UCH UNIVERSIDAD DE CIENCIAS Y HUMANIDADES

PROGRAMACIÓN II

SEMANA 14
PHP DATA OBJECT - Transacciones

Eric Gustavo Coronel Castillo
ecoronel@uch.edu.pe
gcoronelc.blogspot.com

PHP Data Object - Transacciones

Todas las aplicaciones manejan transacciones para ejecutar sus procesos, por tal motivo debemos tener mucho cuidado al momento de programarlas. Transacciones mal programadas generan inconsistencias en las bases de datos y por lo tanto los reportes que se obtengan también serán inconsistentes, lo cual sería muy grave para la empresa.

En este capítulo abordare transacciones controladas desde el cliente y transacciones de base de datos con procedimientos almacenados.

Índice

1 DEFINICIONES 4

1.1 TRANSACCIÓN 4
1.2 PROPIEDADES DE UNA TRANSACCIÓN 4
1.3 CONTROL DE TRANSACCIONES 5
1.3.1 Transacciones Controladas Desde el Cliente 6
1.3.2 Transacciones de Base de Datos 7
1.3.3 Transacciones Distribuidas 9

2 TRANSACCIONES CONTROLADAS DESDE EL CLIENTE 10

3 TRANSACCIONES DE BASE DE DATOS 14

1 Definiciones

1.1 Transacción

Una transacción es un grupo de acciones que hacen transformaciones consistentes en las tablas preservando la consistencia de la base de datos. Una base de datos está en un estado consistente si obedece todas las restricciones de integridad definidas sobre ella. Los cambios de estado ocurren debido a actualizaciones, inserciones, y eliminación de datos. Se busca asegurar que la base de datos nunca entre en un estado de inconsistencia, sin embargo, durante la ejecución de una transacción la base de datos puede estar temporalmente en un estado inconsistente, tal como se ilustra en la Figura 1. El punto importante aquí es asegurar que la base de datos regresa a un estado consistente al finalizar la ejecución de una transacción.

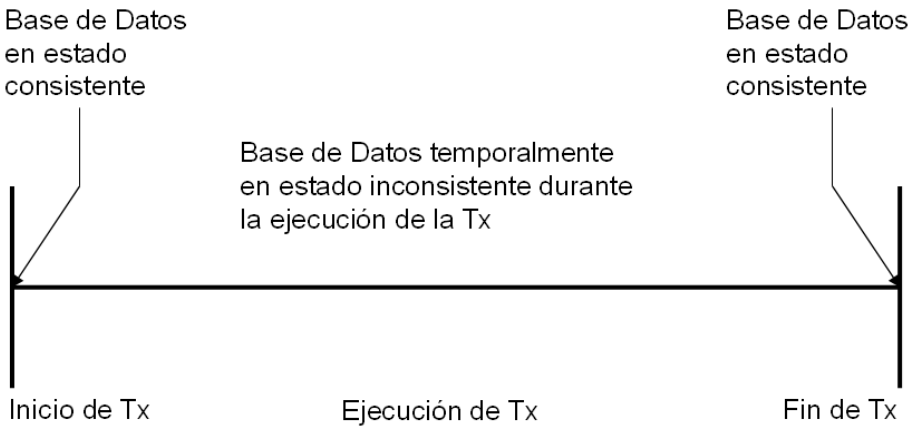


Figura 1 Estado de la Base de Datos durante la transacción.

Lo que se persigue con el manejo de transacciones es por un lado tener una transparencia adecuada de las acciones concurrentes a una base de datos, y por otro lado tener una transparencia adecuada en el manejo de las fallas que se pueden presentar en una base de datos.

1.2 Propiedades de una Transacción

Una transacción debe tener las propiedades ACID, que son las iniciales en inglés de las siguientes características: Atomicity, Consistency, Isolation, Durability.

- **Atomicity (Atomicidad)**

  Una transacción constituye una unidad atómica de ejecución y se ejecuta exactamente una vez; o se realiza todo el trabajo o nada de él en absoluto.

- **Consistency (Coherencia)**

Una transacción mantiene la coherencia de los datos, transformando un estado coherente de datos en otro estado coherente de datos. Los datos enlazados por una transacción deben conservarse semánticamente.

- **Isolation (Aislamiento)**

  Una transacción es una unidad de aislamiento y cada una se produce aislada e independientemente de las transacciones concurrentes. Una transacción nunca debe ver las fases intermedias de otra transacción.

