• Un trigger (ou déclencheur) est une procédure stockée spéciale qui s'exécute automatiquement lorsqu'un événement

se produit dans le serveur.

• On distingue trois types de triggers :

♦ Les triggers DML : s'exécutent lorsqu'un utilisateur essaie de mmettre à jour (INSERT, UPDATE ou DELETE) des

données via un événement DML (Data Manipulation Language) sur une table ou vue.

♦ Les triggers DDL : Les déclencheurs DDL s'exécutent en réponse à différents événements du langage de définition

de données (DDL). Ces événements correspondent essentiellement aux instructions Transact-SQL CREATE, ALTER et DROP ...

♦ Les triggers de connexion : Les déclencheurs de connexion sont activés lorsqu'une session utilisateur est établie.

--> Seuls les triggers DML seront traités dans ce chapitre.

\*/

-- Création des triggers DML :

create trigger TriggerName | create trigger TriggerName

on TableName | on TableName

after [insert][,][update][,][delete] | instead of [insert][,][update][,][delete]

as begin | as begin

... | ...

end | end

/\*

• Un trigger DML est crée au niveau d'une table

--> La suppression de la table provoque la suppression de ses déclencheurs

• On distingue deux types de triggers DML :

♦ AFTER : s'exécute après exécution de la requête INERT, UPDATE, DELETE qui l'a declenchée.

♦ INSTEAD OF : se declénche à la place de la requête.

--> Pas de triggers BEFORE sous SQL Server mais on peut les implémenter autrement.

\*/

-- Modification :

alter trigger ...

-- Suppression :

drop trigger ...

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

-- Exemple 1 : AFTER INSERT

if (object\_id('t\_1', 'TR') is not null)

drop trigger t\_1

go

create trigger t\_1

on Stagiaires

after insert

as begin

print 't\_1 : L''insertion a été effectuée avec succès.'

end

go

-- Test :

insert into Stagiaires(Numéro, Nom) values ('S004', 'Soulaimani Assia')

go

select \* from Stagiaires

go

/\*

Un trigger AFTER est exécuté seulement lorsque toutes les opérations spécifiées dans l'instruction SQL

de déclenchement (INSERT dans cet exemple) ont été exécutées correctement.

\*/

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

-- Exemple 2 : AFTER UPDATE

if (object\_id('t\_2', 'TR') is not null)

drop trigger t\_2

go

create trigger t\_2

on Notes

after update

as begin

print 't\_2 : La modification a été effectuée avec succès.'

end

go

-- Test :

update Notes

set Note = Note + 1

where (Stagiaire = 'S001' and Module = 'M002')

go

select \* from Notes

go

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

-- Exemple 3 : AFTER DELETE

if (object\_id('t\_3', 'TR') is not null)

drop trigger t\_3

go

create trigger t\_3

on Stagiaires

after delete

as begin

print 't\_3 : La suppression a été effectuée avec succès.'

end

go

-- Test :

delete Stagiaires

where (Numéro = 'S004')

go

select \* from Stagiaires

go

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

/\*

Utilisation des tables INSERTED et DELETED : http://msdn.microsoft.com/fr-fr/library/ms191300%28v=SQL.90%29.aspx

• Les instructions de déclenchement DML utilisent deux tables spéciales : la table DELETED et la table INSERTED.

Ces pseudo-tables ont la même structure que la table du déclencheur.

• La table DELETED stocke des copies des lignes affectées par les instructions DELETE et UPDATE.

--> Pendant l'exécution d'une instruction DELETE ou UPDATE, certaines lignes sont supprimées de la table du

déclencheur et déplacées vers la table DELETED.

• La table INSERTED stocke des copies des lignes affectées par les instructions INSERT et UPDATE.

--> Pendant une transaction d'insertion ou de mise à jour, de nouvelles lignes sont ajoutées dans la table

INSERTED et dans la table du déclencheur. Les lignes de la table INSERTED sont des copies des lignes créées

dans la table du déclencheur.

• D'un point de vue théorique, une transaction de mise à jour est une opération de suppression suivie d'une

opération d'insertion ; les anciennes lignes sont d'abord copiées dans la table DELETED, et les nouvelles lignes

sont ensuite copiées dans la table du déclencheur et dans la table inserted.

