Procedimientos Almacenados en MySQL

Procedimientos Almacenados: Introducción.

Comenzando con los procedimientos almacenados.

La siguiente instrucción SELECT devuelve todas las filas en la tabla clientes de la base de datos de muestra:

```
SELECT
    customerName,
    city,
    state,
    postalCode,
    country
FROM
    customers
ORDER BY customerName;
```

Cuando utiliza MySQL Workbench o mysql shell para enviar la consulta al servidor MySQL, MySQL procesa la consulta y devuelve el conjunto de resultados.
Si desea guardar esta consulta en el servidor de la base de datos para su ejecución posterior, una forma de hacerlo es usar un procedimiento almacenado.
La siguiente instrucción CREATE PROCEDURE crea un nuevo procedimiento almacenado que envuelve la consulta anterior:

```
DELIMITER $$

CREATE PROCEDURE GetCustomers()
BEGIN
    SELECT
        customerName,
        city,
        state,
        postalCode,
        country
    FROM
        customers
    ORDER BY customerName;
END$$
DELIMITER ;
```

Por definición, un procedimiento almacenado es un segmento de declaraciones declarativas SQL almacenadas dentro del servidor MySQL. En este ejemplo, acabamos de crear un procedimiento almacenado con el nombre GetCustomers().
Una vez que guarde el procedimiento almacenado, puede invocarlo utilizando la instrucción CALL:

```
CALL GetCustomers ();
```

Y el procedimiento devuelve el mismo resultado que la consulta.
La primera vez que invoca un procedimiento almacenado, MySQL busca el nombre en el catálogo de la base de datos, compila el código del procedimiento almacenado, lo coloca en un área de memoria conocida como caché y ejecuta el procedimiento almacenado.
Si vuelve a invocar el mismo procedimiento almacenado en la misma sesión, MySQL simplemente ejecuta el procedimiento almacenado desde el caché sin tener que volver a compilarlo.
Un procedimiento almacenado puede tener parámetros para que pueda pasarle valores y recuperar el resultado. Por ejemplo, puede tener un procedimiento almacenado que devuelva clientes por país y ciudad. En este caso, el país y la ciudad son parámetros del procedimiento almacenado.
Un procedimiento almacenado puede contener sentencias de flujo de control como IF, CASE y LOOP que le permiten implementar el código de forma procesal.
Un procedimiento almacenado puede llamar a otros procedimientos almacenados o funciones almacenadas, lo que le permite modificar su código.
Tenga en cuenta que aprenderá paso a paso cómo crear un nuevo procedimiento almacenado en el siguiente tutorial.

Ventajas de los procedimientos almacenados de MySQL.

Las siguientes son las ventajas de los procedimientos almacenados.

Reduce el tráfico de red.

Los procedimientos almacenados ayudan a reducir el tráfico de red entre las aplicaciones y el servidor MySQL. Porque en lugar de enviar múltiples sentencias SQL largas, las aplicaciones tienen que enviar solo el nombre y los parámetros de los procedimientos almacenados.

Centraliza la lógica de negocios en la base de datos.

Puede usar los procedimientos almacenados para implementar una lógica de negocios que sea reutilizable por múltiples aplicaciones. Los procedimientos almacenados ayudan a reducir los esfuerzos de duplicar la misma lógica en muchas aplicaciones y hacer que su base de datos sea más consistente.

Hacer la base de datos más segura.

El administrador de la base de datos puede otorgar los privilegios apropiados a las aplicaciones que solo acceden a procedimientos almacenados específicos sin otorgar ningún privilegio en las tablas subyacentes.

Desventajas de los procedimientos almacenados de MySQL.

Además de esas ventajas, los procedimientos almacenados también tienen desventajas:

Usos de recursos.

Si utiliza muchos procedimientos almacenados, el uso de memoria de cada conexión aumentará sustancialmente.
Además, el uso excesivo de una gran cantidad de operaciones lógicas en los procedimientos almacenados aumentará el uso de la CPU porque MySQL no está bien diseñado para operaciones lógicas.

Solución de problemas.

