CENTRE DE RÉADAPTATION MULHOUSE
Rééducation et Formation Professionnelle

Triggers et Procédures Stockées

SQL SERVER

TABLE DES MATIERES

1. La notion de lot d'instructions ou batch 1
2. la notion de transaction 4
3. Les contrôles de flux 5
   3.1. Type de contrôle : le bloc d'instruction 5
   3.2. Type de contrôle : l'alternative 5
   3.3. Type de contrôle : l'aiguillage 6
   3.4. Type de contrôle : la répétitive 7
4. Utilisation de variables 9
   4.1. Les variables locales 9
   4.2. Les variables globales 11
5. Commande SQL complémentaires 12
   5.1. Affichage d'un message à l'écran 12
   5.2. Retour d'un message d'erreur 12
6. Les procédures stockées 14
   6.1. définition d'une procédure stockée 14
   6.2. création d'une procédure stockée 14
   6.3. Limite des procédures stockées 15
   6.4. Les procédures système 15
7. Les déclencheurs 16
   7.1. Définition d'un déclencheur 16
   7.2. Mise en place d'un trigger 16
      7.2.1. Précautions à prendre 16
      7.2.2. Principe de fonctionnement. 16
      7.2.3. Exemples de mise en place de triggers 18
      7.2.4. Triggers d' intégrité référentielle. 18
      7.2.5. Triggers traduisant des règles de gestion. 20
      7.2.6. Les triggers en cascade 21
      7.2.7. Suppression d' un trigger. 24
8. Utilisation d'un curseur 24

8.1. DECLARE CURSOR 25
8.2. OPEN 25
8.3. FETCH 25
8.4. CLOSE 25
8.5. DEALLOCATE 26
9. Annexe A : Variables systèmes 27
10. Annexe B : Liste des procédures système de SQL SERVER 32

1. LA NOTION DE LOT D'INSTRUCTIONS OU BATCH

Un lot d'instructions ou batch est un ensemble d'instructions SQL soumises simultanément et exécutées en groupe.

Un batch se termine toujours par l'instruction **GO**. Ainsi l'instruction GO peut être vu comme un séparateur de lots.

Les batch doivent respecter les règles suivantes:

- les instructions CREATE DEFAULT, CREATE PROCEDURE, CREATE RULE, CREATE TRIGGER, et CREATE VIEW ne peuvent être combinées avec d'autres instructions pour former un batch. Elles doivent être soumises individuellement;
- les règles et les valeurs par défaut doivent être créées ou associées à des colonnes dans un batch n°1, puis utilisées à l'intérieur d'un batch n°2.
- De même, les procédures système sp_bindrule et sp_bindefault ne peuvent coexister dans le même lot d'instructions que les instructions INSERT qui font appel à la règle ou à la valeur par défaut correspondante;
- les contraintes CHECK sont soumises aux mêmes principes que les règles et valeurs par défauts. Elles ne peuvent être créées et utilisées dans le même batch. Une table créée avec la contrainte CHECK ne peut appliquer ces contraintes dans le même lot d'instructions que la définition;
- vous ne pouvez, dans un même lot d'instructions, supprimer un objet et ensuite y faire référence ou le recréer;
- vous ne pouvez, dans un même lot d'instructions, agrandir une table et ensuite faire référence aux nouvelles colonnes;
- les options modifiées à l'aide de l'instruction SET n'entrent en vigueur qu'à la fin du lot d'instructions. Pour plus de détails, consultez l'instruction SET;

Vous pouvez soumettre les lots d'instructions par le biais d'un outil graphique tel que l'utilitaire ISQL/w, d'un utilitaire de ligne de commande tel qu'isql ou de fichiers (scripts) fournis

à isql. Un fichier soumis à isql peut comprendre plusieurs lots d'instructions SQL si chaque lot se termine par le séparateur de lot, à savoir la commande *GO*;

Un script est souvent une série de lots d'instructions soumis l'un après l'autre. Chaque lot étant terminé par l'instruction **GO**.

Règle générale à retenir :

" LA CREATION, MODIFICATION, DESTRUCTION D'UN OBJET DANS LA BASE DE DONNEES (TABLE, CONTRAINTES, VUE, INDEX, DECLENCHEUR, etc…) DOIT ETRE DANS UN LOT D'INSTRUCTION DISTINCT DU LOT ON L'ON SOUHAITE L'UTILISER OU LE RECREER "

EXEMPLE 1. INSTRUCTIONS SELECT MULTIPLES

L'exemple suivant illustre l'utilisation de plusieurs instructions SELECT à l'intérieur d'un même lot d'instructions. Deux jeux de résultats seront renvoyés.

```
SELECT COUNT(*) FROM titres
SELECT COUNT(*) FROM auteurs
```

Exemple 2. Création et utilisation d'un objet

L'exemple suivant illustre la création d'une table, une insertion, puis un SELECT.

```
CREATE TABLE test
(
        column1         char(10)        NOT NULL,
        column2         int                     NULL
)
INSERT test
        VALUES ('bonjour', 598)
SELECT * FROM test
```

Exemple 3. INSERT avec contraintes dans des lots d'instructions distincts

L'exemple suivant montre comment écrire des lots d'instructions contenant une définition de table et des insertions dans la table. Créez d'abord la table dans un lot différent de celui contenant la ou les instructions INSERT, et placez ensuite le ou les INSERT dans un lot distinct.

Cet INSERT échoue car 1234 n'est pas un numéro d'éditeur valide. En l'absence de l'instruction GO, ce contrôle ne serait pas fait car la contrainte ne serait pas prise en compte.

```
CREATE TABLE éditeurs
(
id_éditeur              char(4)                 NOT NULL
        CONSTRAINT UPKCL_pubind PRIMARY KEY CLUSTERED
        CHECK (id_éditeur in ('1389', '0736', '0877', '1622', '1756')
                OR id_éditeur like '99[0-9][0-9]'),
nom_éditeur     varchar(40)                     NULL,
ville           varchar(20)                     NULL,
région          char(2)                 NULL,
pays                    varchar(30)                     NULL
        DEFAULT         ('FR')
)
go

INSERT éditeurs (id_éditeur, nom_éditeur) VALUES('1234',
        'test Editeurs')
```

2. LA NOTION DE TRANSACTION

Au départ on considère que la base de données a des données cohérentes. Ensuite on effectue un ensemble de mises à jour (insertion, modification, destruction) sur une ou plusieurs lignes de une ou plusieurs tables. A l'arrivée on retrouve une base de données avec des données cohérentes. Mais si au moment de ces mises à jour un problème survient (exemple : coupure réseau) on obtient alors des mises à jour partielles et la base de données se retrouve dans un état où les données sont incohérentes. Ainsi la seule solution est de trouver un moyen de revenir à l'état de cohérence des données du départ. C'est le rôle d'une transaction.

