

Programmation de bases de données: Les triggers ou déclencheurs

Plan de la séance:

- Retour sur la dernière séance
 - Point de vue des enseignants
 - Point de vue des étudiants
- Les triggers ou déclencheurs
 - Définition et rôle
 - Syntaxe
 - Les tables INSERTED et DELETED
 - Exemples
 - Points clés

Objectifs de cette séance:

- Retour sur la dernière séance
 - Point de vue des enseignants
 - Point de vue des étudiants
- Rappel, procédures stockées
- Les triggers ou déclencheurs
 - Définition et rôle
 - Syntaxe
 - Les tables INSERTED et DELETED
 - Exemples

Définition et Rôle

- Définition

Les triggers sont des procédures stockées qui s'exécutent automatiquement quand un événement se produit. En général cet événement représente une opération DML (Data Manipulation Language) sur une table

- Rôle des triggers

- Contrôler les accès à la base de données
- Assurer l'intégrité des données
- Garantir l'intégrité référentielle (DELETE, ou UPDATE CASCADE)
- Tenir un journal des logs.

Même si les triggers jouent un rôle important pour une base de données, il n'est pas conseillé d'en créer trop. Certains triggers peuvent rentrer en conflit, ce qui rend l'utilisation des tables impossible pour les mises à jour.

π

Syntaxe

```
CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name  
ON { table | view }  
{ FOR | AFTER | INSTEAD OF }  
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }  
AS { sql_statement }
```

Syntaxe

- AFTER spécifie que le déclencheur DML est déclenché uniquement lorsque toutes les opérations spécifiées dans l'instruction SQL ont été exécutées avec succès.
- Un trigger utilisant AFTER va effectuer l'opération DML même si celle-ci n'est pas valide, un message erreur est quand même envoyé. Utiliser ROLLBACK pour annuler une opération invalide
- FOR fait la même chose que AFTER. Par défaut on utilise AFTER.
- INSTEAD OF indique un ensemble d'instructions SQL à exécuter à la place des instructions SQL qui déclenche le trigger.
- Au maximum, un déclencheur INSTEAD OF par instruction INSERT, UPDATE ou DELETE peut être définie sur une table ou une vue. Définir des vues pour des vues pour des INSTEAD OF.

Fonctionnement: Les tables INSERTED et DELETED

- Lors de l'ajout d'un enregistrement pour un Trigger INSERT, le SGBD prévoit de récupérer l'information qui a été manipulée par l'utilisateur et qui a déclenché le trigger. Cette information (INSERT) est stockée dans une table temporaire appelée **INSERTED**.
- Lors de la suppression d'un enregistrement, DELETE, le SGBD fait la même chose en stockant l'information qui a déclenché le trigger dans une table temporaire appelée **DELETED**.
- Lors, d'une mise à jour, UPDATE l'ancienne valeur est stockée dans la table DELETED et la nouvelle valeur dans INSERTED.

Exemple 1. On contrôle le nombre de bonnes réponses

```
create or alter trigger CTRLnbBonnerep on ReponseTrivia after insert, update as
begin
declare @idQuestion int,
@totalBonneRep smallint;
select @idQuestion = idQuestion from inserted;

select @totalBonneRep = count(idQuestion) from ReponseTrivia
where idQuestion =@idQuestion and estBonne = 'o';

    if (@totalBonneRep > 1)
        begin
            rollback;
            RAISERROR (15600,-1,-1, 'Limite de une bonne reponse par question');
        end;
end;
```

Exécution

```
begin transaction
```

```
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('kba','o',116);
```

```
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('kbo','o',116);
```

```
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('kbi','n',116);
```

```
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('kby','n',116);
```

```
commit;
```

```
(1 ligne affectée)
```

```
Msg 15600, Niveau 15, État 1, Procédure CTRLnbBonnerep, Ligne 16 [Ligne de départ du lot 6]
```

```
Paramètre ou option non valide pour la procédure 'Limite d'une bonne reponse par question'.
```

```
Msg 3609, Niveau 16, État 1, Ligne 9
```

```
La transaction s'est terminée dans le déclencheur. Le traitement a été abandonné.
```

π

Exemple2, Trigger qui contrôle le nombre total de réponses par question.

```
create trigger CTRLnbTotalReponses on ReponseTrivia after insert, update as
begin
declare @idQuestion int,
@nbReponse smallint;

select @idQuestion = idQuestion from inserted;
select @nbReponse = count(*) from ReponseTrivia
where idQuestion =@idQuestion

        if (@nbReponse > 4)
        begin
        rollback;
        RAISERROR (15600,-1,-1, 'Limite de 4 reponses par question.');
```