- **Durability (Durabilidad)**

  Una transacción es una unidad de recuperación. Si una transacción tiene éxito, sus actualizaciones persisten, aun cuando falle el equipo o se apague. Si una transacción no tiene éxito, el sistema permanece en el estado anterior antes de la transacción.

1.3 Control de Transacciones

Para controlar transacciones tenemos tres técnicas:

- Transacciones controladas desde el cliente.
- Transacciones de base de datos.
- Transacciones distribuidas.

La que se utilice dependerá del tipo de aplicación que se esté desarrollando (Cliente-Servidor, Distribuida, Web) y la manera como se ha diseñado el control de transacciones por parte del equipo de trabajo.

1.3.1 Transacciones Controladas Desde el Cliente

Son utilizadas en sistemas Cliente-Servidor; las transacciones son iniciadas y finalizadas con instrucciones que se ejecutan desde la aplicación cliente, tal como se ilustra en la Figura 2.

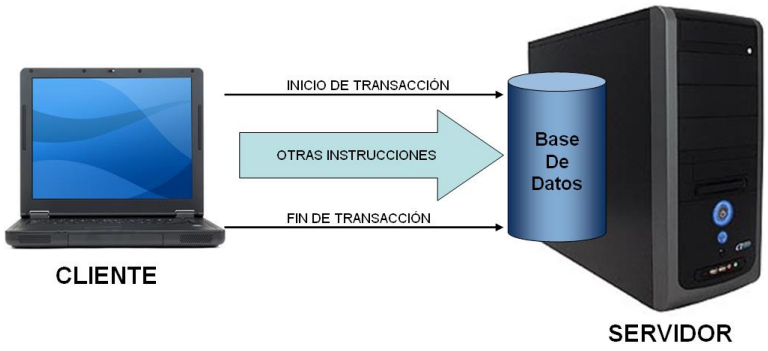


Figura 1 Transacciones controladas desde el cliente.

Los pasos que se siguen son los siguientes:

1. Se envía la instrucción para iniciar la transacción.
2. Se envían las instrucciones para insertar, modificar o eliminar datos.
3. Se envía la instrucción para confirmar la transacción.

Todos estos pasos deben adaptarse al lenguaje de programación que se esté utilizando, por ejemplo si estamos programando con PHP, debemos utilizar PHP PDO y el esquema general sería:

```
try {

    // Inicio de la Transacción
    $dbh->setAttribute(PDO::ATTR_AUTOCOMMIT,FALSE);
    $dbh->beginTransaction();

    // Otras instrucciones

    // Confirmar Transacción
    $dbh->commit();

} catch (PDOException $e) {

    // Cancelar transacción
    $dbh->rollBack();

    // Otras instrucciones de control

}
```

1.3.2 Transacciones de Base de Datos.

Para este caso se utilizan procedimientos almacenados en la base de datos, y son estos los que toman el control de la transacción. Por lo tanto, las transacciones son iniciadas y finalizadas con instrucciones que se ejecutan desde el procedimiento almacenado, tal como se ilustra en la Figura 3.

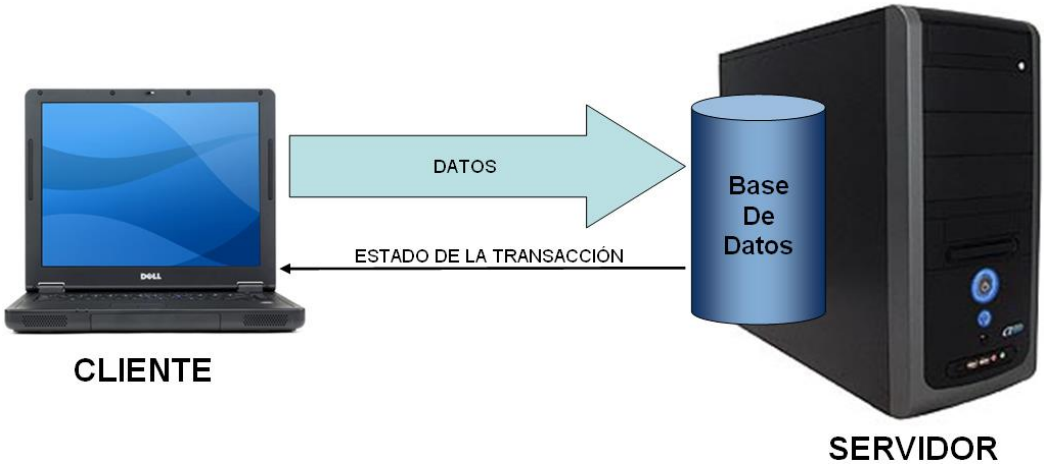


Figura 2 Transacciones controladas en la base de datos.