Es difícil depurar los procedimientos almacenados. Desafortunadamente, MySQL no proporciona ningún elementos para depurar procedimientos almacenados como otros productos de bases de datos empresariales como Oracle y SQL Server. Aunque existen soluciones de terceros como DBForge y posiblemente Oracle incorpore algo a las versiones de pago de MySQL.

Mantenimiento.

Desarrollar y mantener procedimientos almacenados a menudo requiere un conjunto de habilidades especializadas que no todos los desarrolladores de aplicaciones poseen. Esto puede generar problemas tanto en el desarrollo de aplicaciones como en el mantenimiento.

Creando procedimientos almacenados.

MySQL CREATE PROCEDURE

Esta consulta devuelve todos los productos en la tabla de products de la base de datos de muestra
.

```
SELECT * FROM products;
```

La siguiente instrucción crea un nuevo procedimiento almacenado que envuelve la consulta:

```
DELIMITER //

CREATE PROCEDURE GetAllProducts()
BEGIN
    SELECT * FROM products;
END //

DELIMITER ;
```

Para ejecutar estas declaraciones:

- Primero, inicie MySQL Workbench.
- En segundo lugar, cree una nueva pestaña SQL para ejecutar consultas.
- Tercero, escriba el procedimiento en la pestaña SQL.
- Cuarto ejecuta el script. Tenga en cuenta que puede seleccionar todas las declaraciones en la pestaña SQL (o nada) y hacer clic en el botón Ejecutar. Si todo está bien, MySQL creará el procedimiento almacenado y lo guardará en el servidor.

- Quinto, verifique el procedimiento almacenado abriendo el nodo Procedimientos almacenados . Si no ve el procedimiento almacenado, puede hacer clic en el botón Actualizar al lado del título SCHEMAS.

Enhorabuena! Ha creado con éxito el primer procedimiento almacenado en MySQL.
Examinemos la sintaxis del procedimiento almacenado.
El primer y último comando es DELIMITER, no son parte del procedimiento almacenado. El primer comando DELIMITER cambia el delimitador predeterminado a // y el último comando DELIMITER vuelve a cambiar el delimitador al predeterminado que es punto y coma (;).
Para crear un nuevo procedimiento almacenado, use la instrucción CREATE PROCEDURE .

```
CREATE   PROCEDURE  procedure_name(parameter_list)
BEGIN
  statements;
END  //
```

En esta instrucción

- Primero, especificamos el nombre del procedimiento almacenado que desea crear después de las palabras clave CREATE PROCEDURE.
- En segundo lugar, especificamons una lista de parámetros separados por comas entre paréntesis, después del nombre del procedimiento.
- Tercero, escriba el código entre el bloque BEGIN END. El ejemplo anterior solo tiene una simple instrucción SELECT. Después de la palabra clave END, coloca el carácter delimitador para finalizar la instrucción del procedimiento.

Ejecutar un procedimiento almacenado

Para ejecutar un procedimiento almacenado, use la instrucción CALL :

```
CALL  stored_procedure_name(argument_list);
```

En esta sintaxis, especifica el nombre del procedimiento almacenado después de la palabra clave CALL. Si el procedimiento almacenado tiene parámetros, debe pasar argumentos entre paréntesis después del nombre del procedimiento almacenado.
Este ejemplo ilustra cómo llamar al procedimiento almacenado GetAllProducts() :

```
CALL  GetAllProducts();
```

Ejecutar el procedimiento es lo mismo que ejecutar una sentencia SQL.

Crear un procedimiento almacenado utilizando el asistente MySQL Workbench

Al usar el asistente MySQL Workbench, no tiene que tomar muchas cosas como delimitadores o ejecutar el comando para crear procedimientos almacenados.

- Primero, haga clic con el botón derecho en los procedimientos almacenados desde el navegador y seleccione el elemento de menú Crear procedimiento almacenado... Se abrirá la siguiente una pestaña.

- En segundo lugar, cambie el nombre del procedimiento almacenado y agregue el código entre el bloque BEGIN END: El nombre del procedimiento almacenado es GetAllCustomers() que devuelve todas las filas en la tabla de customers de la base de datos de muestra.
- Tercero, haga clic en el botón Aplicar , MySQL Workbench abrirá una nueva ventana para revisar el script SQL antes de aplicarlo en la base de datos.
- Cuarto, haga clic en el botón Aplicar para confirmar. MySQL Workbench creará el procedimiento almacenado.
- Quinto, haga clic en el botón Finalizar para cerrar la ventana.
- Finalmente, vea el procedimiento almacenado en la lista Procedimientos almacenados.