Exécution

```
begin transaction
```

```
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('kba','o',116);
```

```
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('kbo','n',116);
```

```
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('kbi','n',116);
```

```
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('kby','n',116);
```

```
insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('kbu','n',116);
```

```
commit;
```

```
(1 ligne affectee)
```

```
Msg 15600, Niveau 15, État 1, Procédure CTRLnbTotalReponses, Ligne 13 [Ligne de départ du lot 6]
```

```
Paramètre ou option non valide pour la procédure 'Limite de 4 reponses par question.'.
```

```
Msg 3609, Niveau 16, État 1, Ligne 12
```

```
La transaction s'est terminée dans le déclencheur. Le traitement a été abandonné.
```

π

Exemple 3, on supprime dans la table categorieTrivia et on met à null le idCategorie dans la table questionTrivia (pour les categories supprimées)

```
create trigger updateQuestion on CategorieTrivia instead of delete as
begin
    declare @iCategorie char(1);
    select @iCategorie= idCategorie from deleted;
    update QuestionTrivia set idCategorie= null
    where idCategorie =@iCategorie;
    delete from CategorieTrivia where idCategorie =@iCategorie
end;
```

Pour tester, il faut s'assurer que la colonne idCategorie dans la table QuestionTrivia accepte les valeurs NULL. Sinon on exécute cette instruction en premier:

```
alter table questionTrivia alter column idCategorie char(1) null;
```

Puis, pour tester le trigger:

```
delete from CategorieTrivia where idCategorie = 'h';
```

Exemple 4

```
create TRIGGER ctrlSalairePermanent on EmpPermanent after update as
BEGIN
declare
@empno int,
@ancienne money,
@nouvelle money;
    select @empno =empno from deleted
        select @ancienne = Salaire from deleted where empno =@empno;
    select @nouvelle = Salaire from inserted where empno =@empno;
        IF (@ancienne > @nouvelle)
            begin
                rollback;
                RAISERROR (15600,-1,-1, 'Ne pas diminuer le salaire');
            end;
END;
```

Pour tester :

```
update EmpPermanent set Salaire =salaire -1 where empno =9;
```

- › Dans la table DELETED on retrouve l'ancien salaire de l'employé numéro 9. Le salaire est de 50000
- › Dans la table INSERTED on retrouve le nouveau salaire de l'employé numéro 9. Le salaire est 49999
- › Ce que nous demandons au trigger c'est de faire un ROLLBACK si le nouveau salaire est plus bas que l'ancien salaire.
- › Remarquez:
 - Dans l'instruction UPDATE, nous n'avons pas de BEGIN TRANSACTION, ce qui est normal puis que c'est une transaction autonome.
 - Nous avons un ROLLBACK dans le trigger. (Sans BEGIN Transaction)

Execution

Ce qui qu'il faut faire c'est: (commiter la transaction à l'extérieur du trigger)

begin transaction

update EmpPermanent set Salaire =salaire -1 where empno =9;

commit;

Ce qui qu'il ne faut absolument pas faire est «ELSE dans le trigger pour un COMMIT »

A ne pas faire dans un trigger. (ce qui est en rouge)

IF (@ancienne > @nouvelle) ROLLBACK **ELSE COMMIT**



Activer et désactiver un trigger

Parfois, il est nécessaire de désactiver un triggers pour pouvoir effectuer certaines opérations sur la table. Certains triggers pourrait également être en conflit. Au lieu de les détruire, on pourrait simplement les désactiver.

Syntaxe:

```
DISABLE TRIGGER {[ schema_name . ] trigger_name [ ,...n ] | ALL }  
ON { object_name | DATABASE | ALL SERVER } [ ; ]
```

Exemples:

```
disable trigger afterInsertemp ,VerfiferInsert on EmpPermanent;
```

```
disable trigger all on EmpPermanent
```

Pour Activer un trigger, utilisez: ENABLE. (c'est la même syntaxe:

```
enable trigger ctrlInsertionPermanent on EmpPermanent;
```

```
enable trigger all on EmpPermanent
```

RAISERROR

- › RAISERROR est une fonction qui génère un message erreur défini par l'utilisateur. Le message n'arrête pas le trigger (ce n'est pas comme Raise_Application_error d'Oracle).
- › Elle permet à un programmeur de mieux gérer ses messages.
- › RAISERROR(id_message, sévérité ,État ,'Message');

RAISERROR

π

- *id_message:*

indique le numéro du message. Il doit être compris entre **plus grand que 50 000**

- *Sévérité :*

indique le degré de gravité associé à l'erreur (ici le trigger), ce niveau de gravité est défini par l'utilisateur.

