FICHE TECHNIQUE FTC005
Les Triggers et Procédures Stockées

V23.02.05 – 30/09/2022

Les Triggers et Procédures Stockées

Version	Date	Description	Version logiciel
1	31/08/2018	Complément d'informations sur la réplication	V21.00.11
0	13/06/2016	Originale	V19.00.03

WaveSoft

FICHE TECHNIQUE FTC005
Les Triggers et Procédures Stockées

V23.02.05 – 30/09/2022

Sommaire

Les Triggers

Les triggers sont une fonctionnalité intéressante de SQL Server.
Un **trigger** (déclencheur) est un programme qui se lance automatiquement lorsqu'un événement se produit.
Par événement, on entend toute modification (Insert, Delete, update) des données se trouvant dans les tables.

Syntaxe

```
CREATE TRIGGER < Trigger_Name >
  ON  < Table_Name >
  FOR | AFTER | INSTEAD OF < INSERT,DELETE,UPDATE >
  NOT FOR REPLICATION
AS
BEGIN
   -- Insert statements for trigger here

END
```

Explications

CREATE TRIGGER : Indique que l'on crée un trigger.

trigger_name : Nom que l'on souhaite donner au trigger.

ON {table | view} : Précise sur quelle table ou quelle vue s'applique le trigger.

{FOR | AFTER | INSTEAD OF} : Précise le comportement du trigger.

{[INSERT] , [UPDATE] , DELETE]} : Précise le ou les évènements déclencheurs.

NOT FOR REPLICATION : Optionnel mais recommandé. Indique que le trigger ne doit pas être déclenché lorsque la réplication est mise en place pour les CRM nomades et les magasins autonomes.

AS : Introduit le code SQL du trigger

Nous voyons donc qu'un trigger associe un code SQL avec une table (ou une vue), un évènement déclencheur et un comportement.

Les évènements déclencheurs des triggers

Il y a trois évènements susceptibles de déclencher un trigger. Ils correspondent aux trois actions possibles sur un enregistrement dans une table INSERT, DELETE et UPDATE. Un trigger doit spécifier au moins l'un de ces comportements mais peut tout à fait être déclenché par deux ou trois évènements, il suffit pour cela de les séparer par une virgule dans la déclaration :

```
CREATE TRIGGER < Trigger_Name >
  ON  < Table_Name >
  AFTER < INSERT,DELETE,UPDATE >
AS
```

L'évènement INSERT

L'évènement **insert** est déclenché lors de **l'ajout** d'un enregistrement.

L'évènement UPDATE

L'évènement **update** est déclenché lors de la **modification** d'un enregistrement.

L'évènement DELETE

L'évènement **DELETE** est déclenché lors de la **suppression** d'un enregistrement.

Si un trigger est incorrect, il faut le supprimer, car le désactiver ne suffit pas.
En effet, à chaque migration de dossier, il sera alors par erreur réactivé.

Les modes de comportements des triggers

Les triggers possèdent deux comportements différents : soit ils effectuent des opérations **à la suite** de l'action déclenchante, soit ils effectuent des opérations **à la place** de l'action déclenchante.

Le comportement AFTER

Le comportement **AFTER** indique que le trigger est déclenché **après** l'action déclenchante :

Lors d'une action de suppression : d'abord les enregistrements sont supprimés de la table, ensuite les contraintes sont validées, enfin le trigger est déclenché.

Il peut y avoir plusieurs triggers AFTER sur chaque événement.

Le comportement FOR

FOR est considéré comme obsolète et équivalent à **AFTER**.

Le comportement INSTEAD OF

Le comportement **INSTEAD OF** indique que le trigger est déclenché **à la place** de l'action déclenchante.

Deux triggers INSTEAD OF d'une même table ne peuvent se déclencher par le même évènement.

Les pseudo tables

Pour **récupérer les informations** sur l'opération déclenchante, on utilise les pseudos tables : **INSERTED** et **DELETED**.

Les pseudos tables possèdent la même définition que la table sur laquelle le trigger est appliqué. Elles ne permettent que des opérations de sélection (Pas de DELETE, INSERT, UPDATE).

