Procedimientos almacenados

Crear Procedimientos Almacenados

Un procedimiento almacenado de SQL Server es un grupo de una o varias instrucciones Transact-SQL o una referencia a un método de Common Runtime Language (CLR) de Microsoft .NET Framework . Los procedimientos se asemejan a las construcciones de otros lenguajes de programación, porque pueden:

- Aceptar parámetros de entrada y devolver varios valores en forma de parámetros de salida al programa que realiza la llamada.
- Contener instrucciones de programación que realicen operaciones en la base de datos. Entre otras, pueden contener llamadas a otros procedimientos.
- Devolver un valor de estado a un programa que realiza una llamada para indicar si la operación se ha realizado correctamente o se han producido errores, y el motivo de estos.

Ventajas de usar procedimientos almacenados

En la siguiente lista se describen algunas de las ventajas que brinda el uso de procedimientos.
Tráfico de red reducido entre el cliente y el servidor
Los comandos de un procedimiento se ejecutan en un único lote de código. Esto puede reducir significativamente el tráfico de red entre el servidor y el cliente porque únicamente se envía a través de la red la llamada que va a ejecutar el procedimiento. Sin la encapsulación de código que proporciona un procedimiento, cada una de las líneas de código tendría que enviarse a través de la red.

Mayor seguridad

Varios usuarios y programas cliente pueden realizar operaciones en los objetos de base de datos subyacentes a través de un procedimiento, aunque los usuarios y los programas no tengan permisos directos sobre esos objetos subyacentes. El procedimiento controla qué procesos y actividades se llevan a cabo y protege los objetos de base de datos subyacentes. Esto elimina la necesidad de conceder permisos en cada nivel de objetos y simplifica los niveles de seguridad.
La cláusula EXECUTE AS puede especificarse en la instrucción CREATE PROCEDURE para habilitar la suplantación de otro usuario o para permitir que los usuarios o las aplicaciones puedan realizar ciertas actividades en la base de datos sin necesidad de contar con permisos directos sobre los objetos y comandos subyacentes. Por ejemplo, algunas acciones como TRUNCATE TABLE no tienen permisos que se puedan conceder. Para poder ejecutar TRUNCATE TABLE, el usuario debe tener permisos ALTER en la tabla especificada. Puede que la concesión de permisos ALTER a un usuario en una tabla no sea lo ideal, pues en realidad el usuario tendrá permisos muy superiores a la posibilidad de truncar una tabla. Si se incorpora la instrucción TRUNCATE TABLE en un módulo y se especifica la ejecución del módulo como un usuario con permisos para modificar la tabla, se pueden ampliar los permisos para truncar la tabla al usuario al que se concedan permisos EXECUTE para el módulo.
Al llamar a un procedimiento a través de la red, solo está visible la llamada que va a ejecutar el procedimiento. Por lo tanto, los usuarios malintencionados no pueden ver los nombres de los objetos de base de datos y tabla, incrustados en sus propias instrucciones Transact-SQL , ni buscar datos críticos.
El uso de parámetros de procedimientos ayuda a protegerse contra ataques por inyección de código SQL. Dado que la entrada de parámetros se trata como un valor literal y no como código ejecutable, resulta más difícil para un atacante insertar un comando en la instrucción Transact-SQL del procedimiento y comprometer la seguridad.Los procedimientos pueden cifrarse, lo que ayuda a ofuscar el código fuente. Para más información, consulte SQL Server Encryption.

Reutilización del código

El código de cualquier operación de base de datos redundante resulta un candidato perfecto para la encapsulación de procedimientos. De este modo, se elimina la necesidad de escribir de nuevo el mismo código, se reducen las inconsistencias de código y se permite que cualquier usuario o aplicación que cuente con los permisos necesarios pueda acceder al código y ejecutarlo.
Mantenimiento más sencillo
Cuando las aplicaciones cliente llaman a procedimientos y mantienen las operaciones de base de datos en la capa de datos, solo deben actualizarse los cambios de los procesos en la base de datos subyacente. El nivel de aplicación permanece independiente y no tiene que tener conocimiento sobre los cambios realizados en los diseños, las relaciones o los procesos de la base de datos.
Rendimiento mejorado
De forma predeterminada, un procedimiento se compila la primera vez que se ejecuta y crea un plan de ejecución que vuelve a usarse en posteriores ejecuciones. Como el procesador de consultas no tiene que crear un nuevo plan, normalmente necesita menos tiempo para procesar el procedimiento.
Si ha habido cambios importantes en las tablas o datos a los que se hace referencia en el procedimiento, el plan precompilado podría hacer que el procedimiento se ejecutará con mayor lentitud. En este caso, volver a crear el procedimiento y forzar un nuevo plan de ejecución puede mejorar el rendimiento.