- Ce nombre se situe entre 0 et 25.
 - Les utilisateurs ne peuvent donner que le nombre entre **0 et 18**.
 - Les nombre entre 19 et 25 sont réservés aux membres du groupe sysadmin. Les nombre de 20 à 25 sont considérés comme fatals. Il est même possible que la connexion à la BD soit interrompue.
 - Si négatif ramené à la sévérité de l'erreur elle-même.
 - Pour les triggers la sévérité est 15 ou 16
 - Violation de contrainte CHECK et FK, sévérité =16
 - Duplication de clé primaire : sévérité 14
- Si vous prêtez attention aux messages erreurs renvoyés par le SGBD, vous constaterez qu'ils se présentent sous la forme du RAISERROR vous pouvez vous baser sur ces messages pour fixer le degré de sévérité.

État :

utilisé lorsque la même erreur définie par l'utilisateur se retrouve à plusieurs endroits, l'état qui est un numéro unique permet de retrouver la section du code ayant générée l'erreur. L'état est un nombre entre 0 et 255. Si négatifs alors ramené à 0.

Message :

représente le message défini par l'utilisateur. Au maximum 2047 caractères.

Vous pouvez également laisser le soin au SGBDR d'utiliser ses propres paramètres.

Exemple 1. On contrôle le nombre de bonnes réponses.

```
create or alter trigger CTRLnbBonnerep on ReponseTrivia after insert,  
update as  
begin  
declare @idQuestion int,  
@totalBonneRep smallint;  
select @idQuestion = idQuestion from inserted;  
  
select @totalBonneRep = count(idQuestion) from ReponseTrivia  
    where idQuestion =@idQuestion and estBonne = 'o';  
  
    if (@totalBonneRep > 1)  
        rollback;  
end;
```

Remarquez que nous avons juste ROLLBACK. L'erreur sera capturée dans la transaction

Exécution, avec les messages erreurs pour le trigger de l'exemple 1

```
begin try

    begin transaction

        insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('moi','o',114);
        insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('toi','o',114);
        insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('vous','n',114);
        insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('nous','n',114);

    commit;

end try

begin catch

if (@@TRANCOUNT>1) ROLLBACK; -- si pas de trigger ou trigger désactivé

select ERROR_MESSAGE() as Unmessage, ERROR_SEVERITY() as Gravité,
ERROR_STATE() as etat,@@TRANCOUNT as nb_transaction
```

message	Gravité	etat	nb_transaction
Paramètre ou option non valide pour la procédure 'Limite d'une bonne reponse par question'.	15	1	0

Exécution, avec les messages erreurs pour le trigger de l'exemple 1

```
begin try

    begin transaction

        insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('moi','o',114);
        insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('toi','o',114);
        insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('vous','n',114);
        insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('nous','n',114);

    commit;

end try

begin catch

if (@@TRANCOUNT>1) ROLLBACK; -- si pas de trigger

THROW --- permet d'afficher le ERROR_MESSAGE() uniquement

(0 lignes affectées)
Msg 547, Niveau 16, État 0, Ligne 17
L'instruction INSERT est en conflit avec la contrainte CHECK "ck_difficulte". Le conflit s'est produit dans la base de données "BDTrivi
```

Exécution, avec les messages erreurs pour le trigger de l'exemple 2

```
begin try

    begin transaction

    insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('moi','o',115);

    insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('toi','n',115);

    insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('vous','n',115);

    insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('nous','n',115);

    insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('nous','n',115);

    commit;

end try

begin catch

if (@@TRANCOUNT>=1) ROLLBACK; -- si pas de trigger ou trigger désactivé

select ERROR_MESSAGE() as message, ERROR_SEVERITY() as Gravité,

ERROR_STATE() as etat,@@TRANCOUNT as nb_transaction
```

message	Gravité	etat	nb_transaction
Paramètre ou option non valide pour la procédure 'Limite de 4 reponses par question.'	15	1	0

Points clés, triggers

- Les triggers sont un bon moyen pour garantir l'intégrité des données. Il ne faut pas en abuser.
- MS SQL Server manipule deux types de triggers: AFTER(FOR) et INSTEAD OF
- Vous avez un trigger INSTEAD OF par table.
- Les triggers sont définis sur une table. Lorsque la table est détruite (DROP), le trigger l'est aussi. (ce qui n'est pas le cas d'une procédure ou d'une fonction)
- Il n'y a pas de commande EXECUTE pour déclencher un trigger. C'est l'opération DML qui le déclenche.
- Avec MS SQL Server, lorsque l'opération DML n'est pas valide, le trigger ne l'arrête pas. Il faut faire un ROLLBACK de manière explicite. Dans ce cas, même s'il n'y a pas de BEGIN TRANSACTION, le rollback va se faire.
- Les transactions sont commitées à l'extérieur du trigger
- Les triggers ne sont pas des procédures stockées dans lesquelles vous allez faire vos transactions.

