Chapitre 3.2.3 : Programmer des déclencheurs.

Événements qui déclenchent les triggers.
- Formalisme à respecter.
- Programmation des Triggers.
- Test du trigger et correction des erreurs.

Les triggers ou déclencheurs servent à étendre les mécanismes de gestion de l'intégrité référentielle (liens d'héritage par exemple) et permettre le contrôle de saisie. Il s'agit de code déclenché lors de certains événements de la base de données. Un trigger est toujours rattaché à une table. Les événements qui déclenchent un trigger sont :

- l'insertion de données (INSERT)
- la suppression de données (DELETE)
- la mise à jour (UPDATE)

Ils sont donc déclenchés systématiquement par une requête SQL, après l'exécution de cette requête (AFTER), ou à la place de l'exécution de cette requête (INSTEAD). SQL Server n'implémente pas de trigger BEFORE (avant exécution), mais nous verrons comment le simuler...

Exemples

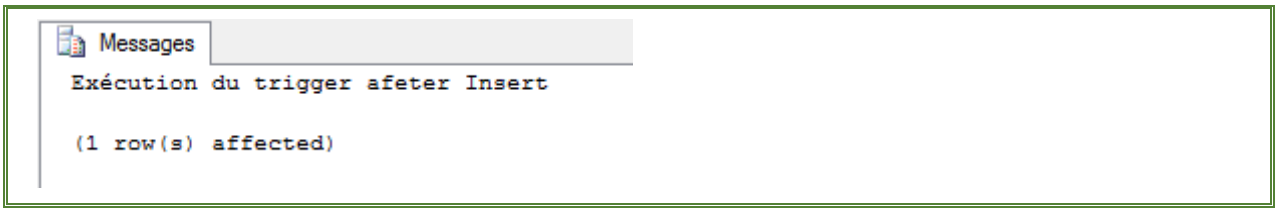
TP 24 : Bonjour Trigger

Question 1 :

```
Create trigger insert_stagiaire on Stagiaires
after insert
as
begin
 print 'Exécution du trigger afeter Insert'
end
```

Question 2 : Tester l'exécution du trigger.

```
insert into Stagiaires(nom,prenom) values('madni','kamal');
```



Syntaxe

```
CREATE TRIGGER nom_trigger ON { table | vue }
 [ WITH ENCRYPTION ]
 {FOR | AFTER | INSTEAD OF } {  INSERT , UPDATE ,DELETE }
  [ WITH APPEND ]    [ NOT FOR REPLICATION ]
  AS
  [ IF UPDATE ( colonne )
  | IF ( COLUMNS_UPDATED ( )opérateur_comparaison_bits)]
Instructions_SQL
```

FOR

Permet de préciser à quel ordre SQL DML le déclencheur est associé. Par défaut, le déclencheur est de type AFTER.

AFTER

C'est le mode par défaut des déclencheurs. Le code est exécuté après vérification des contraintes d'intégrité et après modification des données.

INSTEAD OF

Le corps du déclencheur est exécuté à la place de l'ordre SQL envoyé sur la table ou la vue. Ce type de déclencheur est particulièrement bien adapté pour les vues.

INSERT, UPDATE, DELETE

Un déclencheur peut agir par rapport à une ou plusieurs actions. Dans ce cas, on séparera les actions par des virgules.

IF UPDATE (*colonne*)

Ne peut être utilisé que pour les déclencheurs UPDATE ou INSERT et ne s'exécutera que si la ou les colonnes sont concernées.

Instructions_SQL

Il est possible d'utiliser toutes les instructions Transact SQL de manipulations de données (DML). Les instructions suivantes ne sont pas autorisées :

- CREATE et DROP.
- ALTER
- TABLE et ALTER DATABASE.
- TRUNCATE.
- GRANT et REVOKE.
- UPDATE
- STATISTICS.
- RECONFIGURE.
- LOAD
- DATABASE.

Il possible de définir, sur une même table, plusieurs déclencheurs pour chaque opération INSERT, UPDATE et DELETE

Les déclencheurs sont exécutés après (AFTER) vérification des contraintes d'intégrité et insertion des données dans la table.

Si l'instruction SQL échoue, alors le déclencheur n'est pas exécuté.

IF (COLUMNS_UPDATED ( ) *opérateur_comparaison_bits*)

Cette fonction permet de connaître les indices de la ou des colonnes qui ont été mises à jour. Pour chaque colonne affectée par la mise à jour, un bit est levé. Pour savoir quelles ont été les colonnes mises à jour, une simple comparaison binaire suffit.

WITH ENCRYPTION

La définition du déclencheur est enregistrée de façon cryptée. Il n'est donc pas possible de connaître le code du déclencheur a posteriori. Cette option évite également que le déclencheur soit publié dans le cadre d'une réplication.

WITH APPEND

Cette clause n'est nécessaire que si le niveau de compatibilité de la base est inférieur ou égal à 65. Elle permet alors d'ajouter plusieurs déclencheurs sur un même ordre SQL et un même objet. Ce comportement est celui par défaut depuis la version (70).

NOT FOR REPLICATION

Indique que le déclencheur ne doit pas être exécuté lorsque la modification des données est issue d'un processus de réplication.

Table de travail : Inserted et Deleted

Lors des modifications de données, SQL Server crée des lignes dans des tables de travail ayant la même structure que la table modifiée : les tables **inserted** et **deleted**.