La pseudo table INSERTED

La pseudo table **Inserted** possède la même définition que la table sur laquelle le trigger est appliqué.
Elle représente soit les **nouveaux enregistrements** dans le cas de **l'évènement INSERT** soit les nouvelles valeurs des enregistrements dans le cas de l'évènement **UPDATE**.

La pseudo table DELETED

Elle représente soit les enregistrements **supprimés** dans le cas de l'évènement DELETE, soit les anciennes valeurs des enregistrements dans le cas de l'évènement **UPDATE**.

Exemple

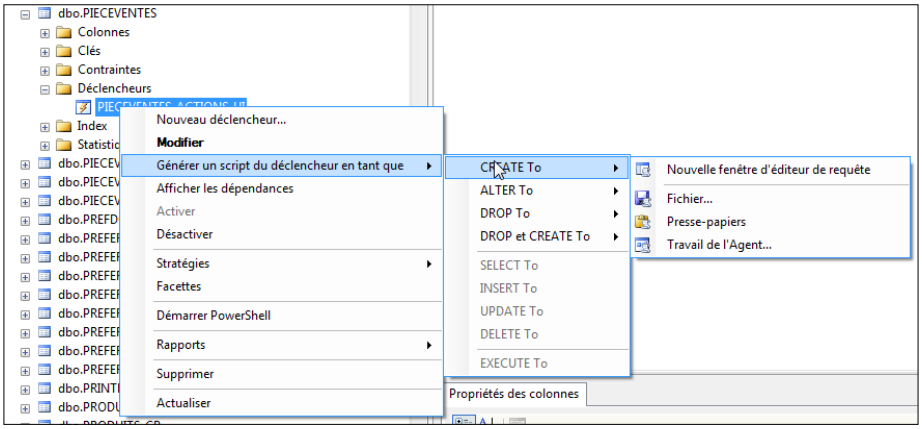
Exemple de mise en œuvre sur la table PIECEVENTES de la Base de données SPORTPLUS.

Créer un trigger permettant lors de la création d'une nouvelle pièce de type Commande, de mettre automatiquement dans le code regroupement " PCVCRITREGROUPE " de la commande le N° cde " PCVNUM" afin de pouvoir livrer N fois une commande mais de regrouper les BL issus d'une même cde automatiquement.

La requête est la suivante :

```
UPDATE PIECEVENTES SET PCVCRITREGROUPE= PIECEVENTES.PCVNUM
FROM INSERTED I
WHERE PIECEVENTES.PCVID = I.PCVID AND PIECEVENTES.PITCODE='C'
```

1. Allons sur SQL server, pour créer le trigger :



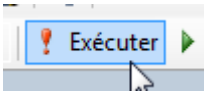
Se positionner sur la table souhaitée, faire un clic droit et sélectionner "Générer un script du déclencheur en tant que …"

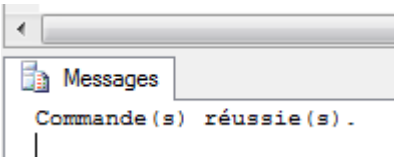
Saisissez votre script et enrergistrez le.

⚠ Les triggers externe à Wavesoft doivent être enregistrés sous le nom : **EXT_xxxxxxx**

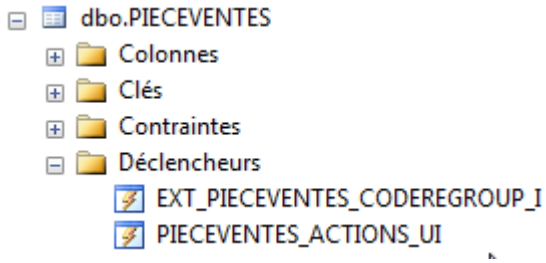
EXT_PIECEVENTES_CODEREGROUP....))

```
/****** Object:  Trigger [EXT_PIECEVENTES_CODEREGROUP_I]    Script Date: 05/26/2011 15:21:34 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
-- =============================================
-- Author:      WaveSoft
-- Create date: 24/05/2011
-- Description: Mise à jour du Code Regroupement
-- =============================================
CREATE TRIGGER [EXT_PIECEVENTES_CODEREGROUP_I] ON [PIECEVENTES] FOR INSERT AS
IF (SELECT COUNT(*) FROM  inserted ) = 1
BEGIN
-- Maj du Code Regroupement
UPDATE PIECEVENTES SET PCVCRITREGROUPE=PIECEVENTES.PCVNUM FROM INSERTED I WHERE PIECEVENTES.PCVID = I.PCVID AND PIECEVENTES.PITCODE='C'
END
```