Sentencia EXECUTE

Ejecuta uno de los siguientes módulos: procedimiento almacenado del sistema, procedimiento almacenado definido por el usuario, función con valores escalares definida por el usuario o procedimiento almacenado extendido.
Ejecuta una cadena de comandos o una cadena de caracteres dentro de un proceso por lotes de Transact-SQL.

El siguiente ejemplo ejecuta una consulta sql simple

Sintaxis

```
EXEC ('SELECT * FROM Production.Product');
```

Tipos de procedimientos almacenados

Definidos por el usuario

Un procedimiento definido por el usuario se puede crear en una base de datos definida por el usuario o en todas las bases de datos del sistema excepto en la base de datos Resource El procedimiento se puede desarrollar en Transact-SQL o como una referencia a un método de Common Runtime Language (CLR) de Microsoft .NET Framework .

Temporales

Los procedimientos temporales son una forma de procedimientos definidos por el usuario. Los procedimientos temporales son iguales que los procedimientos permanentes salvo porque se almacenan en tempdb. Hay dos tipos de procedimientos temporales: locales y globales. Se diferencian entre sí por los nombres, la visibilidad y la disponibilidad. Los procedimientos temporales locales tienen como primer carácter de sus nombres un solo signo de número (#); solo son visibles en la conexión actual del usuario y se eliminan cuando se cierra la conexión. Los procedimientos temporales globales presentan dos signos de número (##) antes del nombre; son visibles para cualquier usuario después de su creación y se eliminan al final de la última sesión en la que se usa el procedimiento.

Sistema

Los procedimientos del sistema se incluyen con SQL Server. Están almacenados físicamente en la base de datos interna y oculta Resource y se muestran de forma lógica en el esquema sys de cada base de datos definida por el sistema y por el usuario. Además, la base de datos msdb también contiene procedimientos almacenados del sistema en el esquema dbo que se usan para programar alertas y trabajos. Dado que los procedimientos del sistema empiezan con el prefijo sp_, le recomendamos que no use este prefijo cuando asigne un nombre a los procedimientos definidos por el usuario. Para obtener una lista completa de los procedimientos del sistema.

sp_tables
sp_columns
sp_databases
sp_execute

SQL Server admite los procedimientos del sistema que proporcionan una interfaz de SQL Server a los programas externos para varias actividades de mantenimiento. Estos procedimientos extendidos usan el prefijo xp_. Para obtener una lista completa de los procedimientos extendidos.

xp_cmdshell

Crear un procedimiento almacenado simple
El siguiente ejemplo crea un procedimiento almacenado llamado PA_Empleados que obtiene un listado con el Apellido, Nombre y Departamento de los empleados.

Sintaxis

```
IF OBJECT_ID ( 'dbo.PA_Empleados', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.PA_Empleados;
GO

CREATE PROCEDURE dbo.PA_Empleados
AS
    SELECT LastName, FirstName, Department
    FROM   HumanResources.vEmployeeDepartmentHistory;
GO

-- LLAMADO O INVOCACIÓN
EXEC dbo.PA_Empleados;
GO
```

Crear un procedimiento con parámetros de entrada.

El siguiente ejemplo crea un procedimiento almacenado llamado PA_Empleados que obtiene un listado con el Apellido, Nombre y Departamento de los empleados filtrando por Apellido y Nombre.

Sintaxis

```
IF OBJECT_ID ('dbo.PA_Empleados', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.PA_Empleados;
GO

CREATE PROCEDURE dbo.PA_Empleados
    @LastName   VARCHAR(50),
    @FirstName  VARCHAR(50)
AS
    SELECT LastName, FirstName, Department
    FROM   HumanResources.vEmployeeDepartmentHistory
    WHERE  FirstName = @FirstName AND LastName = @LastName;
GO
-- Invocación al SP pasando los parámetros teniendo en cuenta el orden
EXEC dbo.PA_Empleados 'Ackerman', 'Pilar';

-- Invocación al SP pasando los parámetros sin tener en cuenta el orden
EXEC dbo.PA_Empleados  @FirstName = 'Pilar', @LastName = 'Ackerman';
```

Crear un procedimiento con parámetros de entrada predeterminados.

El siguiente ejemplo crea un procedimiento almacenado llamado PA_Empleados que obtiene un listado con datos del empleado cuyo Apellido y Nombre son pasados como valores de parámetros, por otro lado el procedimiento no recibe valores en sus parámetros mostrará los datos de Arifin Zainal.

