Secteur Tertiaire Informatique
Filière étude - développement

Activité « Développer la persistance des données »

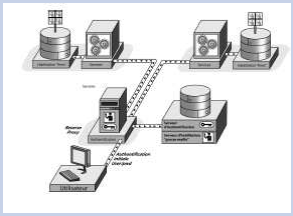
Transact SQL : Le Langage DML

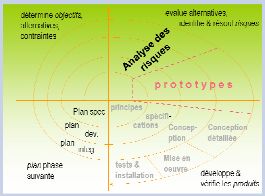
Accueil

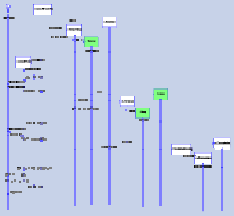
Apprentissage

Période en entreprise

Evaluation







Code barre

SOMMAIRE

I LES ELEMENTS COMPLEMENTAIRES DU LANGAGE

Les principaux éléments complémentaires de Transact-SQL sont les commentaires, les variables, les fonctions, et les instructions de contrôle du déroulement.

I.1 LES COMMENTAIRES

Les **commentaires en ligne** sont situés après deux tirets -- ;

Un commentaire en ligne peut être placé sur la même ligne qu'une instruction, à la suite de l'instruction ou bien en début de ligne, toute la ligne constituant alors le commentaire. Si le commentaire nécessite plusieurs lignes, il faut répéter les tirets sur chaque ligne.

Il est également possible de créer un **bloc de commentaires** en plaçant **/*** en début et ***/** fin de bloc

I.2 LES VARIABLES LOCALES

On définit une variable locale à l'aide d'une instruction **DECLARE**, on lui affecte une valeur initiale à l'aide de l'instruction **SET** et on l'utilise dans le cadre de l'instruction, du lot ou de la procédure dans laquelle elle a été déclarée.

Le nom d'une variable locale commence toujours par @.

Syntaxe

```
DECLARE  @nom_variable type_de_données
```

Exemple : Liste de tous les employés dont le nom commence par une chaîne de caractères, donnée (B).

```
DECLARE   @vrech varchar(24)
SET @vrech ='B%'
SELECT noemp, prenom, nom, dept FROM Employés
        WHERE nom like @vRech
```

Exemple : renvoi du numéro d'employé le plus élevé.

```
DECLARE  @vempId int
SELECT @vempId = max(noemp) FROM Employés
```

Si l'instruction SELECT renvoie plusieurs lignes, la valeur affectée à la variable est celle correspondant à la dernière ligne renvoyée.

```
DECLARE  @vempId int
SELECT @vempId = noemp FROM Employés
```

I.3 LES VARIABLES SYSTEME

TRANSACT-SQL propose de nombreuses variables qui renvoient des informations ; certaines d'entre elles demandant des paramètres en entrée.

Elles se distinguent des variables utilisateur par le double @ :

La syntaxe d'utilisation de la plupart des fonctions est :

@@nom_fonction

Quelques exemples …

La variable @@**VERSION** renvoie des informations sur les versions de SQL SERVER et de système d'exploitation utilisées.

```
SELECT @@VERSION
```

La variable @@**TRANCOUNT** renvoie le nombre de transactions ouvertes.

La variable @@**ROWCOUNT** renvoie une valeur représentant le nombre de lignes affectées par la dernière requête effectuée.

La variable @@**ERROR** contient le numéro de l'erreur de la dernière instruction TRANSACT-SQL exécutée. Elle est effacée et réinitialisée chaque fois qu'une instruction est exécutée ; Si l'instruction s'est exécutée avec succès, @@ERROR renvoie la valeur 0 ; On peut utiliser la fonction @@ERROR pour détecter un numéro d'erreur particulier ou sortir d'une procédure stockée de manière conditionnelle.

I.4 LES ELEMENTS DE CONTROLE DU DEROULEMENT

TRANSACT-SQL contient plusieurs éléments qui permettent d'influer sur le déroulement des scripts, des lots et des procédures stockées.

a) La fonction CASE

La fonction **CASE** évalue une liste de conditions et renvoie la valeur de l'expression de résultat correspondant à la condition sélectionnée.

