420-KBA-LG, programmation de bases de données

Saliha Yacoub

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.

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.

Fonctionnement: Les tables INSERTED et DELETED

Exemple 1: trigger qui contrôle le nombre de bonne reponses par question.

```
create   trigger CTRLReponses on ReponseTrivia after  insert 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;
```

Fonctionnement: Les tables INSERTED et DELETED

Pour tester:

Begin transaction

insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Chat ','o',121);

insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Chien','o',121);

insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Cheval','n',121);

insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('Chacal','n',121);

commit;

Évidement il faudra compléter le trigger pour qu'il vérifie que le nombre de réponse n'excède pas 4 (à la limite soit exactement 4)

Fonctionnement: Les tables INSERTED et DELETED

Exemple 2, 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:

```
alter table questionTrivia alter column idCategorie char(1) null;
delete from CategorieTrivia where idCategorie ='g';
```

Fonctionnement: Les tables INSERTED et DELETED

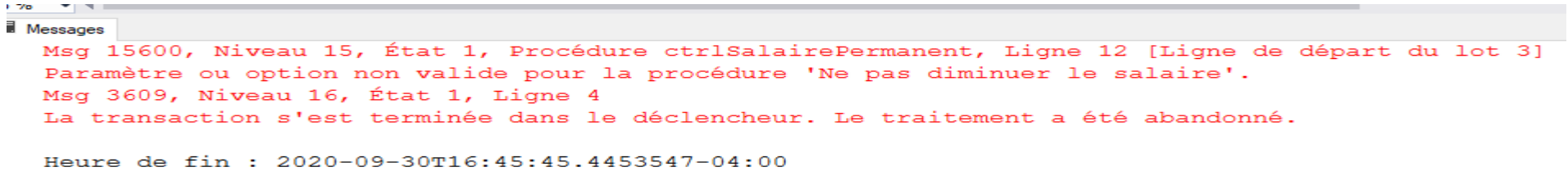
Exemple3, controle des salaires: Le salaire ne doit pas baisser

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

Fonctionnement: Les tables INSERTED et DELETED

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)

Fonctionnement: Les tables INSERTED et DELETED

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 synatxe:

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

- *id_message*, indique le numéro du message. Ce numéro doit être >50000. Lorsqu'il n'est pas indiqué ce numéro vaut 5000.

- *Sévérité* : indique le degré de gravité associé à l'erreur (ici le trigger), ce niveau de gravité est défini par l'utilisateurs.

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 1

- 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.

RAISERROR

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.

RAISERROR

```
begin try
        begin transaction
          insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('aloo','o',122);
          insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('a','o',122);
        insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('a','n',122);
          insert into ReponseTrivia(laReponse,estBonne,idQuestion) values('a','n',122);
        commit;
 end try
 begin catch
                select ERROR_MESSAGE() as message, ERROR_SEVERITY() as Gravité,
                ERROR_STATE() as etat,@@TRANCOUNT as nb_transaction

end catch;
```

RAISERROR

```
23      else if (@totalBonneRep > 1)
24          begin
25          rollback;
26          RAISERROR (15600,-1,-1, 'Limite de 1 bonne reponse par question'
27          end;
28
```

Résultats | Messages

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

Points clés:

- 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.

CONCLUSION

QUESTIONS ??