Sintaxis

```
IF OBJECT_ID ('dbo.PA_Empleados', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.PA_Empleados;
GO


CREATE PROCEDURE dbo.PA_Empleados
    @LastName  VARCHAR(50)='Arifin'
    , @FirstName VARCHAR(50)='Zainal'
AS
    SELECT
                LastName, FirstName,Department
    FROM
                HumanResources.vEmployeeDepartmentHistory
    WHERE
                FirstName = @FirstName
                AND LastName = @LastName;
GO
```

```
-- Invocación al SP pasando nombre y apellido del empleado
EXEC dbo.PA_Empleados 'Ackerman', 'Pilar';

-- Invocación al SP usando los valores predeterminados
EXEC dbo.PA_Empleados;
```

Crear un procedimiento con parámetros de entrada y salida.
El siguiente ejemplo crea un procedimiento almacenado llamado PA_Empleados que recibe Nombre y Apellido del empleado y guarda en el parámetro de salida el nombre del departamento.

Sintaxis

```
IF OBJECT_ID ('dbo.PA_Empleados', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.PA_Empleados;
GO

CREATE PROCEDURE dbo.PA_Empleados
     @LastName    VARCHAR(50)='Arifin',
     @FirstName   VARCHAR(50)='Zainal',
     @Department    VARCHAR(50) OUTPUT
AS
   SELECT
                  @Department=[Department]
    FROM
                  HumanResources.vEmployeeDepartmentHistory
    WHERE
                  FirstName = @FirstName
                  AND LastName = @LastName;
GO

-- Declaro la variable donde se guardará el departamento del empleado
DECLARE @Departamento VARCHAR(50);

-- Obtengo el departamento de Akerman
EXEC dbo.PA_Empleados 'Ackerman', 'Pilar', @Departamento OUTPUT;

-- Verifica el departamento obtenido
SELECT @Departamento AS [Valor obtenido desde un OUTPUT];
GO
```

Crear un procedimiento con valor de retorno

El siguiente ejemplo crea un procedimiento almacenado llamado PA_Empleados que recibe Nombre y Apellido del empleado y retorna cero si no tiene departamento o uno en caso contrario.

Sintaxis

```
IF OBJECT_ID ('dbo.PA_Empleados', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.PA_Empleados;
GO

CREATE PROCEDURE dbo.PA_Empleados
     @LastName    VARCHAR(50)='Arifin',
     @FirstName   VARCHAR(50)='Zainal'
AS
    DECLARE @Department VARCHAR(50);

    SELECT
                  @Department=[Department]
    FROM
                  HumanResources.vEmployeeDepartmentHistory
    WHERE
                  FirstName = @FirstName
                  AND LastName = @LastName;

    IF @Department IS NULL
          RETURN 0;
    ELSE
          RETURN 1;

GO

DECLARE @Resultado TINYINT;


-- Guardo el valor de retorno en la variable resultado
EXEC @Resultado=dbo.PA_Empleados 'Ackerman', 'Pilar';

-- Verifico si tiene o no departamento
SELECT
      @Resultado AS [Valor obtenido desde un RETURN]
```

Crear un procedimiento combinando parámetros y valor de retorno

El siguiente ejemplo crea un procedimiento almacenado llamado PA_Empleados que recibe Nombre y Apellido del empleado y retorna cero si no tiene departamento almacenando Desconocido en el parámetro de salida o uno en caso contrario y el nombre del departamento.

Sintaxis

```
IF OBJECT_ID ('dbo.PA_Empleados', 'P' ) IS NOT NULL
    DROP PROCEDURE dbo.PA_Empleados;
GO

CREATE PROCEDURE dbo.PA_Empleados
    @LastName   VARCHAR(50)='Arifin'
    ,@FirstName  VARCHAR(50)='Zainal'
    ,@Department VARCHAR(50) OUTPUT
AS

    SELECT
                  @Department=Department
    FROM
                  HumanResources. vEmployeeDepartmentHistory
    WHERE
                  FirstName = @FirstName
                  AND LastName = @LastName;

   -- Si el empleado no tiene departamento
    IF @Department IS NULL
    BEGIN
             SET @Department='Departamento Desconocido';
           RETURN 0;
    END

    RETURN 1;
GO

DECLARE @Resultado TINYINT;
DECLARE @Departamento   VARCHAR(50);

EXEC @Resultado=dbo.PA_Empleados 'Ackerman', 'Pilar', @Departamento
OUTPUT;

-- Verifica los datos del empleado
SELECT
           @Resultado     AS [Valor obtenido desde un RETURN],
           @Departamento  AS [Valor obtenido desde un OUTPUT];
```