Ainsi une transaction peut être vu comme un ensemble de mises à jour de données aboutissant à un nouvel état de cohérence de la base. Une transaction peut alors :

- soit s'effectuer complètement sans problème
- soit elle est interrompue pour une raison quelconque, et alors un mécanisme automatique de retour en arrière s'enclenche. Il permet de défaire l'ensemble des mises à jour partiellement effectuées et de retrouver l'état de cohérence des données existant au début de la transaction.

L'ensemble des instructions SQL permettant ces mises à jour sont encapsulées entre un **BEGIN TRANSACTION** et un **COMMIT TRANSACTION.**

BEGIN TRANSACTION nom_de_la_transaction

Ordre SQL1

Ordre SQL2

…

Ordre SQLN

COMMIT TRANSACTION nom_de_la_transaction

La commande SQL **BEGIN TRANSACTION** marque le début d'une transaction définie par l'utilisateur.

C'est à dire qu'à partir de cette instruction toutes les mises à jour de la base sont consignées dans le journal de transactions. Pour cela on écrit l'état de chaque ligne **avant** mise à jour et l'état **après** la mise à jour. Les 2 états sont enregistrés dans le journal avant l'écriture effective dans la base. Nous avons affaire à une écriture anticipée.

La commande SQL **COMMIT TRANSACTION** marque la fin d'une transaction définie par l'utilisateur.

Tant que la commande **COMMIT TRANSACTION** n'a pas été exécutée les modifications ne sont pas complètement enregistrés dans la base.

Avec le journal de transaction on peut défaire une transaction :

- soit de manière automatique en cas d'un incident imprévu survenu au cours de la transaction
- soit de manière volontaire, par programmation en utilisant la commande SQL **ROLLBACK TRANSACTION** nom_de_la_transaction

Ainsi **ROLLBACK TRANSACTION** annule une transaction définie par l'utilisateur jusqu'au dernier point de reprise défini à l'intérieur de la transaction ou au début.

3. LES CONTROLES DE FLUX

Les contrôles de flux permet de structurer le déroulement :

- d'un batch
- d'un script
- d'une procédure stockée
- d'un déclencheur

4. TYPE DE CONTROLE : LE BLOC D'INSTRUCTION

Délimite une série d'instructions SQL de sorte que des éléments de langage de contrôle de flux tels qu' IF...ELSE affectent l'exécution d'un groupe d'instructions, et non uniquement l'instruction qui suit immédiatement IF...ELSE.

Les blocs BEGIN...END peuvent être imbriqués.

Syntaxe

```
BEGIN
    instruction_SQL1
    instruction_SQL2
    …
    BEGIN
        instruction_SQL1
        instruction_SQL2
        …
    END

    instruction_SQLN
END
```

5. TYPE DE CONTROLE : L'ALTERNATIVE

Impose des conditions à l'exécution d'une instruction SQL. L'instruction SQL suivant le mot-clé **IF** et sa condition n'est exécutée que si cette dernière est remplie (lorsque l'expression booléenne renvoie vrai, **TRUE**). Le mot-clé **ELSE** est facultatif et introduit une instruction SQL de remplacement qui n'est exécutée que si la condition IF n'est pas remplie (lorsque l'expression booléenne renvoie faux, **FALSE**).

Syntaxe

IF expression_booléenne

{instruction_SQL | bloc_d'instructions}

ELSE

{instruction_SQL | bloc_d'instructions}

6. TYPE DE CONTROLE : L'AIGUILLAGE

L'expression CASE permet de simplifier les expressions SQL pour des valeurs conditionnelles.

Syntaxe

Expression CASE simple:

CASE expression

WHEN expression1 **THEN** expression1

WHEN expression2 **THEN** expression2

…

WHEN expressionN **THEN** expressionN

ELSE expression_dans_tous_les_autres_cas

END

Expression de recherche CASE:

CASE

WHEN expression_booléenne1 **THEN** expression1

WHEN expression_booléenne2 **THEN** expression2

…

WHEN expression_booléenneN **THEN** expressionN

ELSE expression_dans_tous_les_autres_cas

END

Exemple

SELECT "Catégorie de prix" =

CASE

WHEN prix **IS NULL THEN** 'Pas d''estimation'

```
                WHEN prix < 50 THEN 'Prix modéré'
            WHEN prix >= 50 AND prix < 140 THEN 'Prix raisonnable'
                ELSE 'Livre très coûteux!'
            END,
        "Titre abrégé" = CONVERT(varchar(20), titre),
        Catégorie =
            CASE type
                WHEN 'informatique' THEN 'Informatique'
                WHEN 'cui_moderne' THEN 'Cuisine moderne'
                WHEN 'gestion' THEN 'Gestion'
                WHEN 'psychologie' THEN 'Psychologie'
                WHEN 'cui_traditio' THEN 'Cuisine traditionnelle'
                ELSE 'Hors catégorie'
            END
FROM titres
ORDER BY prix
GO
```

EXEMPLE DE RESULTAT

```
Catégorie de prix     Titre abrégé         Catégorie
--------------------- -------------------- -------------------
Pas d'estimation      Guide des bonnes man  Informatique
Pas d'estimation      La psychologie des o  Hors  catégorie
Prix modéré           Les micro-ondes par   Cuisine moderne
Prix modéré           Le stress en informa  Gestion
Prix modéré           Vivre sans crainte    Psychologie
Prix raisonnable      Equilibre émotionnel  Psychologie
Prix raisonnable      La colère : notre en  Psychologie
Prix raisonnable      Cinquante ans dans l  Cuisine traditionnelle
Prix raisonnable      La cuisine - l'ordin  Gestion
```

7. TYPE DE CONTROLE : LA REPETITIVE

Fixe une condition à l'exécution répétitive d'une instruction_SQL ou d'un bloc d'instructions. L'instruction est exécutée de manière répétitive aussi longtemps que la condition précisée est logiquement vraie. L'exécution des instructions d'une boucle **WHILE** peut être contrôlée depuis l'intérieur de la boucle à l'aide des mots-clés **BREAK** et **CONTINUE**.

BREAK

Provoque la sortie de la boucle **WHILE**. **BREAK** est souvent (pas toujours) activé par un test IF. L'exécution se poursuit à la première instruction suivant le mot clé **END** marquant la fin de la boucle.

CONTINUE

Renvoie au début de la boucle **WHILE**, ce qui empêche l'exécution des instructions placées après **CONTINUE**. **CONTINUE** est souvent (pas toujours) activé par un test **IF**.