Syntaxe

```
CASE expression_en_entrée
    WHEN expression_cas_particulier THEN expression_resultat
    […n]
    [ELSE expression_resultat_dans_les_autres_cas]
END
```

Exemple : Affichage du salaire sous forme d'un commentaire texte basé sur une fourchette de salaire.

```
SELECT 'Catégorie de Salaire ' =
    CASE
        WHEN salaire IS NULL THEN 'Non divulgué !'
        WHEN salaire < 1500 THEN 'Agent de maîtrise'
        WHEN salaire >= 1500 and salaire < 2000 THEN 'Cadre'
    ELSE 'PDG!'
    END,
    nom
    FROM Employés ORDER BY salaire
```

Voici le jeu de résultats :

```
Catégorie de Salaire            Nom
----------------------------------------------------------------
Non divulgué                    CAZENEUVE
Agent de maîtrise               BALZAC
Cadre                           MARTIN
Cadre                           DURAND
Cadre                           ZOLA
Cadre                           VIALA
Cadre                           DELMAS
PDG                             BOULIN
```

b) La structure BEGIN … END

C'est une structure de bloc permettant de délimiter une série d'instructions (utilisé avec IF et WHILE).

c) L'instruction RETURN

Elle permet d'afficher un message.

d) L'instruction PRINT

Elle permet de sortir d'une procédure en renvoyant éventuellement une variable entière.

e) La fonction IF

C'est la structure alternative qui permet des tester une condition et d'exécuter une instruction si le test est vrai.

Syntaxe

IF condition
    {Instruction | bloc}
[**ELSE**]
    {Instruction | bloc}

Exemple: 1 Si la moyenne des salaires est inférieure à 1500, l'augmentation est nécessaire …

```
IF (Select AVG(salaire) from EMPLOYES ) < 1500
    BEGIN
        UPDATE …. ....
        PRINT 'Augmentation effectuée'
    END
```

Exemple 2 : Vérification que le département 'E21' comporte au moins un salarié avant de le supprimer

```
IF EXISTS (SELECT Nodept FROM Depart WHERE Nodept ='E21')
        PRINT '*** impossible de supprimer le client ***'
ELSE
        BEGIN
            DELETE Depart WHERE Nodept ='E21'
            PRINT '*** Client supprimé **'
        END
```

f) La fonction WHILE

C'est la structure répétitive qui permet d'exécuter une série d'instructions tant qu'une condition est vraie.

L'exécution des instructions de la boucle peut être contrôlée par les instructions **BREAK** et **CONTINUE**.

L'instruction BREAK permet de sortir de la boucle.

CONTINUE permet de repartir à la première instruction de la boucle.

Syntaxe

```
WHILE condition
        {Instruction | bloc}
```

Exemple 1 : Utilisation du While

```
DECLARE @Compteur int
SET @Compteur =1
WHILE @Compteur <= 10
        BEGIN
                INSERT.....
                SET @compteur = @compteur + 1
        END
```

Exemple 2 : Utilisation du While

```
DECLARE @Compteur int
SET @Compteur =1
WHILE @Compteur <= 10
        BEGIN
                INSERT  .....
                IF <cond>
                        BEGIN
                                ... instructions ...
                                BREAK
                        END
                SET @compteur = @compteur + 1

END
```

g) La clause OUTPUT

Output [listecolonne] INTO @variable
La variable doit être de type table et déclarée avant son utilisation dans la clause OUTPUT. Elle stockera l'ensemble des colonnes citées, qui peuvent provenir directement de l'instruction DML. En utilisant les préfixes DELETED et INSERTED, on stockera les données touchées par la suppression/modification et modification/insertion.

Exemple : Utilisation de Output

```
DECLARE @Tajout table(num int )

INSERT INTO EMPLOYES (NOEMP, NOM, PRENOM, DEPT)
    VALUES (00140,'REEVES','HUBERT','A00')
    OUTPUT INSERTED.NOEMP INTO @Tajout
```

On pourra ensuite visualiser les données ajoutées en affichant le contenu de la table Tajout.

I.5 LES MESSAGES D'ERREUR

Pour chaque erreur, SQL Server produit un message d'erreur ; La plupart de ces messages sont définis dans SQL Server, mais il est possible de définir ses propres messages grâce à la procédure stockée système **sp_addmessage**.
Tous les messages stockés à l'aide de **sp_addmessage** peuvent être affichés grâce à l'affichage catalogue **sys.messages .**

```
SELECT * FROM SYS.MESSAGES
```

Quelque soit leur origine, les messages d'erreur possèdent tous la même structure :

- un numéro qui identifie le message
- le texte du message qu'on peut personnaliser grâce à des variables
- une sévérité qui donne le niveau de gravité de l'erreur

Sévérité	
< 9	Message d'information non bloquant
10	Message d'information : erreur de saisie utilisateur
Entre 11 et 16	L'erreur peut être résolue par l'utilisateur
Entre 17 et 19	Erreurs logicielles
Entre 20 et 25	Problèmes système

(voir l'aide en ligne pour plus de détails)

- un numéro d'état qui associe à l'erreur son contexte

Syntaxe

```
sp_addmessage [ @msgnum = ] num_msg ,
        [ @severity = ] gravité ,
        [ @msgtext = ] 'msg'
        [ , [ @lang = ] 'langage' ]
        [ , [ @with_log = { TRUE | FALSE } ]
        [ , [ @replace = ] 'replace' ]
```

@msgnum

Spécifie le numéro du message supérieur à 50001 pour les messages utilisateur

@severity

Spécifie la gravité du message compris entre 1 et 25

@msgtext

Texte du message

%d permet d'insérer une variable numérique, une variable chaîne.