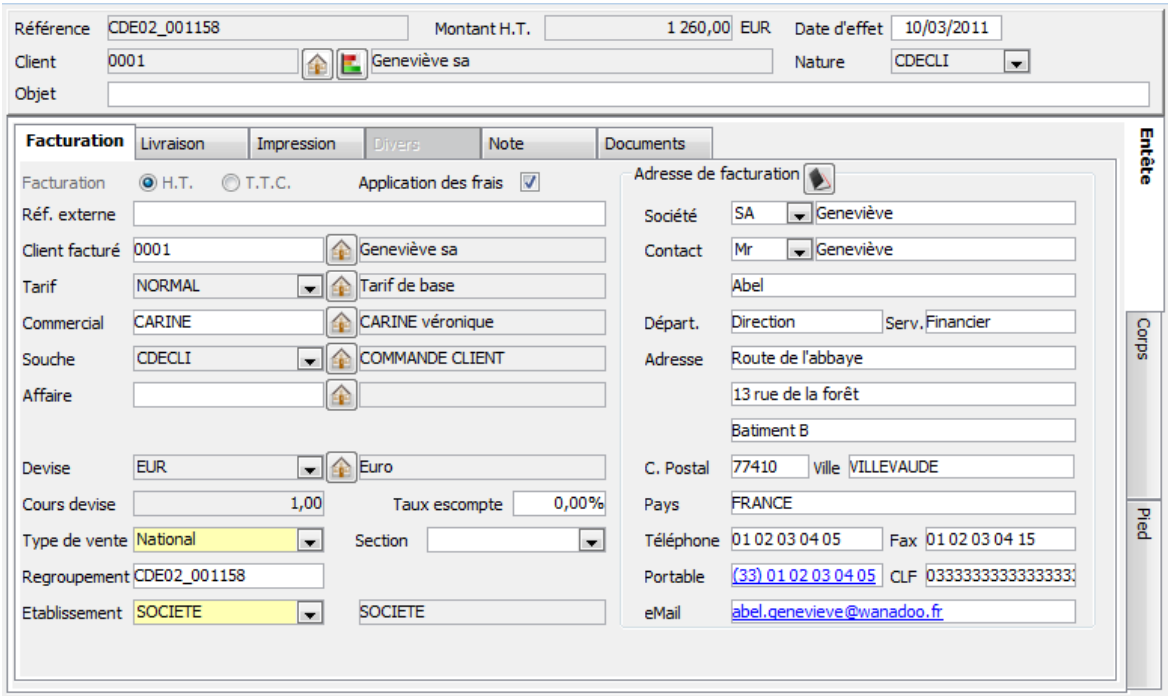
2. Exécutez votre script pour créer le trigger par le bouton 

Vous obtenez le message : 

Déconnectez vous et reconnectez vous au serveur et vérifiez la création du déclencheurs sur la table PIECEVENTES :



3. Maintenant, créez une commande et vérifiez le bon fonctionnement du trigger après l'enregistrement de la pièce de vente.
Le N° de la Commande apparait bien dasn la zone « Regroupement »



Les procédures stockées

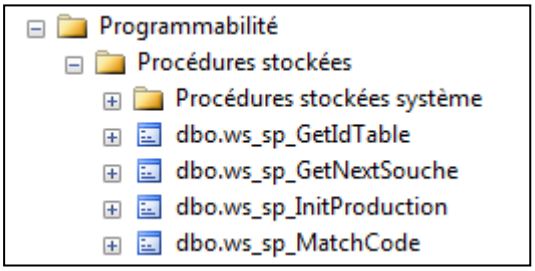
Différence entre un trigger et une procédure stockée

Le trigger est un ensemble d'instructions SQL stocké sur le serveur, dans la base (plus précisément, dans le dictionnaire de données) .Les triggers sont déclenchés sur événements (ex : ajout d'une ligne dans une table).

