STORED PROCEDURES Y TRIGGERS

FACULTAD de INGENIERÍA de
SISTEMAS E INFORMÁTICA

Agenda de hoy

- Vistas.
- Stored Procedure.
- Triggers.

VISTAS

Vista (View)

- Una vista es una tabla virtual que representa los datos de una o más tablas de una forma alternativa
- Una vista es una consulta, que refleja el contenido de una o más tablas, desde la que se puede acceder a los datos como si fuera una tabla
- Las vistas no tienen una copia física de los datos

Beneficios de las Vistas

- **Seguridad, nos puede interesar que los usuarios tengan acceso a una parte de la información que hay en una tabla, pero no a toda la tabla**
- **Comodidad, si tenemos que escribir complejas sentencias SQL, tener una vista nos simplifica esta tarea**

Creación de Vistas

```
CREATE VIEW
   <nombre_vista>
AS
(<sentencia_select>)
```

nombre_vista

Es el nombre de la vista. Los nombres de las vistas deben cumplir las reglas de los identificadores. La especificación del nombre del propietario de la vista es opcional

AS

Especifica las acciones que va a llevar a cabo la vista

sentencia_select

Es la instrucción SELECT que define la vista. Dicha instrucción puede utilizar más de una tabla y otras vistas.

Una vista no tiene por qué ser un simple subconjunto de filas y de columnas de una tabla determinada. Es posible crear una vista que utilice más de una tabla u otras vistas mediante una cláusula SELECT

Ejemplo de una Vista

- *Crear una vista sobre la tabla alquileres, en la que se nos muestre el nombre y apellidos del cliente en lugar de su código*

```
CREATE VIEW vAlquileres
AS
(
SELECT nombre,       apellidos
FROM tAlquileres,     tClientes
WHERE  tAlquileres.codigo_cliente = tClientes.codigo
)
```

PROCEDIMIENTOS ALMACENADOS

Procedimiento Almacenado

- Un PA es un programa que el DBMS almacena en forma persistente y que se ejecuta en el servidor de base de datos
- Un PA ejecuta una acción o conjunto de acciones especificas
- Un PA tiene un nombre, un conjunto de parámetros (opcional) y un bloque de código

Beneficios de los Procedimientos Almacenados

- ✓ Encapsula las reglas y políticas del negocio
- ✓ Permite crear lógica de aplicaciones reusable
- ✓ Facilita el trabajo de los usuarios
- ✓ Brinda mecanismos de seguridad
- ✓ Mejora el rendimiento
- ✓ Reduce el costo derivado de transferencia y la comunicación de datos
- ✓ Reduce la duplicidad de esfuerzo y mejora la modularidad del software

Creación de Procedimiento Almacenado

```
CREATE { PROC | PROCEDURE }
    [schema_name.] procedure_name
    [ ; number ]
    [
    { @parameter
    [ type_schema_name. ] data_type
    }       [ VARYING ] [ = default ] [
OUT | OUTPUT ]     ] [ ,...n ] [ WITH
<procedure_option> [ ,...n ]
    ]
    [ FOR REPLICATION ] AS {
<sql_statement> [;][ ...n ] |
<method_specifier> } [;]
<procedure_option> ::=     [
ENCRYPTION ]     [ RECOMPILE ]     [
EXECUTE_AS_Clause ]
<sql_statement> ::= { [ BEGIN ]
statements [ END ] }
<method_specifier> ::= EXTERNAL
NAME
assembly_name.class_name.metho
d_name
```

schema_name
: Es el nombre del esquema al que pertenece el procedimiento

procedure_name
: Es el nombre del nuevo procedimiento almacenado. Se recomienda no utilizar el prefijo sp_ en el nombre del procedimiento. SQL Server utiliza este prefijo para designar a los procedimientos almacenados del sistema

; number
: Es un entero opcional que se utiliza para agrupar procedimientos que tengan el mismo nombre.