@lang

Spécifie la langue du message ;

le message doit être défini en anglais en premier lieu (us_english), puis il peut être créé en français (French).

@with_log

Spécifie si le message sera consigné ou non dans le journal Windows.

@replace

Remplace le message existant.

Les messages d'erreur peuvent être modifiés avec **sp_altermessage**, et supprimés avec **sp_dropmessage**.

Exemple 1 : Ajout d'un message simple mentionnant qu'un employé n'existe pas

```
EXECUTE sp_addmessage 50001, 16, 'Le code employé est inexistant', 'us-english'
EXECUTE sp_addmessage 50001, 16, 'Le code employé est inexistant'
```

Le message est d'abord ajouté en langue anglaise, puis en français.

Exemple 2 : Ajout d'un message simple mentionnant que l'employé {code} n'existe pas dans le service {nom}

```
EXECUTE sp_addmessage 50002, 16,
'Le code employé %d est inexistant dans le service %s', 'us_english'

EXECUTE sp_addmessage 50002, 16,
'Le code employé %d est inexistant dans le service %s'
```

II MODE D'EXECUTION DES INSTRUCTIONS

Il existe plusieurs manières d'exécuter les instructions Transact-SQL : elles peuvent être exécutées individuellement ou en lots, être enregistrées dans des scripts et exécutées sous forme de procédures stockées ou de déclencheurs.

II.1 UTILISATION DES LOTS

Un lot (batch) est un ensemble d'instructions Transact SQL envoyées ensemble à SQL Server et exécutées collectivement. Un lot peut être soumis de façon interactive ou dans le cadre d'un script.

Le séparateur de lot GO n'est pas une instruction Transact-SQL ; il n'est reconnu que par l'Analyseur de requête et signale qu'il doit soumettre la ou les requêtes précédentes.

SQL Server optimise, compile et exécute ensemble les instructions d'un même lot ; la portée des variables est limitée à un seul lot.

Les instructions exécutées au sein d'un même lot doivent respecter un certain nombre de règles :

- Il n'est pas possible de combiner les instructions CREATE DEFAULT, CREATE VIEW, CREATE TRIGGER, CREATE PROCEDURE avec d'autres instructions dans un même lot
- Si une table est modifiée, les nouvelles colonnes ne pourront être utilisées que dans le lot suivant

Exemple de lot dont l'exécution échouera …

```
CREATE DATABASE
CREATE TABLE
CREATE TRIGGER
CREATE TRIGGER
GO
```

Exemple de lot dont l'exécution réussira …

```
CREATE DATABASE
GO
CREATE TABLE
GO
CREATE TRIGGER
GO
CREATE TRIGGER
GO
```

II.2 UTILISATION DES SCRIPTS

L'un des modes d'exécution des instructions les plus utilisés passe par la création de scripts. Un script est une ou plusieurs instructions TRANSACT SQL écrites depuis l'Analyseur de requêtes (ou le Bloc Note) enregistrées dans un fichier dont l'extension est .SQL.

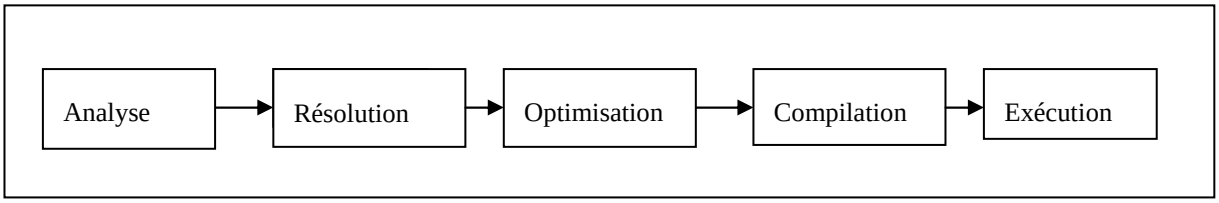
Le script est intéressant pour la régénération d'une base de données.