DELIMITER

Un procedimiento almacenado generalmente contiene varias declaraciones separadas por punto y coma (;). Para usar compilar todo el procedimiento almacenado como una sola declaración compuesta, debe cambiar temporalmente el delimitador del punto y coma (;) a delimitadores de anteras como $$ o //.

DROP PROCEDURE MySQL

DROP PROCEDURE elimina un procedimiento almacenado de la base de datos.
A continuación se muestra la sintaxis de la instrucción DROP PROCEDURE :

```
DROP   PROCEDURE  [ IF EXISTS ] stored_procedure_name;
```

En esta sintaxis:

- Primero, especifique el nombre del procedimiento almacenado que desea eliminar después de las palabras clave DROP PROCEDURE .
- En segundo lugar, use la opción IF EXISTS para descartar condicionalmente el procedimiento almacenado si solo existe.

Cuando s descarta un procedimiento que no existe sin usar la opción IF EXISTS, MySQL emite un error. En este caso, si usa la opción IF EXISTS, MySQL emite una advertencia en su lugar.
Tenga en cuenta que debe tener el privilegio ALTER ROUTINE para que el procedimiento almacenado lo elimine. Ejemplos de DROP PROCEDURE MySQL

Algunos ejemplos del uso de la instrucción DROP PROCEDURE .

1. Usando MySQL DROP PROCEDURE ejemplo
    - Primero, cree un nuevo procedimiento almacenado que devuelva información de empleados y oficina:

```
DELIMITER  $$

CREATE   PROCEDURE  GetEmployees()
BEGIN
    SELECT
       firstName,
       lastName,
       city,
       state,
       country
    FROM  employees
    INNER JOIN  offices  using  (officeCode);

END $$

DELIMITER  ;
```

- En segundo lugar, use el DROP PROCEDURE para eliminar el procedimiento almacenado GetEmployees() :

```
DROP  PROCEDURE  GetEmployees;
```

1. Usando MySQL DROP PROCEDURE con IF EXISTS ejemplo

El siguiente ejemplo descarta un procedimiento almacenado que no existe:

```
DROP   PROCEDURE  abc;
```

MySQL muestra el siguiente error:

- Error Code: 1305. PROCEDURE classicmodels.abc does not exist

, pero con la opción IF EXISTS :

```
DROP   PROCEDURE   IF EXISTS  abc;
```

Esta vez, MySQL emitió una advertencia. 0 row (s) affected, 1 warning(s): 1305 PROCEDURE classicmodels.abc does not exist
La declaración SHOW WARNINGS muestra la advertencia:

```
SHOW   WARNINGS ;
```

Descartar un procedimiento almacenado usando MySQL Workbench

Esta declaración crea un nuevo procedimiento almacenado llamado GetPayments() :

```
DELIMITER  $$

CREATE   PROCEDURE  GetPayments()
BEGIN
```

```
    SELECT
       customerName,
       checkNumber,
       paymentDate,
       amount
    FROM  payments
    INNER JOIN  customers
        using  (customerNumber);
END $$

DELIMITER  ;
```

- Para descartar el procedimiento almacenado usando MySQL Workbench, siga estos pasos:
- Primero, haga clic con el botón derecho en el nombre del procedimiento almacenado que desea eliminar y elija Descartar procedimiento almacenado ... opción. MySQL Workbench mostrará una ventana de confirmación.
- En segundo lugar, haga clic en Revisar SQL para revisar la declaración SQL que MySQL Workbench aplicará a la base de datos o Soltar ahora si desea eliminar el procedimiento almacenado de inmediato.
- Tercero, revise el código SQL que se ejecutará y haga clic en el botón Ejecutar para descartar el procedimiento almacenado.

Introducción a los parámetros de procedimiento almacenado de MySQL

Los procedimientos casi almacenados que desarrolla requieren parámetros. Los parámetros hacen que el procedimiento almacenado sea más flexible y útil.