- Lors d'une commande INSERT, la table **inserted** contient une copie logique des lignes créées.
- Lors d'une commande DELETE, les lignes supprimées sont placées dans la table **deleted**.
- Lors d'une commande UPDATE, les lignes contenant les modifications sont placées dans la table **inserted**, les lignes à modifier dans la table **deleted**.

Les tables **inserted** et **deleted** peuvent être accessibles pendant l'exécution du trigger.

TP 25 : Trigger sur Insert, Inserted

Question 1 : Créer un trigger « BonjourInserted » qui affiche la table « Inserted »

```
Create Trigger BonjourInserted on Stagiaires
after Insert
as
Begin
select * from Inserted
End
```

Question 2 : Ecrire un script qui permet de tester le trigger « BonjourInserted »

```
insert into Stagiaires(nom,prenom) values('madani','Ali');
```

Results | Messages

	id	nom	prenom	cin	e_mail	date_naissance	tel_fixe	tel_portable
1	9	madani	Ali	NULL	NULL	NULL	NULL	NULL

TP 26: Trigger Sur Delete , Deleted

Question 1 : écrire le trigger « BonjourDeleted » qui affiche la table « Deleted » après la suppression d'un stagiaire

```
Create Trigger BonjourDeleted on Stagiaires
after Delete
as
Begin
select * from deleted
End
```

Question 2 : écrire un script qui permet de tester le trigger « BonjourDeleted »

```
delete from Stagiaires where id = 9
```

Results | Messages

	id	nom	prenom	cin	e_mail	date_naissance	tel_fixe	tel_portable
1	9	ali	Ali	NULL	NULL	NULL	NULL	NULL

TP 27: Trigger sur Update, Deleted, Inserted

Question 1 : écrire le trigger « BonjourDeletedInserted » qui affiche les tables « Deleted » et « Inserted » après l'exécution de la requête « Update » sur la table stagiaire.

```
Create Trigger BonjourDeletedInserted on Stagiaires
after Update
as
Begin
select * from Deleted
select * from Inserted
End
```

Question 2 : écrire un script qui permet de tester le trigger « BonjourDeletedInserted »

```
update Stagiaires set nom = 'madani' , prenom = 'Kamal' where id = 8
```

Results | Messages

	id	nom	prenom	cin	e_mail	date_naissance	tel_fixe	tel_portable
1	8	madani	ali	NULL	NULL	NULL	NULL	NULL

	id	nom	prenom	cin	e_mail	date_naissance	tel_fixe	tel_portable
1	8	madani	Kamal	NULL	NULL	NULL	NULL	NULL

TP 28 : For Insert, Delete, Update

Ecrire un trigger qui affiche la tâche après « Insert, Delete et Update »

TP 29 : Trigger qui corrige le nombre de jours de réalisation

Question 1 : Ecrire un trigger qui corrige le nombre de jour de la tâche après l'affectation d'une tâche à un stagiaire.

Question 2 : Ecrire un trigger qui corrige le nombre de jour de réalisation d'un projet après l'insertion ou la modification du nombre de jour d'une tâche.

Instead OF

TP 30: Bonjour Instead OF

Question 1 : Ecrire le trigger « BonjourInsteadOf » qui empêche l'insertion d'un projet avec le nombre de jours de réalisation supérieur à zéro.

```
Create Trigger BonjourInsteadOf on Projets
Instead of Insert
as
Begin
 declare @titre varchar(255)
 declare @nombre_jours int
 select @titre= titre , @nombre_jours = nombre_jours_realisation from Inserted
 if @nombre_jours > 0
    begin

    RaisError('Le nombre de jours de réalisation du projet doit être égale Zéro', 16,1)

    end
 else
    begin
     insert into projet values(@titre,@nombre_jours)
    end
End
```

Question 2 :

Ecrire un script pour tester le trigger « BonjourInsteadOf »

```
Insert into Projets values( 'Projet 1',1)
```

```
alter Trigger BonjourInsteadOf on Projets
Instead of Insert
as
Begin
 declare @titre varchar(255)
 declare @nombre_jours int
 select @titre= titre , @nombre_jours = nombre_jours_realisation from Inserted
 if @nombre_jours > 0
    begin

    RaisError('Le nombre de jours de réalisation du projet doit être égale Zéro', 16,1)

    end
 else
    begin
     insert into projets values(@titre,@nombre_jours)
    end
End
```

TP31 : Validation email par trigger

Question 1 : écrire un script qui vérifier si une chaine de caractère est un émail correcte.

LIKE '%_@_%_.__%'

Question 2 : écrire un script qui vérifier si une chaine de caractère est un numéro de téléphone correcte.

Question 3 : écrire un trigger qui empêche l'insertion d'un stagiaire avec un email et téléphone incorrecte.

Exemple de Code

Exemple 1 : Utilisation de rollback et insert avec select

```
CREATE TRIGGER TR_IOF_I_personne
    ON dbo.personne
    INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS
    (
          SELECT *
          FROM INSERTED
          WHERE prenom = 'test'
    )
    BEGIN
          RAISERROR('Le prénom ne peut pas être test !!!',
16, 1)
          ROLLBACK -- afin de stopper l'enregistrement
    END
    ELSE
    BEGIN
          INSERT INTO dbo.personne
          (
               mesColonnes
          )
          SELECT mesColonnes
          FROM INSERTED
    END
END
```