\*/

-- Exemple 4 : Pseudo-tables INSERTED et DELETED - 1 -

if (object\_id('t\_4', 'TR') is not null)

drop trigger t\_4

go

create trigger t\_4

on Modules

after insert, update, delete

as begin

select \* from inserted

select \* from deleted

end

go

-- Tests :

insert into Modules values('M004', 'UML', 5)

update Modules set Coéfficient = 6 where Code = 'M004'

delete Modules where Code = 'M004'

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

-- Exemple 5 : INSTEAD OF - Au lieu de ... -

if (object\_id('t\_5', 'TR') is not null)

drop trigger t\_5

go

create trigger t\_5

on Modules

instead of insert

as begin

print 'Aucun enregistrement n''a été inséré dans la table ???'

select \* from Modules

end

go

-- Test :

insert into Modules values ('M005', 'Introduction aux réseaux', 4)

/\*

Remarques très importantes :

• Il n'est pas possible de créer deux triggers du type INSTEAD OF pour la même opération sur la

même table.

• Il est possible de créer plusieurs triggers de type AFTER sur la même opération sur une table

donnée, l'ordre de déclenchement de ces triggers est aléatoire ...

\*/

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

-- Exemple 6 : Implémentation de la contrainte PRIMARY KEY

if (object\_id('t\_6', 'TR') is not null)

drop trigger t\_6

go

create trigger t\_6

on Stagiaires

instead of insert

as begin

set nocount on

declare @numéro varchar(4),

@nom varchar(50)

select @numéro = Numéro, @nom = Nom

from inserted

if exists (select \* from Stagiaires where Numéro = @numéro)

print 'Violation de la contrainte clé primaire : Opération annulée !!!'

else

begin

insert into Stagiaires -- Attention : récursivité

select \* from inserted

print 'L''insertion a été effectuée avec succès'

end

end

go

-- Test :

insert into Stagiaires (Numéro, Nom) values ('S003', 'Er-Roudani Ahmed')

go

insert into Stagiaires (Numéro, Nom) values ('S005', 'Doukali Hatem')

go

select \* from Stagiaires

-- Problème : Le trigger précédent ne supporte pas les insertions avancées.

--> Solution : Utilisation des curseurs

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

-- Exemple 7 : Changements des salaires :

if exists (select \*

from sys.tables

where name = 'Changements')

drop table Changements

go

create table Changements(

Matricule varchar(4),

AncienSalaire money,

NouveauSalaire money,

DateChangement datetime default getdate())

go

/\*

Le trigger suivant utilise le fonction UPDATE(...)

• Cette fonction permet de tester si une colonne est visée par un changement de valeur.

• Elle ne peut être utilisée que dans les triggers de type INSERT et UPDATE.

• La fonction COLUMNS\_UPDATED() est similaire à UPDATE(...) mais elle est beaucoup plus puissante.

\*/

if (object\_id('t\_7', 'TR') is not null)

drop trigger t\_7

go

create trigger t\_7

on Formateurs

after update

as begin

declare @matricule varchar(4),

@ancienSalaire money,

@nouveauSalaire money

if update(Salaire)

begin

select @matricule = Matricule, @ancienSalaire = Salaire

from deleted

select @NouveauSalaire = Salaire

from inserted

if (@AncienSalaire != @NouveauSalaire)

begin

insert into Changements values (@Matricule, @AncienSalaire, @NouveauSalaire, default)

select \* from Changements

end

end

end

go

-- Test:

update Formateurs

set Salaire = 1.1 \* Salaire -- Ajout de 10% du salaire au salaire du formateur F001

where Matricule = 'F001'

go

update Formateurs

set Salaire = Salaire

where Matricule = 'F001'

go

-- Problème : Le trigger précédent ne supporte pas les insertions avancées.

--> Solution : Utilisation des curseurs

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

-- Exemple 8 : Utilisation de ROLLBACK TRANSACTION (TRIGGER BEFORE)

if (object\_id('t\_8', 'TR') is not null)

drop trigger t\_8

go

create trigger t\_8

on Modules

after update

as begin

if update(Libellé)

begin

declare @Libellé varchar(50)

set @Libellé = (select Libellé from inserted)

if (select count(\*)

from Modules

where [Libellé] = @Libellé) >= 2

begin

print 'Opération annulée !!!'

rollback

end

end

end

go

-- Tests :

update Modules set Libellé = 'SGBD1' where Code = 'M002'

select \* from Modules

go

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

---\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*-.\_\_.-\*\*---

-- Autres applications :

-- Implémentation des contraintes

-- Suppression en cascade

-- Maintenance des champs calculés

-- sys.triggers :

select \* from sys.triggers

-- sys.trigger\_events

select \* from sys.trigger\_events