

Secteur Tertiaire Informatique
Filière « Etude et développement »

Séquence « Développer des composants d'accès
aux données »

SQL Server et Transact SQL
Le langage DML partie3 - Programmation

Apprentissage

Mise en pratique

Evaluation

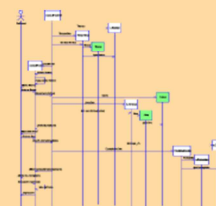
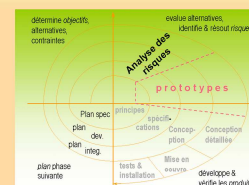
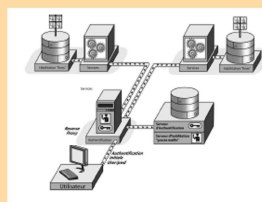


TABLE DES MATIERES

Table des matières	3
1. LES ELEMENTS COMPLEMENTAIRES DU LANGAGE	5
1.1 Les commentaires.....	5
1.2 Les variables locales.....	5
1.3 Les variables système.....	6
1.4 Les éléments de contrôle du déroulement.....	6
1.4.1 La structure BEGIN ... END.....	6
1.4.2 L'instruction PRINT.....	6
1.4.3 La structure IF	7
1.4.4 La structure CASE	8
1.4.5 La structure WHILE	9
1.4.6 L'instruction RETURN.....	10
1.4.7 La clause OUTPUT.....	10
1.5 Les messages d'erreur.....	11
2. MODE D'EXECUTION DES INSTRUCTIONS	13
2.1 Utilisation des lots	13
2.2 Utilisation des scripts	14
2.3 Mode de traitement des requêtes.....	14
2.4 Elaboration dynamique des instructions	15

Objectifs

Ce document a pour but de vous faire découvrir les instructions SQL complémentaires intégrées dans le langage Transact-SQL et qui sont nécessaires pour développer fonctions, procédures stockées et triggers dans SQL Server.

Les modalités de mise en œuvre n'est pas détaillée ici et fera l'objet de ressources complémentaires spécifiques à chacun des cas de programmation du SGBD.

Pré requis

Maîtriser la logique des requêtes SQL de base et être initié à la logique de programmation procédurale.

Outils de développement

SQL Server (2012) et son IDE SQL Server Management Studio.

Méthodologie

Ce document est conçu comme un aide-mémoire pour les principales techniques à utiliser dans la programmation du SGBD SQL Server ; il complète sans la remplacer la documentation de référence.

Mode d'emploi

Symboles utilisés :



Renvoie à des supports de cours, des livres ou à la documentation en ligne constructeur.



Propose des exercices ou des mises en situation pratiques.



Point important qui mérite d'être souligné !

Ressources

Lectures conseillées

Au fur et à mesure des besoins, la documentation de référence accessible directement par l'aide intégrée et indexée dans SQL Server Management Studio (touche F1).

1. LES ELEMENTS COMPLEMENTAIRES DU LANGAGE

Les principaux éléments complémentaires de Transact-SQL pour programmer le SGBD sont les **commentaires**, les **variables**, les **instructions de contrôle** du déroulement et la **gestion des erreurs** d'exécution.

1.1 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.

1.2 LES VARIABLES LOCALES

On définit une **variable locale** à l'aide d'une instruction **DECLARE**, on lui affecte une valeur à l'aide de l'instruction **SET** (ou même **SELECT**) 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 (ici « 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
```

1.3 LES VARIABLES SYSTEME

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


Elles se distinguent des variables utilisateur par le **double @** au début de leur nom.


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 variable **@@ERROR** pour détecter un numéro d'erreur particulier ou sortir d'une procédure stockée de manière conditionnelle.

1.4 LES ELEMENTS DE CONTROLE DU DEROULEMENT

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

1.4.1 La structure BEGIN ... END

Un ensemble d'instructions Transact-SQL encadrées par **BEGIN** et **END** forme une **structure de bloc** permettant de délimiter une série d'instructions formant un tout ; cette structure est utilisée essentiellement dans la structure conditionnelle **IF** et la répétitive **WHILE**, ainsi que pour délimiter le contenu d'une procédure stockée ou d'un déclencheur.

1.4.2 L'instruction PRINT

Elle permet d'afficher un message, ou du moins de l'insérer dans le flot du message à retourner.

1.4.3 La structure IF

C'est la **structure alternative** qui permet de tester une condition et d'exécuter une instruction (ou un bloc) uniquement si le test est vrai ; en option, une instruction (ou un bloc) peut être précisée dans une clause **ELSE** afin d'exécuter un autre traitement *si la condition est évaluée fausse*.

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
```

1.4.4 La structure CASE

La structure **CASE** évalue une liste de conditions et renvoie la valeur de l'expression de résultat correspondant à la condition vérifiée ; cette structure peut permettre d'écrire, dans une syntaxe légère, des **tests imbriqués, exclusifs les uns des autres**. A noter que son emploi reste limité essentiellement à l'intérieur des instructions LMD **SELECT**.

Syntaxe :

```
CASE expression_en_entrée
    WHEN valeur_cas_particulier1 THEN expression_resultat1
    WHEN valeur_cas_particulier2 THEN expression_resultat2
    [...]
    [ELSE expression_resultat_dans_les_autres_cas]
END - fin du case
```

ou bien

CASE

```
    WHEN expression_cas_particulier1 THEN expression_resultat1
    WHEN expression_cas_particulier2 THEN expression_resultat2
    [...]
    [ELSE expression_resultat_dans_les_autres_cas]
END - fin du case
```

Dans la première syntaxe, **une expression est évaluée** dès le début et **ses différentes valeurs possibles sont énumérées** derrière les mots-clés **WHEN**.