El problema está en pasarle los datos al procedimiento, sobre todo cuando estos no son fijos, por ejemplo como pasarle los datos de una factura, donde el número de items es variable.

A continuación tenemos esquemas generales para las bases de datos más utilizadas:

- **MySQL**

```
delimiter //

CREATE PROCEDURE nombre_procedimiento ( parámetros )
BEGIN

	-- Iniciar Transacción
	START TRANSACTION;

	-- Instrucciones INSERT, UPDATE ó DELETE

	-- Confirmar Transacción
	COMMIT;

END
//

delimiter;
```

- **SQL Server 2000**

```
CREATE PROCEDURE nombre_procedimiento
( parámetros )
AS
BEGIN
	-- Inicio de la Transacción
	BEGIN TRANSACTION

	-- Instrucciones INSERT, UPDATE ó DELETE

	-- Confirmar Transacción
	COMMIT TRANSACTION
END
GO
```

- **SQL Server 2005 / 2008**

```
CREATE PROCEDURE esquema.nombre_procedimiento
( parámetros )
AS
BEGIN

	BEGIN TRY

		-- Inicio de la Transacción
		BEGIN TRANSACTION

		-- Instrucciones INSERT, UPDATE ó DELETE

		-- Confirmar Transacción
		COMMIT TRANSACTION

	END TRY
	BEGIN CATCH

		-- Cancelar Transacción
		ROLLBACK TRANSACTION

		-- Instrucciones INSERT, UPDATE ó DELETE

	END CATCH
END
GO
```

- **Oracle**

```
CREATE OR REPLACE PROCEDURE nombre_procedimient
( parámetros )
AS
BEGIN

       -- Instrucciones INSERT, UPDATE ó DELETE

       -- Confirmar Transacción
       COMMIT;

EXCEPTION

       WHEN OTHERS THEN
              ROLLBACK;
              -- Otras Instrucciones

END;
```

Cada esquema se debe adaptar al caso específico que se quiere programar.

1.3.3 Transacciones Distribuidas

En este caso se utiliza un servidor de aplicaciones que se encarga de administrar la transacción, tal como se ilustra en la Figura 4.

Figura 3 Transacciones Distribuidas.

En este tipo de solución el servidor de aplicaciones proporciona los componentes de software que controlan el acceso a los datos y las transacciones que se realizan. Las fuentes de datos pueden ser una ó más bases de datos, incluso de diferentes proveedores.

2 Transacciones Controladas Desde el Cliente

La clase PDO provee los métodos para controlar las transacciones desde el cliente, los métodos a utilizar son los siguientes:

- **beginTransaction:** Para iniciar una transacción.
- **commit:** Para confirmar una transacción.
- **rollBack:** Cancelar una transacción.

Pero previamente se debe establecer el atributo **ATTR_AUTOCOMMIT** en **FALSE**.

El esquema de trabajo es el siguiente:

```
try {

   // Inicio de la Transacción
   $dbh->setAttribute(PDO::ATTR_AUTOCOMMIT,FALSE);
   $dbh->beginTransaction();

   // Otras instrucciones

   // Confirmar Transacción
   $dbh->commit();

} catch (PDOException $e) {

   // Cancelar transacción
   $dbh->rollBack();

   // Otras instrucciones de control

}
```