II.3 MODE DE TRAITEMENT DES REQUETES

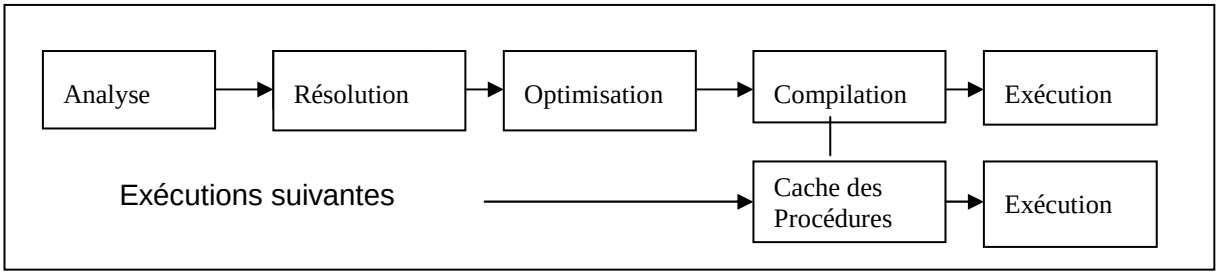
Chaque requête(instruction unique, lot d'instructions ou demande d'exécution de procédure stockée) est traitée en un certain nombre d'étapes avant de passer à son exécution proprement dite. Ces requêtes produisent un **plan de requête** qui sera ensuite exécuté.

Afin d'améliorer les performances, SQL Server enregistre en mémoire les plans de requête compilés pour une réutilisation ultérieure.

Requête non placée en mémoire cache



Requête placée en mémoire cache



Etapes	Description
Analyse	Vérification de la syntaxe de la requête
Résolution	Contrôle de l'existence des objets référencés et des autorisations sur ces objets
Optimisation	Détermination des index et des stratégies de jointure à utiliser
Compilation	Traduction de la requête en une forme exécutable
Exécution	Exécution de la requête compilée

Les plans de requête sont supprimés du cache lorsque un besoin de mémoire supplémentaire se fait sentir : L'ordre de suppression des plans dépend de sa dernière utilisation et de son coût calculé en fonction des ressources utilisées pour le compiler.

Exemple :

Use Northwind

Select * From Products Where unitprice = $12.5

Select * From Products Where unitprice = 12.5

Select * From Products Where unitprice = $12.5

Des plans de requête distincts seront créés pour la 1ere et la 2eme instruction. La 3eme instruction utilisera le plan de requête de la 1ere instruction ($12.5 est passé en donnée monétaire alors que 12.5 est passé en données à virgule flottante).

II.4 ELABORATION DYNAMIQUE DES INSTRUCTIONS

Les instructions élaborées dynamiquement sont utiles si la valeur d'une variable doit être prise en considération au moment de l'exécution, par exemple pour appliquer la même action à toute une série d'objets de la base de données.
On utilisera soit la procédure stockée système **sp_executesql** soit l'instruction **EXECUTE**.

Exemple 1 : Utilisation de l'instruction EXECUTE

DECLARE @dbname varchar(30), @tablename varchar(30)

SET @dbname = 'Northwind'

SET @tablename = 'Products'

EXEC ('USE' + @dbname +

' SELECT * FROM ' + @tablename)

L'instruction fournie en argument de l'instruction EXECUTE peut être un littéral de chaîne,une variable locale de type chaîne ou la concaténation de plusieurs de ces éléments : toutes les variables de chaîne EXECUTE doivent être d'un type de données caractère(char, varchar, nchar ou nvarchar) ; toutes mes données numériques doivent être converties.

Exemple 2 : Utilisation de la procédure stockée sp_executesql

DECLARE @dbname varchar(30), @tablename varchar(30)

DECLARE @sqlstrinf nvarchar(200)

SET @dbname = 'Northwind'

SET @tablename = ‘Products’

SET @sqlstring =N’USE ‘ + @dbname +

N’ SELECT * FROM ‘ + @tablename

EXEC sp_executeql @sqlstring

L’instruction fournie en argument de la procédure **sp_executesql** doit être un littéral de chaîne Unicode ou une variable pouvant être convertie en texte Unicode(le N majuscule précédant les chaînes de caractères sert à indiquer un littéral de chaîne Unicode)

L’instruction exécutée par SQL Server dans les 2 exemples est

USE Northwind

SELECT * FROM Products

Etablissement référent
Direction de l'ingénierie Neuilly

Equipe de conception
Groupe d'étude de la filière étude - développement

Remerciements :

Date de mise à jour 17/02/2009
afpa © Date de dépôt légal février 09

afpa / Direction de l'Ingénierie13 place du Générale de Gaulle / 93108 Montreuil Cedex
association nationale pour la formation professionnelle des adultes
Ministère des Affaires sociales du Travail et de la Solidarité