Modos de los parámetros de los procedimientos almacenados en MySQL

En MySQL, un parámetro tiene uno de los tres modos: IN, OUT o INOUT.

Parámetros IN.

IN es el modo predeterminado. Cuando define un parámetro IN en un procedimiento almacenado, el programa que realiza la llamada debe pasar un argumento al procedimiento almacenado. Además, el valor de un parámetro IN está protegido. Significa que incluso el valor del parámetro IN se cambia dentro del procedimiento almacenado, su valor original se retiene después de que finaliza el procedimiento almacenado. En otras palabras, el procedimiento almacenado solo funciona en la copia del parámetro IN.

Parámetros OUT

El valor de un parámetro OUT se puede cambiar dentro del procedimiento almacenado y su nuevo valor se devuelve al programa de llamada. Observe que el procedimiento almacenado no puede acceder al valor inicial del parámetro OUT cuando se inicia.

Parámetros INOUT

Un parámetro INOUT es una combinación de parámetros IN y OUT. Significa que el programa que realiza la llamada puede pasar el argumento, y el procedimiento almacenado puede modificar el parámetro INOUT y devolver el nuevo valor al programa que realiza la llamada. Definiendo un parámetro
Aquí está la sintaxis básica de definir un parámetro en procedimientos almacenados:

```
[ IN  |  OUT  |  INOUT ] parameter_name datatype[( length )]
```

En esta sintaxis,

- Primero, especifique el modo de parámetro, que puede ser IN , OUT o INOUT , según el propósito del parámetro en el procedimiento almacenado.
- En segundo lugar, especifique el nombre del parámetro. El nombre del parámetro debe seguir las reglas de nomenclatura del nombre de la columna en MySQL.
- Tercero, especifique el tipo de datos y la longitud máxima del parámetro.

Ejemplos de parámetros de procedimientos almacenados de MySQL

Algunos ejemplos del uso de parámetros de procedimientos almacenados.

- Ejemplo del parámetro IN

  El siguiente ejemplo crea un procedimiento almacenado que busca todas las oficinas que se encuentran en un país especificado por el parámetro de entrada countryName :

```
DELIMITER //

CREATE PROCEDURE GetOfficeByCountry(
    IN countryName VARCHAR (255)
)
BEGIN
    SELECT *
     FROM offices
    WHERE country = countryName;
END //

DELIMITER ;
```

  En este ejemplo, countryName es el parámetro IN del procedimiento almacenado.

  Suponga que desea encontrar oficinas ubicadas en los USA. Necesita pasar un argumento (USA) al procedimiento almacenado como se muestra en la siguiente consulta:

```
CALL GetOfficeByCountry( 'USA' );
```

  Para buscar oficinas en France , pase la cadena literal France al procedimiento almacenado GetOfficeByCountry siguiente manera:

```
CALL GetOfficeByCountry( 'France' )
```

Como countryName es el parámetro IN, debe pasar un argumento. De lo contrario, se producirá un error:

```
CALL  GetOfficeByCountry();
```

Aquí está el error:

```
- Error Code: 1318. Incorrect number of arguments for PROCEDURE
  classicmodels.GetOfficeByCountry; expected 1, got 0
```

Ejemplo de parámetro OUT.

El siguiente procedimiento almacenado devuelve el número de pedidos por estado de pedido.

```
DELIMITER  $$

CREATE PROCEDURE  GetOrderCountByStatus (
    IN  orderStatus VARCHAR (25),
    OUT total INT
)
BEGIN
    SELECT COUNT (orderNumber)
    INTO  total
    FROM  orders
    WHERE status = orderStatus;
END $$

DELIMITER  ;
```

El procedimiento almacenado GetOrderCountByStatus() tiene dos parámetros:

- orderStatus: es el parámetro IN que especifica el estado de las órdenes a devolver.
- total: es el parámetro OUT que almacena el número de pedidos en un estado específico.

Para encontrar la cantidad de pedidos que ya se enviaron, llame a GetOrderCountByStatus y pase el estado del pedido a partir de Shipped , y también pase una variable de sesión (@total) para recibir el valor devuelto.

```
CALL GetOrderCountByStatus('Shipped' ,@total);
SELECT @total;
```

Para obtener el número de pedidos que están en proceso, llame al procedimiento almacenado GetOrderCountByStatus siguiente manera:

```
CALL GetOrderCountByStatus('in process',@total);
SELECT @total AS total_in_process;
```

Ejemplo de parámetro INOUT.