Syntaxe

```
WHILE expression_booléenne
BEGIN
        [BREAK]
        {instruction_SQL | bloc_d'instructions}
        [CONTINUE]
END
```

Exemple

```
WHILE (SELECT AVG(prix) FROM titres) < 200
BEGIN
        UPDATE titres
                SET prix = prix * 2
        SELECT MAX(prix) FROM titres
        IF (SELECT MAX(prix) FROM titres) > 300
                BREAK
        ELSE
                CONTINUE
END
PRINT 'Prix trop élevés pour le marché'
```

8. UTILISATION DE VARIABLES

Les variables sont des entités définies ou non par l'utilisateur auxquelles des valeurs sont affectées. Elles sont toutes définies par un nom, un type de données, et un sens par rapport à l'application développée.

9. LES VARIABLES LOCALES

Les variables locales sont déclarées au sein d'un lot ou d'une procédure au moyen de l'instruction **DECLARE**, et l'instruction **SELECT** leur affecte des valeurs.

Les variables locales sont souvent utilisées dans un lot ou une procédure en tant que compteurs pour des boucles WHILE ou pour des blocs IF...ELSE.

Le nom d'une variable locale doit toujours commencer par un seul **@** et ne doit pas dépasser 29 caractères.

Les variables locales ne peuvent être utilisées que dans la procédure ou le lot dans lequel elles sont déclarées. Elles sont donc **locales** à une procédure ou un lot .

Syntaxe

Déclaration de la variable:

```
DECLARE     @nom_de_la_variable1  type_de_données1 ,
            @nom_de_la_variable2  type_de_données2 ,
            …
            @nom_de_la_variableN type_de_donnéesN
```

Affectation de valeurs à la variable:

```
SELECT @variable1 = {expression | instruction_de_sélection} ,
          @variable2 = {expression | instruction_de_sélection}
         …
[FROM liste_de_tables]
[WHERE critères_de_sélection]
[Clause GROUP BY]
[Clause HAVING]
[Clause ORDER BY]
```

Exemple1:

```
DECLARE @nombre int
SELECT @nombre=1
```

Exemple2:

```
@echeance=(SELECT dateEcheance FROM contrat WHERE numContrat=125)
```

L'instruction **SELECT** qui affecte une valeur à la variable locale renvoie généralement une seule valeur. Si le résultat de l'affectation par l'instruction **SELECT** renvoie plus d'une valeur, c'est la dernière valeur renvoyée qui est affectée à la variable. Si le résultat de l'affectation par l'instruction **SELECT** ne renvoie aucune ligne, la variable conserve sa valeur actuelle, sauf si l'affectation se produit dans une sous-requête. Ce n'est que si la sous-requête ne renvoie aucune ligne que la variable se voit affecter la valeur **NULL**.

10. LES VARIABLES GLOBALES

Les variables globales sont prédéfinies. Elles peuvent donc être utilisées sans déclaration. L'utilisateur ne peut pas créer de variables globales. Une variable globale peut être utilisée dans n'importe quelle procédure ou bloc. Certaines variables globales ont une valeur qui est initialisée lorsqu'un utilisateur ouvre une session. La liste des variables globales existantes se trouvent en Annexe **A**.

11. COMMANDE SQL COMPLEMENTAIRES

12. AFFICHAGE D'UN MESSAGE A L'ECRAN

Syntaxe

PRINT {'texte quelconque en ASCII' | @variable_locale | @@variable_globale}

Le message peut compter jusqu'à 255 caractères.

La variable doit être du type char ou varchar et doit être déclarée au niveau du lot ou de la procédure où elle est utilisée.

EXEMPLE 1. COMMANDE PRINT EXECUTEE SOUS CONDITIONS

```
IF EXISTS (SELECT code_postal FROM auteurs WHERE code_postal = '33000')
        PRINT 'auteur bordelais'
```

EXEMPLE 2. CONSTRUCTION ET AFFICHAGE D'UNE CHAINE

Cet exemple définit une variable locale (@message_de_retour) et construit ensuite la chaîne (en combinant des fonctions et des chaînes) à afficher.

```
DECLARE @message_de_retour varchar(255)
SELECT @message_de_retour= 'Mise à jour effectuée le ' +
                                RTRIM(CONVERT(char(30), GETDATE())) + '.'
PRINT @message_de_retour
```

13. RETOUR D'UN MESSAGE D'ERREUR

Syntaxe

RAISERROR ({n° message | message}, gravité, état [,argument1] [,argument2] [WITH LOG]

Permet d'envoyer un message d'erreur, au format des messages système, défini par l'utilisateur ou présent dans la table des messages *sysmessages*.

Les numéros de messages définis par l'utilisateur doivent être supérieurs à 50000.

Le message peut être paramétré; les arguments doivent être alors renseignés.

Exemple1

```
IF @code < 100
    RAISERROR ("Le code doit être inférieur à 100")
```

ou

```
IF @code < 100
    RAISERROR  (60000 "Le code doit être inférieur à 100")
```

Dans ce dernier cas, le numéro 60000 est affecté à la variable globale @@ERROR

Exemple2

```
RAISERROR (15252, 16, 1)
```

Affiche le message de numéro 15252 de la table *sysmessages*.

Exemple3

```
RAISERROR (15218, 16, 1, @table)
```

Affiche le message de numéro 15218 de la table *sysmessages* avec l'argument @table.

14. LES PROCEDURES STOCKEES

15. DEFINITION D'UNE PROCEDURE STOCKEE

Une procédure stockée est une collection d'instructions SQL pré compilées, qui peut admettre et/ou renvoyer des paramètres définis par l'utilisateur. Ainsi il peut y avoir un échange de paramètres.

Les procédures stockées ne sont compilées que lors de leur première exécution. Elles permettent d'accroître les performances et la cohérence lors de l'exécution de tâches récurrentes. Les contrôles de syntaxe et la compilation sont effectués une seule fois.

Leur exécution est déclenchée de manière volontaire grâce à la commande:

EXECUTE *nom_procédure* [liste paramètres]

Les avantages des procédures stockées sont les suivantes :

- l'exécution compilée est plus rapide
- elles peuvent être exécutées à partir de toute application frontale comme Visual Basic
- la maintenance des procédures est centralisée au niveau du SGBDR
- l'exécution d'une procédure est locale au serveur, seuls les paramètres sont transmis au client

16. CREATION D'UNE PROCEDURE STOCKEE

Syntaxe

CREATE PROCEDURE nom_de_procédure

@nom_de_paramètre1 type_de_données1 [= valeur_par_défaut1] [OUTPUT] **,**

@nom_de_paramètre2 type_de_données2 [= valeur_par_défaut2] [OUTPUT] **,**

@nom_de_paramètre3 type_de_données3 [= valeur_par_défaut3] [OUTPUT] **,**

…

@nom_de_paramètre255 type_de_données255 [= valeur_par_défaut255] [OUTPUT]

AS instructions_SQL

Tous les types de données pour un paramètre, à l'exception du type image, sont acceptés.