Ejemplo 1

En este ejemplo ilustraremos como programar una transacción controlada desde el cliente, el propósito es hacer una función que permita crear nuevas cuentas, el script del programa es el siguiente:

```
<?php

try {
   // Construyendo el Registro
   $rec["cliente"] = '00013';
   $rec["empl"] = '0008';
   $rec["importe"] = 1000.00;
   $rec["clave"] = '123456';
   $rec["moneda"] = '02';
   // Creando la Cuenta
   $cuenta = crearCuenta($rec);
   // Mostrando mensaje
   echo "Proceso ok.<br/>";
   echo "Cuenta Creada: $cuenta";
} catch( PDOException $e ) {
   echo 'Error: ' . $e->getMessage();
}

/*
 *  Estructura del Registro:
 *   $rec["cliente"] -> Código del cliente
 *   $rec["empl"] -> Código del empleado
 *   $rec["importe"] -> Importe de apertura
 *   $rec["clave"] -> Clave de la cuenta
 *   $rec["moneda"] -> Moneda
 *  Retorna el código de la cuenta
*/
function crearCuenta($rec) {
   $pdo = null;
   try {
      $dsn = 'mysql:host=localhost;dbname=eurekabank';
      $pdo = new PDO($dsn, 'root', '');
      $pdo->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
      $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
      $pdo->setAttribute(PDO::ATTR_CASE, PDO::CASE_LOWER);
      $pdo->setAttribute(PDO::ATTR_AUTOCOMMIT, FALSE);
      $pdo->beginTransaction();
      // Leer la sucursal del empleado
      $query = "select chr_sucucodigo from asignado
         where chr_emplcodigo = ? and dtt_asigfechabaja is null";
      $stm = $pdo->prepare($query);
      $stm->execute(array($rec["empl"]));
      if( $stm->rowCount() == 0 ){
```

```
            throw new PDOException("Empleado no es válido");
        }
        $row = $stm->fetch();
        $sucursal = $row['chr_sucucodigo'];
        // Generar el codigo de cuenta
        $query = "select int_sucucontcuenta as cont from sucursal
                where chr_sucucodigo = ?";
        $stm = $pdo->prepare($query);
        $stm->execute( array( $sucursal ) );
        $row = $stm->fetch();
        $cont = $row["cont"] + 1;
        $query = "update sucursal set int_sucucontcuenta = ?
                where chr_sucucodigo = ?";
        $stm = $pdo->prepare($query);
        $stm->execute( array( $cont, $sucursal ) );
        $codCuenta = $sucursal . str_pad("$cont", 5, "0", STR_PAD_LEFT);
        // Insertar Cuenta
        $query = "insert into cuenta(chr_cuencodigo,chr_monecodigo,chr_sucucodigo,
                chr_emplcreacuenta,chr_cliecodigo,dec_cuensaldo,dtt_cuenfechacreacion,
                vch_cuenestado,int_cuencontmov,chr_cuenclave) values(?,?,?,?,?,?,
                sysdate(),'ACTIVO',1,?)";
        $stm = $pdo->prepare($query);
        $stm->bindParam(1,$codCuenta);
        $stm->bindParam(2,$rec["moneda"]);
        $stm->bindParam(3,$sucursal);
        $stm->bindParam(4,$rec["empl"]);
        $stm->bindParam(5,$rec["cliente"]);
        $stm->bindParam(6,$rec["importe"]);
        $stm->bindParam(7,$rec["clave"]);
        $stm->execute();
        // Insertar Movimiento
        $query = "insert into movimiento(chr_cuencodigo,int_movinumero,
                dtt_movifecha,chr_emplcodigo,chr_tipocodigo,dec_moviimporte,
                chr_cuenreferencia) values(?,1,sysdate(),?,'001',?,null)";
        $stm = $pdo->prepare($query);
        $stm->bindParam(1,$codCuenta);
        $stm->bindParam(2,$rec["empl"]);
        $stm->bindParam(3,$rec["importe"]);
        $stm->execute();
        // Confirmar Transacción
        $pdo->commit();
        return $codCuenta;
    } catch ( PDOException $e ) {
        try {
            $pdo->rollBack();
        } catch (Exception $exc) {
        }
        throw $e;
    }
}
?>
```

La función recibe los datos en un registro, luego inicia su transacción, note usted que hay tres acciones que deben formar parte de la transacción:

1. Obtener e incrementar el contador de la sucursal, previamente se debe obtener el código de la sucursal según el código del empleado.
2. Generar el número de cuenta y registrar los datos de la cuenta.
3. Registrar el movimiento de apertura.

En caso de que alguna de las acciones genere un error se cancela la transacción.

El resultado obtenido es el siguiente;

```
Proceso ok.
Cuenta Creada: 00500001
```

3 Transacciones de Base de Datos

En este caso la transacción se programa en un procedimiento almacenado, su esquema de trabajo es el siguiente:

```
delimiter //

CREATE PROCEDURE nombre_procedimiento ( parámetros )
BEGIN

    -- Iniciar Transacción
    START TRANSACTION;

    -- Instrucciones INSERT, UPDATE ó DELETE

    -- Confirmar Transacción
    COMMIT;

END
//

delimiter;
```

Pero antes de ejecutar el procedimiento se debe establecer el atributo **ATTR_AUTOCOMMIT** en **TRUE**.

Ejemplo 2

En este ejemplo ilustraremos como programar una transacción en un procedimiento almacenado y luego como lo ejecutaremos desde PHP.