El siguiente ejemplo muestra cómo usar un parámetro INOUT en el procedimiento almacenado.

```
 DELIMITER  $$

 CREATE PROCEDURE SetCounter(
     INOUT counter INT ,
      IN inc INT
)
 BEGIN
     SET counter = counter + inc;
 END $$

 DELIMITER  ;
```

En este ejemplo, el procedimiento almacenado SetCounter() acepta un parámetro INOUT (counter) y un parámetro IN (inc). Aumenta el contador (counter) en el valor especificado por el parámetro inc .
Estas declaraciones ilustran cómo llamar al procedimiento almacenado SetSounter:

```
 SET  @counter = 1;
CALL  SetCounter(@counter,1);  -- 2
CALL  SetCounter(@counter,1);  -- 3
CALL  SetCounter(@counter,5);  -- 8
SELECT  @counter;  -- 8
```

Modificar un procedimiento almacenado.

A veces, es posible que desee modificar un procedimiento almacenado agregando o eliminando parámetros o incluso cambiando su cuerpo.
Desafortunadamente, MySQL no tiene ninguna declaración que le permita modificar directamente los parámetros y el cuerpo del procedimiento almacenado.
Para realizar dichos cambios, debe eliminar el procedimiento almacenado y volver a crearlo utilizando las instrucciones DROP PROCEDURE y CREATE PROCEDURE.

Modificar un procedimiento almacenado usando MySQL Workbench

MySQL Workbench le proporciona una buena herramienta que le permite cambiar rápidamente un procedimiento almacenado.

- Primero, cree un procedimiento almacenado que devuelva el importe total de todos los pedidos de ventas:

```
DELIMITER  $$

CREATE   PROCEDURE  GetOrderAmount()
BEGIN
    SELECT
        SUM (quantityOrdered * priceEach)
```

```
    FROM  orderDetails;
END $$

DELIMITER  ;
```

Suponga que desea obtener el importe total de un pedido de ventas determinado. Por lo tanto, debe agregar un parámetro y cambiar el código en el procedimiento almacenado.

- En segundo lugar, haga clic con el botón derecho en el procedimiento almacenado que desea cambiar y seleccione Alterar procedimiento almacenado ... MySQL Workbench abrirá una nueva pestaña que contiene la definición del procedimiento almacenado.
- Tercero, realice los cambios y haga clic en el botón Aplicar. MySQL Workbench mostrará una ventana de revisión de SQL Script. Como puede ver, utiliza una secuencia de instrucciones DROP PROCEDURE y CREATE PROCEDURE para llevar la modificación.
- Cuarto, haga clic en el botón Aplicar para ejecutar el script. MySQL Workbench mostrará una ventana que muestra el estado de la ejecución del script.
- Finalmente, haga clic en el botón Finalizar para completar el cambio.

Variables en procedimientos almacenados.

Una variable es un objeto de datos con nombre cuyo valor puede cambiar durante la ejecución del procedimiento almacenado . Por lo general, utiliza variables en procedimientos almacenados para mantener resultados inmediatos. Estas variables son locales para el procedimiento almacenado. Antes de usar una variable, debe declararla.

Declarando variables

Para declarar una variable dentro de un procedimiento almacenado, use la declaración DECLARE siguiente manera:
DECLARE variable_name datatype(size) [DEFAULT default_value];
En esta sintaxis:

- Primero, especifique el nombre de la variable después de la palabra clave DECLARE . El nombre de la variable debe seguir las reglas de nomenclatura de los nombres de columna de la tabla MySQL.
- En segundo lugar, especifique el tipo de datos y la longitud de la variable. Una variable puede tener cualquier tipo de datos MySQL como INT , VARCHAR y DATETIME .
- Tercero, asigne un valor predeterminado a una variable usando la opción DEFAULT . Si declara una variable sin especificar un valor predeterminado, su valor es NULL .