Le nom d'une table ou d'un objet de la base de donnée ne peut être passé en paramètre.

Valeur par défaut d'un paramètre: Si une valeur par défaut est définie, l'utilisateur peut exécuter la procédure sans fournir de paramètre. La valeur par défaut, qui doit être une constante, peut contenir les caractères génériques (%, _, [] et [^]) si la procédure emploie le nom du paramètre avec le mot-clé LIKE.

L'option **OUTPUT** indique que le paramètre est un paramètre de retour. La valeur spécifiée dans cette option peut être renvoyée à l'instruction **EXECUTE** qui a appelé la procédure. Les paramètres de retour permettent de renvoyer des informations à la procédure appelante. Dans l'instruction **EXECUTE**, le nom d'un paramètre en sortie doit également être suivi du mot **OUTPUT:**

```
EXECUTE RechercheOuvrages @editeur, @nombre OUTPUT
```

Exécution d'une procédure *RechercheOuvrages* en passant comme paramètre un nom d'éditeur et récupérant le nombre d'ouvrages de cet éditeur.

L'option **OUTPUT** ne peut être définie pour un paramètre de type text.

Une imbrication de procédures stockées a lieu quand une procédure stockée en appelle une autre. Le niveau d'imbrication est incrémenté quand la procédure appelée commence à s'exécuter, puis décrémenté quand son exécution se termine. Au-delà de 16 niveaux d'imbrication, la transaction toute entière échoue. Le niveau d'imbrication courant est stocké dans la variable globale **@@NESTLEVEL**.

Le texte d'une procédure stockée ne peut dépasser 65 025 caractères. Cette limite est imposée parce que le texte est stocké dans les tables système syscomments, où chaque procédure peut occuper 255 lignes de 255 octets chacune (255 * 255 = 65 025).

17. LIMITE DES PROCEDURES STOCKEES

Les objets référencés ou utilisées dans une procédure doivent être créés au préalable.

Les objets utilisés par une procédure ne peuvent pas être supprimés et recréés sous le même nom au sein d'une procédure.

Le corps d'une procédure est en fait considéré comme une transaction.

18. LES PROCEDURES SYSTEME

Ce sont des procédures stockées préexistantes qui commencent toutes par **sp_** (comme system procedure).

Elles facilitent l'accès aux fonctions d'administration et de gestion de la base ainsi que l'accès aux objets de la base et à leurs propriétés.

La liste des procédures systèmes sont en **Annexe B**.

19. LES DECLENCHEURS

20. DEFINITION D'UN DECLENCHEUR

Forme évoluée de règles utilisés pour renforcer l' intégrité de la base de données. Le plus souvent on parle de **TRIGGERS.**

Les triggers sont stockés avec les données dans la base.

Principes :

Les triggers sont un type particulier de procédure mémorisée.

- sont attachés à des tables
- réagissent aux fonctions de création (create), modification (update) et suppression (delete)
- ne peuvent pas être appelés explicitement dans les applications

Les triggers sont déclenchés automatiquement par le noyau SQL à chaque intervention sur la table qui les supportent.

Pas de possibilités de passer outre quel que soit l'utilisateur.

21. MISE EN PLACE D'UN TRIGGER

22. PRECAUTIONS A PRENDRE

Un trigger est toujours associé à une table, il est impossible de l' associer à une vue ou a une table temporaire.

Une table peut avoir au maximum trois triggers, chacun étant déclenché en fonction de

- Insertion de ligne
- Modification de ligne
- Suppression de ligne

Chaque trigger est facultatif et une table peut n' avoir aucun trigger

D'autre part, un trigger ne s'applique qu'à une seule table, plusieurs tables ne pouvant pas avoir le même trigger.

La suppression d'une table entraîne la destruction de ses triggers

23. PRINCIPE DE FONCTIONNEMENT.

Deux tables virtuelles sont créées au moment de la MAJ sur la table:

INSERTED

DELETED

Elles sont destinées à contenir les lignes de la table sur lesquelles elles ont été effectuées. Les tables INSERTED et DELETED peuvent être utilisées par le trigger pour déterminer comment le traitement doit se dérouler. Ce traitement est à écrire par le développeur. Les mécanismes d'utilisation par SQL Server de ces deux tables diffèrent en fonction de l'opération effectuée sur la table contenant le ou les triggers.

23.1.1.1. CAS DE SUPPRESSION D'UNE LIGNE DE TABLE (DELETE)

La/les lignes supprimées sont placées dans la table temporaire DELETED et supprimées de la table; la table DELETED et les tables de la base ne peuvent pas avoir de lignes en commun.

23.1.1.2. 7.2.2.2 CAS DE CREATION D'UNE LIGNE DE TABLE (INSERT)

La/les lignes nouvelles sont placées dans la table temporaire INSERTED et dans la table réelle ; toutes les lignes de la table INSERTED apparaissent dans la table de la base.

23.1.1.3. 7.2.2.3 CAS DE MODIFICATION D'UNE LIGNE DE TABLE (UPDATE)

La/les lignes avant modification sont placées dans la table temporaire DELETED et la/les lignes après modification sont placées dans la table temporaire INSERTED et dans la table

Un trigger est un objet SQL Server, en conséquence de quoi:

- les triggers seront gérés de manière autonome
- chaque trigger porte un nom
- chaque trigger possède un certain nombre de propriétés.

Les triggers doivent être créés juste après la création de la table de sorte que l' intégrité référentielle soit respectée.

- de manière interactive grâce à l' interface graphique
- en utilisant les procédures de SQL Server.

Utiliser la procédure **create trigger** :

Syntaxe :

CREATE TRIGGER *nom de trigger*

ON *nom de table*

FOR {INSERT, UPDATE, DELETE}

AS

séquence1 d'ordres SQL

INSERT, UPDATE et DELETE précisent l'instruction de mise à jour qui, lorsqu'elle sera appliquée à cette table, déclenchera l'exécution du trigger. Un même trigger peut s'appliquer à plusieurs types de mises à jour.

24. EXEMPLES DE MISE EN PLACE DE TRIGGERS

Les triggers définis sur la base de données ont deux rôles principaux:

- contrôler que les données manipulées vérifient l intégrité référentielle

Exemple: créer un nouveau salarié nécessite de l'affecter à un service existant.

- mettre en œuvre des traitements correspondant à une règle de gestion de l'entreprise

Exemple: lorsqu'un produit est passé en commande, on mettra à jour la quantité disponible en stock dans la table Produits.