Les procédures stockées (ou ***stored procedure*** en anglais) sont des ensembles d'instructions SQL pré-compilées, stockées dans la base de données et exécutées sur demande et non pas exécutés automatiquement lors d'événements déclencheurs comme les triggers.

Les procédures stockées peuvent être lancées par un utilisateur ou encore de façon automatique par un événement déclencheur (de l'anglais "trigger").

Des procédures stockées SQL propres à WAVESOFT ont été rajoutées dans la base de données d'exemple SPORTPLUS.



Fonctionnement

Les requêtes envoyées à un serveur SQL font l'objet d'une 'analyse syntaxique' puis d'une interprétation avant d'être exécutées. Ces étapes sont très lourdes si l'on envoie plusieurs requêtes complexes.

Les procédures stockées répondent à ce problème : une requête n'est envoyée qu'une unique fois sur le réseau puis analysée, interprétée et stockée sur le serveur sous forme exécutable (pré-compilée). Pour qu'elle soit exécutée, le client n'a qu'à envoyer une requête comportant le nom de la procédure stockée.

On peut ainsi passer des paramètres à une procédure stockée lors de son appel, et recevoir le résultat de ses opérations comme celui de toute requête SQL.

Les paramètres sont utilisés pour échanger des données entre d'une part une procédure stockée ou une fonction et d'autre part, l'application ou l'outil ayant appelé la procédure stockée ou la fonction :

- Les paramètres d'entrée permettent à l'appelant de faire passer une valeur de donnée à la procédure stockée ou à la fonction.
- Les paramètres de sortie permettent à la procédure stockée de faire passer en retour une valeur de donnée à l'appelant. Les fonctions définies par l'utilisateur ne peuvent pas définir de paramètres de sortie.
- Chaque procédure stockée renvoie un code de retour de type entier à l'appelant. Si la procédure stockée ne définit pas explicitement la valeur du code de retour, la valeur de ce code est 0.

La procédure stockée suivante utilise un paramètre d'entrée, un paramètre de sortie et un code de retour :

Intérêts

- **Simplification** : Code plus simple à comprendre
- **Rapidité** : Moins d'informations sont échangées entre le serveur et le client
- **Performance** : Economise au serveur l'interprétation de la requête car elle est précompilée
- **Sécurité** : Les applications et les utilisateurs n'ont aucun accès direct aux tables, mais passent par des procédures stockées prédéfinies

Syntaxe

```
CREATE PROCEDURE < Procedure_Name >
   -- Add the parameters for the stored procedure here
AS
BEGIN
    -- Insert statements for procedure here
    SELECT
END
```

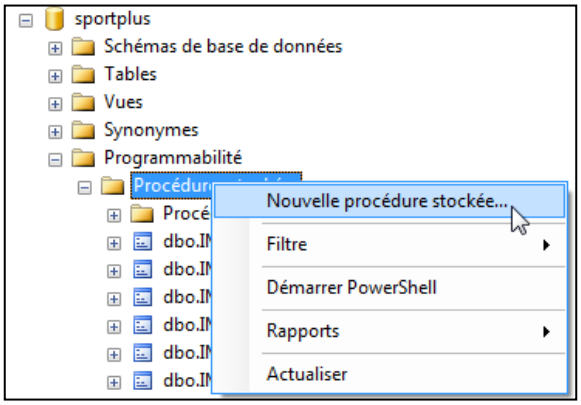
Explications

CREATE PROCEDURE	: Indique que l'on crée une procédure.
procedure_name	: Nom que l'on souhaite donné à la procédure stockée.
AS	: Introduit le code SQL du trigger