@ *parameter*
: Es un parámetro del procedimiento. En una instrucción CREATE PROCEDURE se pueden declarar uno o más parámetros

[*type_schema_name*.] *data_type*
: Es el tipo de datos del parámetro y el esquema al que pertenece

Creación de Procedimiento Almacenado

```
CREATE { PROC | PROCEDURE }
    [schema_name.] procedure_name
    [ ; number ]
    [
    { @parameter
    [ type_schema_name. ] data_type
    }       [ VARYING ] [ = default ] [
OUT | OUTPUT ]     ] [ ,...n ] [ WITH
<procedure_option> [ ,...n ]
    ]
    [ FOR REPLICATION ] AS {
<sql_statement> [;][ ...n ] |
<method_specifier> } [;]
<procedure_option> ::=     [
ENCRYPTION ]     [ RECOMPILE ]     [
EXECUTE_AS_Clause ]
<sql_statement> ::= { [ BEGIN ]
statements [ END ] }
<method_specifier> ::= EXTERNAL
NAME
assembly_name.class_name.metho
d_name
```

VARYING
: Especifica el conjunto de resultados admitido como un parámetro de salida. Este parámetro es creado de forma dinámica por el procedimiento almacenado y su contenido puede variar. Sólo se aplica a los parámetros de tipo cursor

default
: Es un valor predeterminado para el parámetro. Si se define un valor *default*, el procedimiento se puede ejecutar sin especificar un valor para ese parámetro

OUTPUT
: Indica que se trata de un parámetro de salida

RECOMPILE
: Indica que el plan para este procedimiento se compila en tiempo de ejecución

ENCRYPTION
: Indica que SQL Server convertirá el texto original de la instrucción CREATE PROCEDURE en un formato ofuscado

EXECUTE AS
: Especifica el contexto de seguridad en el que se ejecuta el procedimiento almacenado

Creación de Procedimiento Almacenado

```
CREATE { PROC | PROCEDURE }
    [schema_name.] procedure_name
    [ ; number ]
    [
    { @parameter
    [ type_schema_name. ] data_type
    }       [ VARYING ] [ = default ] [
    OUT | OUTPUT ]     ] [ ,...n ] [ WITH
    <procedure_option> [ ,...n ]
    ]
    [ FOR REPLICATION ] AS {
    <sql_statement> [;][ ...n ] |
    <method_specifier> } [;]
    <procedure_option> ::=     [
    ENCRYPTION ]     [ RECOMPILE ]     [
    EXECUTE_AS_Clause ]
    <sql_statement> ::= { [ BEGIN ]
    statements [ END ] }
    <method_specifier> ::= EXTERNAL
    NAME
    assembly_name.class_name.metho
    d_name
```

FOR REPLICATION

Especifica que los procedimientos almacenados creados para la réplica no se pueden ejecutar en el suscriptor

<sql_statement>

Una o más instrucciones Transact-SQL que se van a incluir en el procedimiento. Para obtener más información sobre algunas limitaciones aplicables

EXTERNAL NAME *assembly_name.class_name.method_name*

Especifica el método de un ensamblado de .NET Framework para que un procedimiento almacenado CLR haga referencia a él

Ejemplo de Procedimiento Almacenado

```
USE AdventureWorks;
GO

CREATE PROCEDURE
   HumanResources.uspGetAllEmployees
AS
BEGIN
SELECT LastName, FirstName, JobTitle, Department
   FROM HumanResources.vEmployeeDepartment;
END
GO
```

Ejecución de Procedimiento Almacenado

```
EXEC sp_helptext
   'HumanResources.uspEncryptThis';
GO

EXECUTE
   HumanResources.uspGetAllEmployees;
GO

EXEC HumanResources.uspGetAllEmployees;
GO
```