25. TRIGGERS D' INTEGRITE REFERENTIELLE.

Exemple 1 :

" Un avion ne peut être créé que si le modèle qui lui correspond existe déjà dans la base"

```
CREATE TRIGGER TI_Avion ON AVION FOR INSERT AS
        BEGIN
        IF (SELECT count(*) FROM MODELE x, inserted
        WHERE x.CODEMODELE = inserted.CODEMODELE_MODELE) = 0
                BEGIN
                PRINT 'Déclencheur Insert sur AVION. Modèle inexistant. Création impossible'
                ROLLBACK TRANSACTION
                RETURN
                END
        END
```

Exemple 2 :

" Un modèle ne peut être supprimé que si aucun avion ne correspond à ce modèle "

```
CREATE TRIGGER TD_Modele ON MODELE FOR DELETE AS
        BEGIN
        IF (SELECT count(*) FROM AVION Ref, deleted
        WHERE Ref.CODEMODELE_MODELE = deleted.CODEMODELE) > 0
                BEGIN
                PRINT 'Déclencheur Delete sur modèle. Il existe des avions correspondant au
                ROLLBACK TRANSACTION
                RETURN
                END
        END
```

Exemple 3 .

“ En cas de modification des données concernant un avion, il est nécessaire de contrôler si le modèle qui lui correspond existe effectivement dans la base; mais cette vérification n’ est à faire que si la clé étrangère dans la table avion a été modifiée. ”

```
CREATE TRIGGER TU_Avion ON AVION FOR UPDATE AS
        BEGIN
        IF UPDATE (CODEMODELE_MODELE)
                BEGIN
                IF (SELECT count(*) FROM MODELE Ref, inserted
                WHERE Ref.CODEMODELE = inserted.CODEMODELE_MODELE) = 0
                        BEGIN
                        PRINT 'Déclencheur Update sur Avion. Modèle inexistant, vous ne pouvez pas le modifier.'
                        ROLLBACK TRANSACTION
                        RETURN
                        END
                END
        END
```

Remarquez l'instruction **IF UPDATE** (cham*p de la table*) permettant de tester si un champ précis est affecté par la modification

26. TRIGGERS TRADUISANT DES REGLES DE GESTION.

Certaines règles de gestion concernant les données de la base peuvent être prises en compte dans le base de données elle-même, de telle manière que la synchronisation des données soit indépendante des traitements effectués.

Exemple 4.

“ Un vol effectué à une certaine date ne doit embarquer qu’ un pilote et un seul ”

```
/*----------------------------------------------------------------------*/
/* Règle de gestion: un vol embarque un pilote et un seul */
/* Mise en place sur TRIGGER INSERT  de la table EMBARQUE*/
/*----------------------------------------------------------------------*/

IF(select count(*) from EMBARQUE, PILOTE, inserted
where EMBARQUE.NOVOL_VOL = inserted.NOVOL_VOL
and EMBARQUE.DATE_DATE = inserted.DATE_DATE
and EMBARQUE.NOPERS_PERSONNEL= PILOTE.NOPERS_PERSONNEL
) >1
```

```
BEGIN
PRINT 'Il y a déjà un pilote sur ce vol'
ROLLBACK TRANSACTION
END
```

Exemple 5

“Un modèle d’appareil n’existant plus dans la compagnie, on souhaite ne pas conserver aux pilotes la qualification correspondante”

```
CREATE TRIGGER TD_Modele ON MODELE FOR DELETE AS
        BEGIN
        IF (SELECT count(*) FROM AVION Ref, deleted
        WHERE Ref.CODEMODELE_MODELE = deleted.CODEMODELE) > 0
                BEGIN
                PRINT 'Déclencheur Delete sur modèle. Il existe des avions correspondant au modèle'
                ROLLBACK TRANSACTION
                RETURN
                END
        ELSE
                BEGIN
                DELETE QUALIFICATION FROM QUALIFICATION, DELETED
                WHERE CODEMODELE_MODELE = DELETED.CODEMODELE
                PRINT 'Qualifications correspondant au modèle. Suppression effectuée.'
                END
        END
```

27. LES TRIGGERS EN CASCADE

Une opération sur une table déclenche automatiquement, s’il existe, le trigger correspondant. Si le trigger déclenché effectue une opération sur une autre table, les triggers associés à cette table se déclencheront. Le nombre de niveaux cascadés est fonction du SGBD (16 en SQL Server).

Exemple :

“ Dans la base de données Avions; on décide qu’à la suppression d’un avion, on vérifiera s’il existe d’autres avions du même modèle. S’il s’agit du dernier avion de ce modèle dans la base, on détruira le modèle en question de la table des modèle. Hors le fait de détruire un modèle entraîne la destruction par trigger des lignes de qualification des pilotes pour ce modèle. (cf exemple 5)

On est en pleine cascade!!

Mise en oeuvre.

```
CREATE TRIGGER TD_AVION ON AVION FOR DELETE AS
        BEGIN
        declare @mod char(3)
        IF(SELECT COUNT(*) FROM AVION, deleted
        WHERE AVION.CODEMODELE_MODELE = DELETED.CODEMODELE_MODELE) =0
                BEGIN
                SELECT @mod = (SELECT CODEMODELE_MODELE from deleted)
                PRINT 'Déclencheur DELETE sur Avion. Plus aucun avion pour le modèle'
                PRINT @mod
                DELETE MODELE WHERE CODEMODELE = @mod
                END
        END

CREATE TRIGGER TD_Modele ON MODELE FOR DELETE AS
        BEGIN
        IF (SELECT count(*) FROM AVION Ref, deleted
        WHERE Ref.CODEMODELE_MODELE = deleted.CODEMODELE) > 0
                BEGIN
                PRINT 'Déclencheur Delete sur modèle. Il existe des avions correspondant au modèle'
                ROLLBACK TRANSACTION
                RETURN
                END
        ELSE
                BEGIN
                DELETE QUALIFICATION FROM QUALIFICATION, DELETED
                WHERE CODEMODELE_MODELE = DELETED.CODEMODELE
                PRINT 'Qualifications correspondant au modèle. Suppression demandée.'
                END
        END

CREATE TRIGGER TD_QUALIFICATION ON QUALIFICATION FOR DELETE AS
        BEGIN
        PRINT 'Déclencheur DELETE sur Qualif. Suppression effectuée.'
        END
```

28. SUPPRESSION D' UN TRIGGER.

Utiliser la procédure SQL Server DROP TRIGGER

Syntaxe : DROP TRIGGER *nom de trigger*

29. UTILISATION D'UN CURSEUR

Dans tout ce qui a précédé, nous avons obtenu un résultat global, nous n'avons pas pu faire de traitement ligne par ligne. Il est évident que la machine traite ligne par ligne mais elle ne vous redonne la main que lorsqu'elle a été du début à la fin de sa démarche.

Prenons le cas de l'instruction SELECT, on ne peut exécuter une autre commande que lorsque le SELECT est terminé.

