db->commit();
} catch (\PDOException $e) {
    $db->rollBack();
    throw $e;
}
```

PostgreSQL PHP transaction example

We will create three tables for the demonstration:

1. `accounts`: stores the account information such as first name, last name
2. `plans`: stores the plan information for the account such as silver, gold, and platinum.
3. `account_plans` : stores the plan for each account with the effective date.

The following CREATE TABLE statements create the three tables:

```
1
2
3  CREATE TABLE accounts(
4    id SERIAL PRIMARY KEY,
5    first_name CHARACTER VARYING(100),
6    last_name CHARACTER VARYING(100)
   );
7
8  CREATE TABLE plans(
9    id SERIAL PRIMARY KEY,
10   plan CHARACTER VARYING(10) NOT NULL
   );
11
12 CREATE TABLE account_plans(
13   account_id INTEGER NOT NULL,
14   plan_id INTEGER NOT NULL,
15   effective_date DATE NOT NULL,
16   PRIMARY KEY (account_id,plan_id),
17   FOREIGN KEY(account_id) REFERENCES accounts(id),
18   FOREIGN KEY(plan_id) REFERENCES plans(id)
   );
19
20
```

The following INSERT statement inserts some sample data into the `plans` table.

Whenever we create an account, we need to assign it a plan that may be silver, gold, or platium. To make sure that an account always has at least one plan at a time, we use the transaction API in PDO.

The following `addAccount()` method performs two main steps:

1. First, insert an account into the `accounts` table and returns the account id.
2. Then, assign the account a specific plan by inserting a new row into the `account_plans` table.

At the beginning of the method, we call the `beginTransaction()` method of the PDO object to start the transaction.

If all the steps succeed, we call the `commit()` method to save the changes. In case an exception occurs in any step, we roll back the changes by calling the `rollback()` method in the `catch` block.

```
1
2
3   /**
4    * Add a new account
5    * @param string $firstName
6    * @param string $lastName
7    * @param int $planId
8    * @param date $effectiveDate
9    */
10  public function addAccount($firstName, $lastName, $planId, $effectiveDate) {
11      try {
12          // start the transaction
13          $this->pdo->beginTransaction();
14
15          // insert an account and get the ID back
16          $accountId = $this->insertAccount($firstName, $lastName);
17
18          // add plan for the account
19          $this->insertPlan($accountId, $planId, $effectiveDate);
20
21          // commit the changes
22          $this->pdo->commit();
23      } catch (\PDOException $e) {
24          // rollback the changes
25          $this->pdo->rollBack();
26          throw $e;
      }
  }
```

The `addAccount()` method uses two other private methods: `insertAccount()` and `insertPlan()` as the following:

```
1
2
3       /**
4        *
5        * @param string $firstName
6        * @param string $lastName
7        * @return int
8        */
9       private function insertAccount($firstName, $lastName) {
10          $stmt = $this->pdo->prepare(
11                  'INSERT INTO accounts(first_name,last_name) '
12                  . 'VALUES(:first_name,:last_name)');
13
14          $stmt->execute([
15              ':first_name' => $firstName,
16              ':last_name' => $lastName
17          ]);
18
            return $this->pdo->lastInsertId('accounts_id_seq');
        }
```

```
1
2
3       /**
4        * insert a new plan for an account
5        * @param int $accountId
6        * @param int $planId
7        * @param int $effectiveDate
8        * @return bool
9        */
10      private function insertPlan($accountId, $planId, $effectiveDate) {
11          $stmt = $this->pdo->prepare(
12                  'INSERT INTO account_plans(account_id,plan_id,effective_date) '
13                  . 'VALUES(:account_id,:plan_id,:effective_date)');
14
15          return $stmt->execute([
16                      ':account_id' => $accountId,
17                      ':plan_id' => $planId,
18                      ':effective_date' => $effectiveDate,
          ]);
      }
```

To test the `AccountDB` class, you use the following code in the `index.php` file.

```php


<?php

require 'vendor/autoload.php';

use PostgreSQLTutorial\Connection as Connection;
use PostgreSQLTutorial\AccountDB as AccountDB;

try {
    // connect to the PostgreSQL database
    $pdo = Connection::get()->connect();

    $accountDB = new AccountDB($pdo);

    // add accounts
    $accountDB->addAccount('John', 'Doe', 1, date('Y-m-d'));
    $accountDB->addAccount('Linda', 'Williams', 2, date('Y-m-d'));
    $accountDB->addAccount('Maria', 'Miller', 3, date('Y-m-d'));


    echo 'The new accounts have been added.' . '<br>';
    //
    $accountDB->addAccount('Susan', 'Wilson', 99, date('Y-m-d'));
} catch (\PDOException $e) {
    echo $e->getMessage();
}


```

How it works.

1. First, connect to the PostgresSQL database.
2. Second, insert three accounts with silver, gold, and platinum levels.
3. Third, try to insert one more account but with a plan id that does not exist in the plans table. Based on the input, the step of assigning the plan to the account fails that cause the whole transaction to be rolled back.

The following shows the output of the index.php file:

```
The new accounts have been added.

SQLSTATE[23503]: Foreign key violation: 7 ERROR: insert or update on table "account_plans" violates foreign key constraint "account_plans_plan_id_fkey" DETAIL: Key (plan_id)=(99) is not present in table "plans".
```

If you query the data in the `accounts` and `account_plans` tables, you will see only three rows inserted in each table.

```

stocks=# SELECT * FROM accounts;
id | first_name | last_name
----+------------+-----------
  1 | John       | Doe
  2 | Linda      | Williams
  3 | Maria      | Miller
(3 rows)

stocks=# SELECT * FROM account_plans;
account_id | plan_id | effective_date
------------+---------+----------------
         1 |       1 | 2016-06-13
         2 |       2 | 2016-06-13
         3 |       3 | 2016-06-13
(3 rows)


```

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

?

Previous Tutorial: PostgreSQL PHP: Querying Data

Next Tutorial: PostgreSQL PHP: Calling Stored Procedures

?

PostgreSQL Quick Start

PostgreSQL PHP

- Connect to PostgreSQL Database Using PDO
- Create New Tables in PHP
- Insert Data Into Tables in PHP
- Update Data In a Table using PDD
- Handle Transaction in PHP
- Query Data From PostgresQL using PDO
- Call PostgreSQL Stored Procedures in PHP
- Manage with BLOB in PHP
- Delete Data From a Table in PHP

About PostgreSQL Tutorial

PostgreSQLTutorial.com is a website dedicated to developers and database administrators who are working on PostgreSQL database management system.

We constantly publish useful PostgreSQL tutorials to keep you up-to-date with the latest PostgreSQL features and technologies. All PostgreSQL tutorials are simple, easy-to-follow and practical.

Recent PostgreSQL Tutorials

- How To Change The Password of a PostgreSQL User
- PostgreSQL AGE Function
- PostgreSQL DATE_PART Function
- PostgreSQL List Users
- PostgreSQL NOW Function
- PostgreSQL DATE_TRUNC Function
- PostgreSQL TO_DATE Function: Convert String to Date
- A Look at PostgreSQL User-defined Data Types
- PostgreSQL Copy Database Made Easy
- How to Get Table, Database, Indexes, Tablespace, and Value Size in PostgreSQL

More Tutorials

- PostgreSQL Cheat Sheet
- PostgreSQL Administration
- PostgreSQL PHP
- PostgreSQL Python
- PostgreSQL JDBC
- PostgreSQL Resources