Ejemplo de Procedimiento Almacenado

```
 ========================================
-- Autor: <Autor,,Name>
-- Descripción: <Descripción,,>
========================================
===
CREATE PROCEDURE spVendorByState
      @VendorState varchar(50)
AS
BEGIN
      -- SET NOCOUNT ON added to prevent extra result sets
      -  from interfering with SELECT statements.
      SET NOCOUNT ON;

      SELECT VendorId,VendorFName,VendorLName,
            VendorCity,VendorState,VendorCountry,
            PostedDate,VendorDescription
      FROM Vendor Where VendorState = @VendorState
       ORDER BY PostedDate
END
GO
```

Procedimiento Almacenado con Parámetros

```
USE AdventureWorks;
GO
CREATE PROCEDURE HumanResources.uspGetEmployees2
@LastName nvarchar(50),
@FirstName nvarchar(50)`
AS
SET NOCOUNT ON;
SELECT FirstName, LastName, JobTitle, Department
FROM HumanResources.vEmployeeDepartment
WHERE FirstName = @FirstName AND
        LastName = @LastName;
GO
```

Procedimiento Almacenado con OUTPUT

```
USE AdventureWorks;
GO
CREATE PROCEDURE Production.uspGetList
 @Product varchar(40) , @MaxPrice money
, @ComparePrice money     OUTPUT , @ListPrice money OUTPUT
AS SET NOCOUNT ON;
SELECT p.[Name] AS Product, p.ListPrice AS 'List Price`
FROM Production.Product AS p JOIN Production.ProductSubcategory
    AS s ON p.ProductSubcategoryID = s.ProductSubcategoryID
WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice;
-- Populate the output variable @ListPprice.
SET @ListPrice = (SELECT MAX(p.ListPrice) FROM Production.Product
    AS p JOIN Production.ProductSubcategory AS s ON
    p.ProductSubcategoryID = s.ProductSubcategoryID WHERE
    s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice); --
    Populate the output variable @compareprice. SET @ComparePrice
    = @MaxPrice;
GO
```

Procedimiento Almacenado con RECOMPILE

```
USE AdventureWorks;
GO
CREATE PROCEDURE dbo.uspProductByVendor @Name
   varchar(30) = '%`
WITH RECOMPILE
AS
SELECT v.Name AS 'Vendor name', p.Name AS 'Product
   name`
   FROM Purchasing.Vendor AS v JOIN
        Purchasing.ProductVendor AS pv
   ON v.VendorID = pv.VendorID JOIN
        Production.Product   AS p
   ON pv.ProductID = p.ProductID
   WHERE v.Name LIKE @Name;
```

Procedimiento Almacenado con ENCRYPTION

```
USE AdventureWorks;
GO
CREATE PROCEDURE HumanResources.uspEncryptThis
WITH ENCRYPTION
AS SET NOCOUNT ON;
SELECT EmployeeID, Title, NationalIDNumber,
   VacationHours, SickLeaveHours
 FROM HumanResources.Employee;
GO
```

TRIGGERS

Trigger

- **Un trigger (o desencadenador) es una clase especial de procedimiento almacenado que se ejecuta automáticamente cuando se produce un evento en el servidor de bases de datos.**

Tipos de trigger

- Trigger DML, se ejecutan cuando un usuario intenta modificar datos mediante un evento de lenguaje de manipulación de datos (DML). Los eventos DML son instrucciones INSERT, UPDATE o DELETE de una tabla o vista.
- Trigger DDL, se ejecutan en respuesta a una variedad de eventos de lenguaje de definición de datos (DDL). Estos eventos corresponden principalmente a instrucciones CREATE, ALTER y DROP, y a determinados procedimientos almacenados del sistema que ejecutan operaciones de tipo DDL.

Creación de un Triger de DML

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ; ] > }

<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

schema_name

Es el nombre del esquema al que pertenece un desencadenador DML.

trigger_name

Es el nombre del desencadenador. El parámetro trigger_name debe cumplir con las reglas de los identificadores

table | view

Es la tabla o vista en que se ejecuta el desencadenador DML

DATABASE

Aplica el ámbito de un desencadenador DDL a la base de datos actual.