El siguiente procedimiento almacenado realiza el proceso **depósito** en la base de datos **EurekaBank**.

```
DELIMITER $$

DROP PROCEDURE IF EXISTS usp_deposito$$

CREATE PROCEDURE usp_deposito(
      p_cuenta char(8),
      p_importe decimal(12,2),
      p_empleado char(4)
)
BEGIN

      DECLARE moneda char(2);
      DECLARE costoMov decimal(12,2);
```

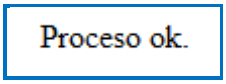
```
	DECLARE cont int;

	DECLARE EXIT HANDLER FOR SQLEXCEPTION, SQLWARNING, NOT FOUND
	BEGIN
		-- Cancela la transacción
		rollback;
		-- Retorna el estado
		select -1 as estado, 'Error en el proceso de actualización.' as
message;
	END;

	-- Iniciar Transacción
	start transaction;

	-- Tabla Cuenta
	select int_cuencontmov, chr_monecodigo into cont, moneda
	from cuenta where chr_cuencodigo = p_cuenta;

	select dec_costimporte into costoMov
	from costomovimiento
	where chr_monecodigo = moneda;

	-- Registrar el deposito
	update cuenta
		set dec_cuensaldo = dec_cuensaldo + p_importe - costoMov,
			int_cuencontmov = int_cuencontmov + 2
		where chr_cuencodigo = p_cuenta;

	-- Registra el movimiento
	set cont := cont + 1;
	insert into movimiento(chr_cuencodigo,int_movinumero,dtt_movifecha,
		chr_emplcodigo,chr_tipocodigo,dec_moviimporte,chr_cuenreferencia)
		values(p_cuenta,cont,current_date,p_empleado,'003',p_importe,null);

	-- Registra el costo del movimiento
	set cont := cont + 1;
	insert into movimiento(chr_cuencodigo,int_movinumero,dtt_movifecha,
		chr_emplcodigo,chr_tipocodigo,dec_moviimporte,chr_cuenreferencia)
		values(p_cuenta,cont,current_date,p_empleado,'010',costoMov,null);

	-- Confirma Transacción
	commit;

	-- Retorna el estado
	select 1 as estado, 'Proceso ok' as message;

END$$

DELIMITER ;
```

El script del programa es el siguiente:

```
<?php

try {
   // Construyendo el Registro
   $rec["cuenta"] = '00100002';
   $rec["importe"] = 1000.00;
   $rec["empl"] = '0001';
   // Realizar el Depósito
   $cuenta = realizarDeposito($rec);
   // Mostrando mensaje
   echo "Proceso ok.<br/>";
} catch( PDOException $e ) {
   echo $e->getMessage();
}

/*
 *  Estructura del Registro:
 *   $rec["cuenta"] -> Codigo de cuenta
 *   $rec["importe"] -> Importe del depósito
 *   $rec["empl"] -> Código del empleado
*  Retorna el estado de la Tx como un result set.
*/
function realizarDeposito($rec) {
   $pdo = null;
   try {
      $dsn = 'mysql:host=localhost;dbname=eurekabank';
      $pdo = new PDO($dsn, 'root', 'mysql');
      $pdo->setAttribute(PDO::ATTR_ERRMODE,PDO::ERRMODE_EXCEPTION);
      $pdo->setAttribute(PDO::ATTR_DEFAULT_FETCH_MODE, PDO::FETCH_ASSOC);
      $pdo->setAttribute(PDO::ATTR_CASE, PDO::CASE_LOWER);
      $pdo->setAttribute(PDO::ATTR_AUTOCOMMIT, TRUE);
      // Preparar la ejecución del procedimiento
      $query = "CALL usp_deposito(?,?,?)";
      $stm = $pdo->prepare($query);
      // Asignar los parámetros
      $stm->bindParam(1,$rec['cuenta']);
      $stm->bindParam(2,$rec['importe']);
      $stm->bindParam(3,$rec['empl']);
      // Ejecutar el procedimiento
      $stm->execute();
      // Obtener el estado
      $row = $stm->fetch();
      if( $row['state'] == -1 ) {
         throw new PDOException($row['message']);
      }
   } catch ( PDOException $e ) {
      throw $e;
   }
```

```
}

?>
```

Si ejecutamos el programa y no existe error en el procedimiento almacenado obtenemos el siguiente resultado:



De haber algún error obtenemos el siguiente resultado.