Pour pouvoir exécuter d'autres commandes entre chaque ligne, nous utiliserons un curseur.

Il existe différentes étapes à suivre pour utiliser un curseur:

- le déclarer (**declare cursor**) afin de définir quel sera son contenu
  Declare cursor contient un ordre SELECT qui définit les données auxquelles on souhaitent accéder

- l'ouvrir (**open**)comme vous le feriez pour un tiroir quand vous voulez aller voir ce qu'il contient.

- le consulter (**fetch**) : cet ordre permet de passer à la suite de l'extraction demandée par le SELECT c'est à dire d'accéder ligne par ligne aux lignes issues du SELECT.

- le fermer (**close**) normal, on l'avait ouvert avant.

- le déallouer(**deallocate**): libère les ressources qui étaient liées au curseur (déverrouillage et libération pour d'autres utilisations)

30. DECLARE CURSOR

```
DECLARE MonCurseur CURSOR FOR
    SELECT nomActeur, prenomActeur from  FROM ActeursAuthors WHERE
    NumActeur < 100;
```

Si une mise à jour doit être effectuée sur les lignes récupérées par SELECT la clause FOR UPDATE [OF liste de colonnes] doit être indiquées dans la déclaration du curseur. La liste de colonnes permet de limiter les champs pouvant être modifier.

30.1. OPEN

Ouvre le curseur pour que l'on puisse accéder aux lignes issues du SELECT.

```
OPEN MonCurseur
```

30.2. FETCH

Permet d'accéder à une ligne depuis le curseur.

- FETCH FIRST récupère la première ligne issue du SELECT déclaré dans le curseur.
- FETCH NEXT récupère la ligne suivant celle qui vient d'être lue via le curseur (Next est pris par défaut)
- FETCH PRIOR récupère la ligne précédant celle qui vient d'être lue via le curseur
- FETCH LAST récupère la dernière ligne issue du SELECT déclaré dans le curseur.
- FETCH ABSOLUTE *n* récupère la n^{ième} ligne issue du SELECT via le curseur.
- FETCH RELATIVE *n* recupère la n^{ième} ligne suivant celle qui vient d'être lue via le curseur

```
FETCH MonCurseur
   extrait la ligne suivante

FETCH MonCurseur INTO @nom, @prenom
   extrait la ligne suivante en plaçant les données dans des variables

FETCH retourne  un status dans la variable globale FETCH_STATUS:
FETCH_STATUS=0: Fetch a abouti
FETCH_STATUS=-1: Fin de données
```

30.3. CLOSE

Ferme le curseur.

```
CLOSE MonCurseur
```

30.4. DEALLOCATE

```
DEALLOCATE MonCurseur
```

Exemple

Pour obtenir la liste des acteurs ayant un cachet < 500000

```
declare @CachetMaxi int
declare @NomActeur varchar(30)
set @CachetMaxi = 500000

   DECLARE Mon_Cursor CURSOR FOR
   SELECT NomActeur from Acteurs a, Participation p, Films f
         where a.NumActeur = p.NumActeur
           and P.CodFilm   = f.Codfilm
           and a.Cachet    < @cachetMaxi
   OPEN  Mon_Cursor

   FETCH  Mon_Cursor  INTO @NomActeur

   WHILE @@FETCH_STATUS = 0
    BEGIN
      PRINT 'le nom de l''acteur est : ' + @NomActeur
      FETCH NEXT FROM Mon_Cursor
    END

   CLOSE Mon_Cursor
   DEALLOCATE Mon_Cursor

GO
```

31. ANNEXE A : VARIABLES SYSTEMES

Les variables avec un (*) ont une instance initialisée à chaque ouverture de session par un utilisateur.

@@CONNECTIONS

Nombre de connexions ou de tentatives de connexion depuis le dernier démarrage de SQL Server.

@@CPU_BUSY

Temps, en nombre de pulsations (un trois-centième de seconde, ou 3,33 millisecondes), consacré par l'unité centrale à SQL Server depuis le dernier démarrage de celui-ci.

@@CURSOR_ROWS*

Nombre de lignes affectées dans le dernier curseur ouvert. **@@CURSOR_ROWS** renvoie:

-m Si le curseur est chargé de façon asynchrone. La valeur renvoyée (-m) est le nombre de lignes actuellement contenues dans l'ensemble du jeu de clés.

-n Si le curseur est totalement chargé. La valeur renvoyée (n) est le nombre total de lignes.

Si aucun curseur n'a été ouvert ou si le dernier curseur ouvert a été fermé ou désaffecté.

@@DATEFIRST

Renvoie la valeur courante du paramètre SET DATEFIRST. Indique le premier jour de chaque semaine: 1 pour lundi, 2 pour mardi, et ainsi de suite jusqu'à dimanche.

@@DBTS

Valeur du type de données timestamp courant pour la base de données. Cette valeur timestamp est unique pour la base de données.

@@ERROR*

Dernier numéro d'erreur générée par le système pour la connexion de l'utilisateur. La variable globale @@ERROR est généralement utilisée pour vérifier le statut d'erreur (réussite ou échec) de l'instruction la plus récemment exécutée. Elle contient 0 si la dernière instruction a été exécutée sans erreur. L'utilisation de @@ERROR avec des instructions de contrôle de flux constitue une méthode efficace de gestion des erreurs. L'instruction IF @@ERROR < > 0 RETURN est vraie si une erreur est rencontrée.

@@FETCH_STATUS*

Contient le statut d'une commande de curseur FETCH. Contient la valeur 0 si le fetch aboutit, - 1 si le fetch a échoué ou si la ligne dépassait le jeu de résultats, - 2 si la ligne recherchée n'est pas trouvée.

@@IDENTITY*

Enregistre la dernière valeur IDENTITY insérée. La variable @@IDENTITY est mise à jour de façon spécifique pour chaque utilisateur lorsqu'une instruction INSERT ou SELECT INTO ou une copie par bloc conduisant à une insertion dans une table est exécutée. Si une instruction modifie la table sans colonne identité, @@IDENTITY prend la valeur NULL. La valeur d'@@IDENTITY ne reprend pas une valeur précédente si l'instruction INSERT ou SELECT INTO ou la copie par bloc échoue, ou si la transaction revient en arrière. Pour plus d'informations, reportez-vous à l'instruction IDENTITY

@@IDLE

Temps, exprimé en nombre de pulsations (un trois-centième de seconde, ou 3,33 millisecondes), pendant lequel SQL Server est resté inactif depuis son dernier démarrage.

@@IO_BUSY

Temps, exprimé en nombre de pulsations (un trois-centième de seconde, ou 3,33 millisecondes), consacré par SQL Server à effectuer des opérations d'entrée/sortie depuis son dernier démarrage.