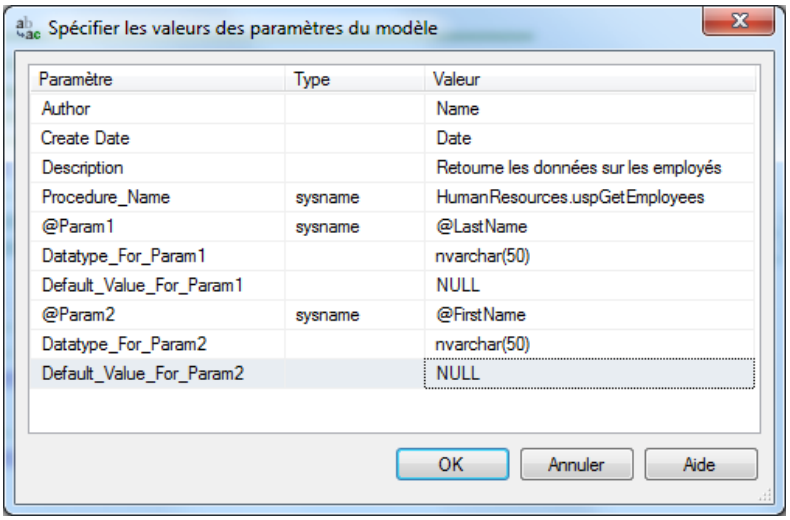
Exemple

Cette partie explique comment créer une procédure stockée Transact-SQL à l'aide de l'Explorateur d'objets dans SQL Server Management Studio

Exemple de mise en œuvre sur la Base de données SPORTPLUS.



1. Dans l'**Explorateur d'objets**, connectez-vous à une instance du moteur de base de données et développez-la.
2. Développez **Bases de données**, développez la base de données à laquelle appartient la procédure stockée, puis développez **Programmabilité**.
3. Cliquez avec le bouton droit sur **Procédures stockées**, puis cliquez sur **Nouvelle procédure stockée**.
4. Dans le menu **Requête**, cliquez sur **Spécifier les valeurs des paramètres de modèle**.



5. Dans la boîte de dialogue **Spécifier des valeurs pour les paramètres de modèle**, la colonne **Valeur** contient des valeurs possibles pour les paramètres. Acceptez les valeurs ou remplacez-les par de nouvelles valeurs, puis cliquez sur **OK**.
6. Dans l'éditeur de requêtes, remplacez l'instruction SELECT par les instructions appropriées pour votre procédure.
7. Pour tester la syntaxe, dans le menu **Requête**, cliquez sur **Analyser**.
8. Pour créer la procédure stockée, dans le menu **Requête**, cliquez sur **Exécuter**.
9. Pour enregistrer le script, dans le menu **Fichier**, cliquez sur **Enregistrer**. Acceptez le nom de fichier ou remplacez-le par un autre nom, puis cliquez sur **Enregistrer**.
10. Pour exécuter la procédure stockée, dans la barre d'outils, cliquez sur **Nouvelle requête**.
11. Dans la fenêtre de requête, entrez les instructions suivantes :

```
USE AdventureWorks2008R2;
GO
EXECUTE HumanResources.uspGetEmployees @FirstName = N'Diane', @LastName = N'Margheim';
GO
```

12. Dans le menu **Requête**, cliquez sur **Exécuter**.

Saisissez votre script et enrergistrez le.

⚠ Les procédures stockées externe à Wavesoft doivent être enregistrés sous le nom : **EXT_sp_xxxxxxx**

Pour exécuter une procédure stockée vous devez lancer l'instruction suivante : **EXECUTE**.

Procédures Stockées WAVESOFT

ws_sp_GetIdTable

Cette procédure permet de générer un nouvel identifiant pour une table données.

Le nom de la table doit être passé en paramètre d'entrée de la procédure stockée.
La procédure renvoie l'identifiant de la table.

```
EXEC @ATLID = ws_sp_GetIdTable @Table
```

Exemple

Nous souhaitons obtenir le prochain AFFID disponible dans la table AFFAIRES.

```
-- Définitions des valeurs des paramètres
DECLARE @ATLID IDD
DECLARE @Table varchar(40)

BEGIN
 SET @table = 'AFFAIRES'
 /****** Execution de la procedure stockée [ws_sp_GetIdTable] qui met à jour
 l'identifiant de la table AFFAIRES ******/
 EXEC @ATLID = ws_sp_GetIdTable @Table
END
```

ws_sp_InitProduction

Cette procédure ne doit pas être utilisée.