El siguiente ejemplo declara una variable llamada totalSale con el tipo de datos DEC(10,2) y el valor predeterminado 0.0 siguiente manera:

```
DECLARE  totalSale  DEC (10,2)  DEFAULT  0.0;
```

MySQL le permite declarar dos o más variables que comparten el mismo tipo de datos usando una sola declaración DECLARE . El siguiente ejemplo declara dos variables enteras x e y , y establece sus valores predeterminados a cero.

```
DECLARE  x , y  INT  DEFAULT 0;
```

Asignando variables

Una vez que se declara una variable, está lista para usar. Para asignar un valor a una variable, usa la instrucción SET :

```
SET variable_name = value ;
```

Por ejemplo:

```
DECLARE total INT  DEFAULT 0;
SET total = 10;
```

El valor de la variable total es 10 después de la asignación.
Además de la instrucción SET, puede usar la SELECT INTO para asignar el resultado de una consulta a una variable como se muestra en el siguiente ejemplo:

```
DECLARE productCount INT  DEFAULT 0;

SELECT COUNT (*)
INTO productCount
FROM products;
```

En este ejemplo:

- Primero, declare una variable llamada productCount e inicialice su valor a 0.
- Luego, use la SELECT INTO para asignar a la variable productCount el número de productos seleccionados de la tabla de products .

Ámbito (alcance) de las variables

Una variable tiene su propio ámbito que define su vida útil. Si declara una variable dentro de un procedimiento almacenado, estará fuera del alcance cuando llegue la instrucción END del procedimiento almacenado.
Cuando declara una variable dentro del bloque BEGIN END, estará fuera del alcance si se alcanza el END.
MySQL le permite declarar dos o más variables que comparten el mismo nombre en diferentes ámbitos. Porque una variable solo es efectiva en su alcance. Sin embargo, declarar variables con el mismo nombre en diferentes ámbitos no es una buena práctica de programación.
Una variable cuyo nombre comienza con el signo @ es una variable de sesión. Está disponible y accesible hasta que finalice la sesión.

Poniéndolo todo junto

El siguiente ejemplo ilustra cómo declarar y usar una variable en un procedimiento almacenado:

```
DELIMITER  $$

CREATE   PROCEDURE  GetTotalOrder()
BEGIN
    DECLARE  totalOrder  INT   DEFAULT  0;

    SELECT COUNT (*)
    INTO  totalOrder
    FROM  orders;

    SELECT  totalOrder;
END $$

DELIMITER  ;
```

Cómo funciona.

- Primero, declare una variable totalOrder con un valor predeterminado de cero. Esta variable contendrá el número de pedidos de la tabla de orders.

```
DECLARE totalOrder INT DEFAULT 0;
```

- En segundo lugar, use la SELECT INTO para asignar a la variable total totalOrder el número de pedidos seleccionados de la tabla de orders:

```
SELECT COUNT (*)
INTO  totalOrder
FROM  orders;
```

- Tercero, seleccione el valor de la variable totalOrder .

```
SELECT  totalOrder;
```

Esta declaración llama al procedimiento almacenado GetTotalOrder() :

```
CALL  GetTotalOrder();
```

Ver los procedimientos almacenados en la base de datos.

Listar los procedimientos almacenados utilizando el diccionario de datos

La tabla de routines en la base de datos information_schema contiene toda la información sobre los procedimientos almacenados y las funciones almacenadas de todas las bases de datos en el servidor MySQL actual.
Para mostrar todos los procedimientos almacenados de una base de datos particular, utilice la siguiente consulta:

```
SELECT
   routine_name
FROM
   information_schema.routines
WHERE
   routine_type =  'PROCEDURE'
        AND  routine_schema =  '<database_name>' ;
```

Por ejemplo, esta declaración enumera todos los procedimientos almacenados en la base de datos de classicmodels :

```
SELECT
   routine_name
FROM
   information_schema.routines
WHERE
   routine_type =  'PROCEDURE'
        AND  routine_schema =  'classicmodels' ;
```

Mostrar procedimientos almacenados utilizando MySQL Workbench

En MySQL Workbench, puede ver todos los procedimientos almacenados desde una base de datos.

- Paso 1. Acceda a la base de datos que desea ver los procedimientos almacenados.
- Paso 2. Abra el menú Procedimientos almacenados. Verá una lista de procedimientos almacenados que pertenecen a la base de datos actual.