@@LANGID*

Identificateur local de la langue actuellement utilisée (définie dans syslanguages.langid).

@@LANGUAGE*

Langue actuellement utilisée (définie dans syslanguages.name).

@@MAX_CONNECTIONS

Nombre maximal de connexions simultanées qu'il est possible d'établir avec SQL Server dans l'environnement informatique courant. L'utilisateur peut configurer SQL Server pour qu'il admette moins de connexions au moyen de la procédure système sp_configure. @@MAX_CONNECTIONS n'équivaut pas nécessairement au nombre actuellement configuré.

@@MAX_PRECISION

Renvoie le niveau de précision utilisé par les types de données decimal et numeric tel qu'il est actuellement défini sur le serveur. Par défaut, la précision maximale est égale à 28; toutefois, il est possible de définir une plus grande précision au démarrage de SQL Server au moyen du paramètre /p ajouté à sqlservr. Pour plus d'informations, reportez-vous à la rubrique Utilitaires et exécutables.

@@MICROSOFTVERSION

Version à usage interne, servant à détecter la version courante du serveur. Si le contrôle des versions est nécessaire, utilisez @@VERSION.

@@NESTLEVEL*

Niveau d'imbrication de l'instruction en cours d'exécution (initialement 0). Chaque fois qu'une procédure stockée en appelle une autre, le niveau d'imbrication est incrémenté. En cas de dépassement du maximum autorisé (16), la transaction s'arrête.

@@PACK_RECEIVED

Nombre de paquets entrants lus par SQL Server depuis son dernier démarrage.

@@PACK_SENT

Nombre de paquets sortants écrits par SQL Server depuis son dernier démarrage.

@@PACKET_ERRORS

Nombre d'erreurs qui se sont produites alors que SQL Server envoyait ou recevait des paquets depuis son dernier démarrage.

@@PROCID*

Identificateur de la procédure stockée de la procédure en cours d'exécution.

@@REMSERVER*

Renvoie le nom du serveur contenu dans l'enregistrement des noms d'accès d'un serveur distant.

@@ROWCOUNT*

Nombre de lignes affectées par la dernière instruction. Cette variable est mise à 0 par les instructions qui ne renvoient pas de lignes, telles que l'instruction IF.

@@SERVERNAME

Nom du serveur SQL local. Vous devez définir ce nom au moyen de la procédure système sp_addserver, puis redémarrer SQL Server. Durant l'installation, le programme d'installation donne à cette variable le nom de l'ordinateur. Bien que vous puissiez modifier le @@SERVERNAME au moyen de la procédure système sp_addserver et en redémarrant SQL Server, cette méthode n'est généralement pas nécessaire.

@@SERVICENAME

Nom d'un service en cours d'exécution. Actuellement, @@SERVICENAME a pour valeur par défaut @@SERVERNAME.

@@SPID*

Numéro d'identification du processus courant sur le serveur (la colonne spid de la table système sysprocesses).

@@TEXTSIZE*

Valeur actuelle de l'option TEXTSIZE de l'instruction SET, qui indique la longueur maximale, en octets, des données text ou image renvoyées par une instruction SELECT. La limite par défaut est de 4 Ko.

@@TIMETICKS

Nombre de millisecondes par pulsation, qui dépend de l'ordinateur utilisé.. Chaque pulsation du système d'exploitation dure 31,25 millisecondes (1/32e de seconde).

@@TOTAL_ERRORS

Nombre d'erreurs rencontrées par SQL Server en lisant ou en écrivant des données depuis son dernier démarrage.

@@TOTAL_READ

Nombre de lectures de données sur le disque effectuées par SQL Server depuis son dernier démarrage (ce nombre n'inclut que les accès au disque, et non les lectures du cache).

@@TOTAL_WRITE

Nombre d'écritures de données sur le disque effectuées par SQL Server depuis son dernier démarrage.

@@TRANCOUNT*

Nombre de transactions actuellement actives pour l'utilisateur courant.

@@VERSION

Date, numéro de version et type de processeur de la version courante de SQL Server.

32. ANNEXE B : LISTE DES PROCEDURES SYSTEME DE SQL SERVER

sp_addalias
sp_addextendedproc
sp_addgroup
sp_addlogin
sp_addremotelogin
sp_addsegment
sp_addserver
sp_addtype
sp_addumpdevice
sp_adduser
sp_bindefault
sp_bindrule
sp_changedbowner
sp_changegroup
sp_column_privileges
sp_columns
sp_commonkey
sp_databases
sp_datatype_info
sp_dboption
sp_defaultdb
sp_defaultlanguage
sp_depends
sp_diskdefault
sp_dropalias
sp_dropdevice
sp_dropextendedproc
sp_dropgroup
sp_dropkey
sp_droplanguage
sp_droplogin
sp_dropremotelogin
sp_dropsegment
sp_dropserver
sp_helpindex
sp_helpjoins
sp_helpkey
sp_helplanguage
sp_helplog
sp_helpremotelogin
sp_helprotect
sp_helpsegment
sp_helpserver
sp_helpsort
sp_helpsql
sp_helptext
sp_helpuser
sp_lock
sp_logdevice
sp_monitor
sp_password
sp_pkeys
sp_placeobject
sp_primarykey
sp_recompile
sp_remoteoption
sp_rename
sp_renamedb
sp_server_info
sp_serveroption
sp_setlangalias
sp_spaceused
sp_special_columns
sp_sproc_columns
sp_statistics
sp_stored_procedures
sp_table_privileges
sp_tables

sp_droptype
sp_dropuser
sp_extendsegment
sp_fkeys
sp_foreignkey
sp_help
sp_helpdb
sp_helpdevice
sp_helpextendedproc
sp_helpgroup
sp_unbindefault
sp_unbindrule
sp_who

sp_addalias Mappe un utilisateur sur un autre dans une base de données.

sp_addextendedproc Enregistre le nom d'une nouvelle procédure stockée étendue dans master..sysobjects

sp_addgroup Ajoute un groupe à une base de données.

sp_addlogin Ouvre un nouveau compte SQL Server en ajoutant un nom d'accès.

sp_addremotelogin Ajoute un nom d'accès distant à la table système master.

sp_addsegment Définit un segment sur une unité de base de données dans la base courante

sp_addserver Définit un serveur distant ou définit le nom du serveur local.

sp_addtype Crée un type de donnée défini par l'utilisateur.

sp_addumpdevice Ajoute une unité de sauvegarde à SQL Server.

sp_adduser Ajoute un nouvel utilisateur à la base de données courante.