ALL SERVER

Aplica el ámbito de un desencadenador DDL o logon al servidor actual.

Creación de un Triger de DML

```
CREATE TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement [ ; ] [ ,...n ] |
    EXTERNAL NAME <method specifier [ ; ] > }

<dml_trigger_option> ::=
    [ ENCRYPTION ]
    [ EXECUTE AS Clause ]
```

WITH ENCRYPTION

Ofusca el texto de la instrucción CREATE TRIGGER

EXECUTE AS

Especifica el contexto de seguridad en el que se ejecuta el desencadenador.

FOR | AFTER

AFTER especifica que el desencadenador sólo se activa cuando todas las operaciones especificadas en la instrucción SQL desencadenadora se han ejecutado correctamente.

AFTER es el valor predeterminado cuando sólo se especifica la palabra clave FOR.

{ [DELETE] [,] [INSERT] [,] [UPDATE] }

Especifica las instrucciones de modificación de datos que activan el desencadenador DML cuando se intenta en esta tabla o vista.

sql_statement

Son las condiciones y acciones del desencadenador.

Tablas Inserted y Deleted

- Las instrucciones de triggers DML utilizan dos tablas especiales denominadas **inserted** y **deleted**
- SQL Server 2008 crea y administra automáticamente ambas tablas
- La estructura de las tablas inserted y deleted es la misma que tiene la tabla que ha desencadenado la ejecución del trigger
- No puede se modificar directamente los datos de estas tablas

Tablas Inserted y Deleted

- La tabla inserted solo está disponible en las operaciones INSERT y UPDATE y en ella están los valores resultantes despues de la inserción o actualización. Es decir, los datos insertados. Inserted estará vacia en una operación DELETE.
- La tabla deleted, solo esta disponible en las operaciones UPDATE y DELETE, están los valores anteriores a la ejecución de la actualización o borrado. Es decir, los datos que serán borrados. Deleted estará vacia en una operacion INSERT.

Ejemplo de Trigger

```
CREATE TRIGGER TR_CUENTAS
ON CUENTAS
AFTER UPDATE
AS BEGIN
-- SET NOCOUNT ON impide que se generen
   mensajes de texto
-- con cada instrucción
SET NOCOUNT ON;
INSERT INTO HCO_SALDOS
(IDCUENTA, SALDO, FXSALDO)
SELECT IDCUENTA, SALDO, getdate()
FROM INSERTED
END
```

Activación del Trigger

```
UPDATE CUENTAS
SET SALDO = SALDO + 10
WHERE IDCUENTA = 1
```

Ejemplo de Trigger

```
ALTER TRIGGER TR_CUENTAS
ON CUENTAS
AFTER UPDATE
AS
BEGIN
 IF UPDATE(SALDO)
 -- Solo si se actualiza SALDO
    BEGIN INSERT INTO HCO_SALDOS
   (IDCUENTA, SALDO, FXSALDO)
    SELECT IDCUENTA, SALDO, getdate()
    FROM INSERTED
    END
END
```

Activación del Trigger

```
-- Desactiva el trigger TR_CUENTAS
   DISABLE TRIGGER TR_CUENTAS ON CUENTAS
GO
-- Activa el trigger TR_CUENTAS
  ENABLE TRIGGER TR_CUENTAS ON CUENTAS
GO
-- Desactiva todos los triggers de la tabla
   CUENTAS
   ALTER TABLE CUENTAS DISABLE TRIGGER ALL
GO
-- Activa todos los triggers de la tabla CUENTAS
  ALTER TABLE CUENTAS ENABLE TRIGGER ALL
```

EVALUACION

32

Evaluacion

- **¿ Como utilizaría Ud. las vistas, stored procedure y triggers en una empresa ?**