Elle est appelée en initialisation du Module de Production.

ws_sp_MatchCode

Cette procédure ne doit pas être utilisée.

Cette procédure permet de déterminer un MatchCode du fichier "Tiers" (détection de doublons dans les tiers divers).

ws_sp_GetNextSouche

Cette procédure permet de déterminer la prochaine souche.

L'Id de la souche doit être passé en paramètre d'entrée de la procédure stockée.
La procédure renvoie la prochaine souche.

```
EXEC ws_sp_GetNextSouche @NUMId, @SOUCHE OUTPUT
```

Exemple

A la création d'une commande client, création automatique d'une affaire.

```
CREATE TRIGGER [dbo].[EXT_PIECEVENTES_AUTOAFFAIRE_I] ON [dbo].[PIECEVENTES] FOR INSERT AS
DECLARE @ATLID IDD
DECLARE @Souche varchar(50)
DECLARE @type varchar(10)
DECLARE @Table varchar(40)
DECLARE @NUMId INT
IF (SELECT COUNT(*) FROM INSERTED I) = 1
BEGIN
	SET @table = 'AFFAIRES'
	/****** Execution de la procedure stockée [ws_sp_GetIdTable] qui met à jour l'identifiant de la table
AFFAIRES ******/
	EXEC @ATLID = ws_sp_GetIdTable @Table

	SET @NUMID = 40  -- NUMCODE = AFFAIRE
	/****** Execution de la procedure stockée [ws_sp_GetNextSouche] qui met à jour le code de la
souche de AFFAIRES ******/
	EXEC ws_sp_GetNextSouche @NUMId, @SOUCHE OUTPUT

	select @type = I.PINCODE FROM INSERTED I    -- Récupération du type de la commande

	IF @type = 'CDECLI'                                                    -- Si le type de la commande
est COMMANDECLIENT
	BEGIN
		INSERT INTO AFFAIRES(AFFID, AFFCODE,
AFFINTITULE,MEMOID,TIRID,AFFISACTIF,AFFCRITREGROUPE)
		VALUES(@ATLID, @Souche,(SELECT I.PCVNUMEXT FROM INSERTED I),NULL,NULL,'O',NULL)
		SELECT @ATLID FROM INSERTED I

		/****** Mise à jour de la Table paramètrable ******/
		INSERT INTO AFFAIRES_P(AFFID) SELECT @ATLID
	END
END
```

Triggers et messages

L'objectif ici est de permettre d'afficher des alertes issues de trigger sous une forme plus conviviale que les messages d'erreur standard de l'ERP WaveSoft.

Sql Server possède une gestion de messages, dans cette gestion, une plage de messages est réservée aux utilisateurs. (Les messages dont l'identifiant est supérieur à 50000). De plus cette gestion de message permet de gérer le multilingue.

Mise en œuvre :

Ajouter le message dans le server SQL à l'aide de la procédure stockée : **sp_addmessage**, vous devez exécuter cette procédure pour chaque langue utilisée. Les messages ainsi définis permettent l'utilisation de paramètre dans le message.

Remarque : vous devez obligatoirement renseigner la langue anglaise en premier.

Exemple :

```
EXEC sp_addmessage @msgnum = 50001, @severity = 16,
   @msgtext = 'The item named %s already exists in %s.',
   @lang = 'us_english';

EXEC sp_addmessage @msgnum = 50001, @severity = 16,
   @msgtext = 'L''élément nommé %s! existe déjà dans %s!',
   @lang = 'French';
```

L'utilisation de ce message dans le trigger à l'aide par exemple de l'instruction **RAISERROR** nous donne :

```
RAISERROR(50001,1,1,15,'param1','param2')
```

L'ERP WaveSoft interceptera ce message comme étant un message utilisateur et donc l'affichera dans un MessageBox standard dans la langue de l'utilisateur connecté.

*Note : Pour supprimer un message utiliser la procédure : **sp_dropmessage***

Exemple :

```
EXEC sp_dropmessage @msgnum = 50001, @lang = 'French';
EXEC sp_dropmessage @msgnum = 50001, @lang = 'us_english';
```