Procedimientos almacenados de MySQL que devuelven múltiples valores.

Las funciones almacenadas de MySQL devuelven solo un valor. Para desarrollar programas almacenados que devuelven múltiples valores, debe usar procedimientos almacenados con parámetros INOUT u OUT .
Si no está familiarizado con los parámetros INOUT u OUT, consulte el apartado de parámetros en procedimientos almacenados para obtener información detallada.

- Ejemplo con parámetro OUT.
- Ejemplo con parámetro INOUT.

Ejemplo de procedimientos almacenados que devuelven valores múltiples

Echemos un vistazo a la tabla de orders en la base de datos de muestra.

```
describe orders;
```

El siguiente procedimiento almacenado acepta el número de cliente y devuelve el número total de pedidos que fueron enviados, cancelados, resueltos y disputados.

```
DELIMITER  $$

CREATE   PROCEDURE  get_order_by_cust(
     IN  cust_no  INT ,
    OUT  shipped  INT ,
    OUT  canceled  INT ,
    OUT  resolved  INT ,
    OUT  disputed  INT )
BEGIN
        -- shipped
        SELECT
            count (*) INTO shipped
        FROM
           orders
        WHERE
           customerNumber = cust_no
           AND status = 'Shipped' ;

        -- canceled
        SELECT
            count (*) INTO canceled
        FROM
           orders
        WHERE
           customerNumber = cust_no
           AND status = 'Canceled' ;

        -- resolved
        SELECT
            count (*) INTO resolved
        FROM
           orders
        WHERE
           customerNumber = cust_no
           AND status = 'Resolved' ;

        -- disputed
        SELECT
            count (*) INTO disputed
        FROM
           orders
        WHERE
           customerNumber = cust_no
           AND status = 'Disputed' ;

END
```

Además del parámetro IN, el procedimiento almacenado toma cuatro parámetros OUT adicionales: enviado, cancelado, resuelto y disputado. Dentro del procedimiento almacenado, utiliza una instrucción SELECT con la función COUNT para obtener el total correspondiente de pedidos en función del estado del pedido y asignarlo al parámetro respectivo.

Para utilizar el procedimiento almacenado get_order_by_cust , pasa el número de cliente y cuatro variables definidas por el usuario para obtener los valores de salida.
Después de ejecutar el procedimiento almacenado, utiliza la instrucción SELECT para generar los valores de las variables.

```
CALL  get_order_by_cust(141,@shipped,@canceled,@resolved,@disputed);
SELECT  @shipped,@canceled,@resolved,@disputed;
```

Procedimientos almacenados de MySQL que devuelven múltiples valores

Llamar a procedimientos almacenados que devuelven múltiples valores desde PHP

El siguiente fragmento de código le muestra cómo llamar al procedimiento almacenado que devuelve múltiples valores de PHP.

```
<?php
/**
 * Call stored procedure that return multiple values
 * @param $customerNumber
 */
function call_sp($customerNumber)
{
    try
    {
        $pdo =  new PDO("mysql:host=localhost;dbname=classicmodels",'root','');

        // execute the stored procedure
        $sql='CALL get_order_by_cust(:no,@shipped,@canceled,@resolved,@disputed)';
        $stmt=$pdo->prepare($sql) ;

        $stmt->bindParam(':no',$customerNumber,PDO::PARAM_INT);
        $stmt->execute() ;
        $stmt->closeCursor() ;

        // execute the second query to get values from OUT parameter
        $r = $pdo->query("SELECT @shipped,@canceled,@resolved,@disputed")
                 ->fetch(PDO::FETCH_ASSOC) ;
        if ($r)
        {
            printf ('Shipped: %d, Canceled: %d, Resolved: %d, Disputed: %d',
                $r ['@shipped'],
                $r ['@canceled'],
                $r ['@resolved'],
                $r ['@disputed']);
        }
    }
    catch   (PDOException $pe)
    {
        die ("Error occurred:".$pe->getMessage());
    }
```

```
}
call_sp (141) ;
```

Las variables definidas por el usuario, precedidas por el signo @, están asociadas con la conexión de la base de datos, por lo tanto, están disponibles para el acceso entre las llamadas.