sp_bindefault Associe une valeur par défaut à une colonne ou un type de donnée défini par

sp_bindrule Associe une règle à une colonne ou un type de donnée défini par l'utilisateur

sp_changedbowner	Change le propriétaire d'une base de données.
sp_changegroup	Change le groupe d'un utilisateur.
sp_column_privileges	Fournit des informations sur les privilèges de colonne pour une table donnée dans l'environnement de SGBD courant.
sp_columns	Fournit des informations de colonne pour un objet donné pouvant être interrogé dans l'environnement de SGBD courant. Les colonnes renvoyées appartiennent soit à une table, soit à une vue.
sp_commonkey	Affiche ou modifie les options de configuration.
sp_databases	Liste les bases de données présentes dans l'installation SQL Server ou accessibles via une passerelle de base de données.
sp_datatype_info	Fournit des renseignements sur les types de données autorisés dans le SGBD
sp_dboption	Affiche ou modifie les options de base de données.
sp_defaultdb	Change la base de données par défaut d'un utilisateur.
sp_defaultlanguage	Modifie la langue par défaut d'un utilisateur.
sp_depends	Affiche des informations sur les dépendances entre objets de base de données.
sp_diskdefault	Définit l'état d'une unité de base de données comme DEFAULTON ou DEFAULTOFF.
sp_dropalias	Supprime un nom d'emprunt.
sp_dropdevice	Supprime une unité de base de données ou de sauvegarde.
sp_dropextendedproc	Supprime une procédure stockée étendue de la table master..sysobjects.
sp_dropgroup	Supprime un groupe dans une base de données.
sp_dropkey	Supprime une clé définie antérieurement à l'aide de sp_primarykey sp_foreignkey ou sp_commonkey.

sp_droplanguage	Supprime une langue de remplacement du serveur
sp_droplogin	Supprime un compte SQL Server.
sp_dropremotelogin	Supprime un compte SQL Server distant.
sp_dropsegment	Supprime un segment d'une base de données ou dissocie un segment d'une unité de base de données.
sp_dropserver	Supprime un serveur dans la liste des serveurs connus.
sp_droptype	Supprime un type de donnée défini par l'utilisateur.
sp_dropuser	Supprime un compte d'utilisateur de la base de données courante.
sp_extendsegment	Etend la portée d'un segment à une autre unité de base de données.
sp_fkeys	Fournit des informations logiques de clé étrangère pour l'environnement de SGBD courant
sp_foreignkey	Définit une clé étrangère dans une table ou une vue.
sp_help	Affiche des renseignements sur un objet de base de données ou sur un type de donnée fourni par SQL Server ou défini par l'utilisateur.
sp_helpdb	Fournit des renseignements sur une base de données ou sur toute les bases.
sp_helpdevice	Fournit des renseignements sur les unités de base de données et les unités de sauvegarde de SQL Server.
sp_helpextendedproc	Fournit des informations sur une procédure stockée étendue ou sur toutes les procédures stockées étendues de master..sysobjects.
sp_helpgroup	Fournit des informations sur un groupe ou sur tous les groupes de la base de données courante.
sp_helpindex	Fournit des renseignements sur les index d'une table.
sp_helpjoins	Liste les colonnes susceptibles d'être jointes dans deux tables ou vues

sp_helpkey	Fournit des renseignements sur les clés principales, étrangères et communes.
sp_helplanguage	Fournit des renseignements sur une langue de remplacement particulière ou sur toutes les langues.
sp_helplog	Fournit le nom de l'unité qui contient la première page du journal.
sp_helpremotelogin	Fournit des renseignements sur les noms de connexion d'un serveur distant particulier ou de tous les serveurs distants.
sp_helprotect	Décrit les permissions pour un objet de base de données et, facultativement, les permissions de l'utilisateur spécifié pour cet objet.
sp_helpsegment	Fournit des renseignements sur un segment particulier ou sur tous les segments dans la base de données courante.
sp_helpserver	Fournit des renseignements sur un serveurdistant particulier ou sur tous les serveurs distants.
sp_helpsort	Affiche l'ordre de tri et le jeu de caractères par défaut de SQL Server.
sp_helpsql	Fournit la syntaxe d'instructions Transact-SQL ou de procédures système, ou des informations sur d'autres sujets spécifiques.
sp_helptext	Affiche le texte d'une procédure stockée, un déclencheur, une vue, une valeur par défaut ou une règle.
sp_helpuser	Fournit des renseignements sur des utilisateurs d'une base de données.
sp_lock	Fournit des renseignements sur les verrous.
sp_logdevice	Place le journal des transactions sur une unité de base de données séparée.
sp_monitor	Fournit des statistiques sur SQL Server.
sp_password	Ajoute ou modifie un mot de passe dans un compte SQL Server.
sp_pkeys	Renvoie des informations de clé principale pour une table donnée dans l'environnement de SGBD courant
sp_placeobject	Place sur un segment particulier les futures allocations d'espace pour une table ou un index.

sp_primarykey	Définit une clé principale pour une table ou une vue.
sp_recompile	Fait recompiler chaque procédure stockée et déclencheur utilisant la table spécifiée lors de son exécution suivante.
sp_remoteoption	Affiche ou modifie des options de connexiondistantes.
sp_rename	Modifie le nom d'un objet de l'utilisateur dans la base de données courante.
sp_renamedb	Modifie le nom d'une base de données.
sp_server_info	Renvoie une liste de noms d'attribut et les valeurs correspondantes pour SQL Server ou la passerelle de base de données et/ou la source de données sous-jacente
sp_serveroption	Affiche ou modifie des options de serveur.
sp_setlangalias	Affecte ou modifie le nom alias d'une langue de remplacement.
sp_spaceused	Affiche pour l'objet de base de données spécifié le nombre de lignes, l'espace disque réservé et l'espace disque utilisé.
sp_special_columns	Renvoie l'ensemble optimal de colonnes qui identifient une ligne de façon unique dans une table et de colonnes qui sont automatiquement modifiées quand une valeur quelconque de la ligne est modifiée par une transaction
sp_sproc_columns	Renvoie des informations de colonne pour une procédure stockée donnée dans l'environnement de SGBD courant
sp_statistics	Renvoie une liste de tous les index d'une table donnée, déterminée par les paramètres qualificateur_de_table, propriétaire_de_table et nom_de_table
sp_stored_procedures	Renvoie la liste des procédures stockées dans l'environnement deSGBD courant
sp_table_privileges	Renvoie des informations sur les privilèges de table pour une table donnée dans l'environnement de SGBD courant
sp_tables	Renvoie la liste des objets pouvant être interrogés dans l'environnement de SGBD courant
sp_unbindefault	Dissocie une valeur par défaut d'une colonne ou d'un type de donnée défini par l'utilisateur.

sp_unbindrule Dissocie une règle d'une colonne ou d'un type de donnée défini par l'utilisateur.

sp_who Fournit des renseignements sur les utilisateurs et les processus

--- FIN DU DOCUMENT ---