Retour sur la commande CREATE TABLE

- Lorsque le trigger est déclenché par une transaction utilisant un TRY CATCH alors dans le CATCH
 - Utilisez: `select ERROR_MESSAGE() as Unmessage, ERROR_SEVERITY() as Gravité, @@TRANCOUNT as nb_transaction` pour comprendre le message erreur en rapport à la transaction
 - Sinon , utilisez THROW pour comprendre pourquoi la transaction a échouée
 - Utilisez : `if (@@TRANCOUNT >=1) ROLLBACK.` Juste ROLLBACK pourrait renvoyer une erreur si le trigger a déjà exécuté un ROLLBACK
- Lors de votre conception, si vous avez déterminé que les enregistrements liés par la Foreign KEY doivent être supprimés car il s'agit d'un lien de composition, comme dans le cas d'un livre et ses chapitres, c'est-à-dire que lorsqu'un livre est supprimé alors tous les chapitres liés à ce livre doivent être également supprimé, alors vous pouvez le faire à la création de table. L'option ON DELETE CASCADE de la commande CREATE TABLE permet de faire cette suppression.

Exemple

Syntaxe et exemple:

```
create table livres
(
  coteLivre char(5),
  titre varchar(40) not null,
  langue varchar(20) not null,
  annee smallint not null,
  nbPages smallint not null,
  constraint pk_livre primary key(coteLivre)
);
```

Exemple

```
create table Chapitres  
(  
  idChapitre char(7) constraint pkChapitre primary key,  
  nomChapitre varchar(40) not null,  
  coteLivre char(5) not null,  
  constraint fk_Livre foreign key (coteLivre)  
  references livres(coteLivre) ON DELETE CASCADE  
)
```

Attention !! L'option ON DELETE CASCADE est irréversible. Ce n'est pas comme un trigger que vous pouvez **désactiver**.

Conclusion et bonnes pratiques

- Pour garantir l'intégrité, faites-le par la base de données au CREATE TABLE. Comme les PK, le FK, Les CHECK...C'est la meilleure façon. Pour la PK, lorsque c'est possible, utilisez IDENTITY.
- Donnez un nom significatif à vos contraintes d'intégrité.
- Les triggers sont là pour renforcer l'intégrité des données. Leur avantage est qu'on peut les désactiver au besoin. De plus ils s'exécutent automatiquement (même s'ils sont oubliés).
- Les procédures stockées sont un excellent moyen pour réduire les risques de briser l'intégrité des données, à condition qu'elles soient utilisées.
- À moins que ce soit obligé, évitez tout le temps le ON DELETE CASCADE.
- Dans ce cours, si vous devez utiliser ON DELETE CASCADE, nous vous le demanderons de manière EXPLICITE.

Conclusion et bonnes pratiques

- Éviter les SELECT *, c'est lourd
- Au lieu de faire des sous-requêtes, optez pour les jointures si c'est possible. Les SGBDs sont conçus pour les jointures.
- Utilisez le WHERE avant le GROUP BY. C'est plus rapide de grouper un ensemble d'enregistrements restreint.
- Utilisez des transactions explicites : BEGIN TRANSACTION /COMMIT TRANSACTION et privilégiez des transactions courtes. Les transactions longues utilisent un verrouillage plus long. (rappel: une transaction est un ensemble d'instructions **DML** qui doivent-êtré exécutées comme un tout)
- Utilisez des procédures stockées . Elles sont :
 - Plus rapide à l'exécution,
 - Préviennent les injections SQL
 - Facilite le travail d'équipe
 - Peuvent être réutilisée
 - Facilitent le débogage.
- À partir de MS SQL server 2016 vous avez la fonction TRY...CATCH, utilisez là si possible.
- Ne pas abuser de l'utilisation de triggers. Ils peuvent rentrer en conflit les uns, les autres.



CONCLUSION



QUESTIONS ??