Dans la deuxième syntaxe, **une liste d'expressions de comparaisons est énumérée** dans les clauses **WHEN**, ce qui permet de tester par rapport à des fourchettes de valeurs.

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 un exemple de 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

1.4.5 La structure 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 (sauvagement) de la boucle, **CONTINUE** permet de repartir (sauvagement) à la première instruction de la boucle ; ces 2 instructions ne devraient pas être nécessaires en bonne programmation structurée...

Syntaxe :

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

Exemple 1 :

Utilisation du While pour répéter systématiquement :

```
DECLARE @Compteur int  
SET @Compteur =1 -- initialisation  
WHILE @Compteur <= 10 - condition de bouclage  
    BEGIN  
        INSERT.....  
        SET @compteur = @compteur + 1 -- progression  
    END - fin de boucle
```

Exemple 2 :

Utilisation du `while` pour répéter sous condition :

```
DECLARE @Compteur int
SET @Compteur =1
WHILE @Compteur <= 10
    BEGIN
        INSERT .....
        IF <cond>
            BEGIN
                ... instructions ...
                BREAK -- arrêt sauvage de la boucle; beuh!
            END - fin du IF
        SET @compteur = @compteur + 1
    END - fin du while
```

1.4.6 L'instruction RETURN

Elle permet de sortir d'un programme Transct-SQL en renvoyant éventuellement une valeur ou une variable entière.

1.4.7 La clause OUTPUT

Cette clause permet de retourner une valeur à l'application/l'utilisateur ou encore de sauvegarder/conserver trace des informations sur toutes les lignes de la table affectée par une opération de mise à jour.

Syntaxe :

```
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 INSERT, DELETE ou UPDATE`. En utilisant les préfixes `DELETED` et `INSERTED`, on stockera les données touchées par la suppression/modification et modification/insertion au sein d'un trigger.

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
```

SQL Server et Transact SQL - Programmation

Afpa © 2015 – Section Tertiaire Informatique – Filière « Etude et développement »

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

1.5 LES MESSAGES D'ERREUR

Pour chaque erreur d'exécution, 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 de la table système **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
- un **numéro d'état** qui associe à l'erreur son contexte

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)

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**

SQL Server et Transact SQL - Programmation

Afpa © 2015 – Section Tertiaire Informatique – Filière « Etude et développement »

@msgtext

Texte du message

%d permet d'insérer une variable numérique, %s, 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 (consultable grâce à l'Observateur d'événements accessible au Panneau de Configuration, Gestion de l'ordinateur).

@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'
```

Tout comme SQL Server effectue une « levée d'erreur » (notification) en cas d'erreur détectée automatiquement, un programme Transact-SQL peut provoquer une levée d'erreur grâce à l'instruction **RAISERROR** (détaillée dans l'aide en ligne SQL Server) :

Exemples :

```
RAISERROR ('Produit inexistant', 16, 1)
```

```
RAISERROR (21508, 16, 1)
```

```
RAISERROR (50002, 16, 1, 123, 'Informatique')
```

Récupération messages
standard de SQL Server
(numéro inférieur à 50000)

Autres syntaxes compatibles avec les versions précédentes de SQL Server :

```
RAISERROR 51000 'Produit inexistant'
```

ou

```
SELECT @msg = 'Produit inexistant'
```

```
RAISERROR 51000 @msg
```

Messages personnalisés :
numéro supérieur à 50000


L'identifiant du message est accessible dans la variable système @@ERROR.

2. 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.

2.1 UTILISATION DES LOTS

SQL Server est conçu pour pouvoir répondre à des sollicitations multiples simultanément (c'est un logiciel serveur) ; l'exécution des commandes reçues est donc organisée par un planificateur et, en production réelle, SQL Serveur peut très bien « entremêler » les commandes de plusieurs clients.

 **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, mot-clé GO, n'est pas une instruction Transact-SQL ; il n'est reconnu que par l'Analyseur de requête et signale qu'il doit soumettre et exécuter la ou les requêtes précédentes avant de passer à la suite.

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
```

2.2 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 par défaut).

Un script peut contenir un ou plusieurs lots. **Le script est à la base des procédures stockées et des triggers.**

Le script est intéressant pour la régénération d'une base de données ; les bons outils de modélisations aboutissent à la génération des scripts). SQL Server peut aussi générer lui-même le script correspondant à la création d'une base existante à des fins de documentation ou de re-crédation ultérieure.

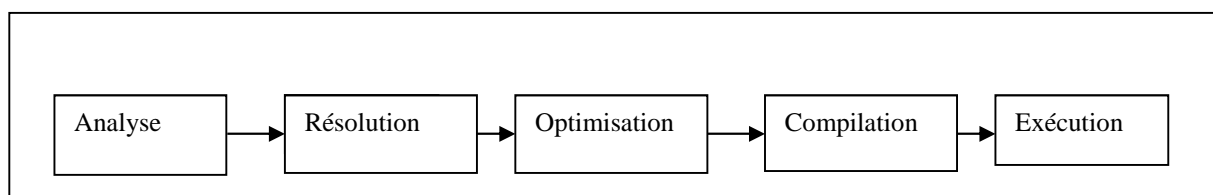
2.3 MODE DE TRAITEMENT DES REQUETES

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.

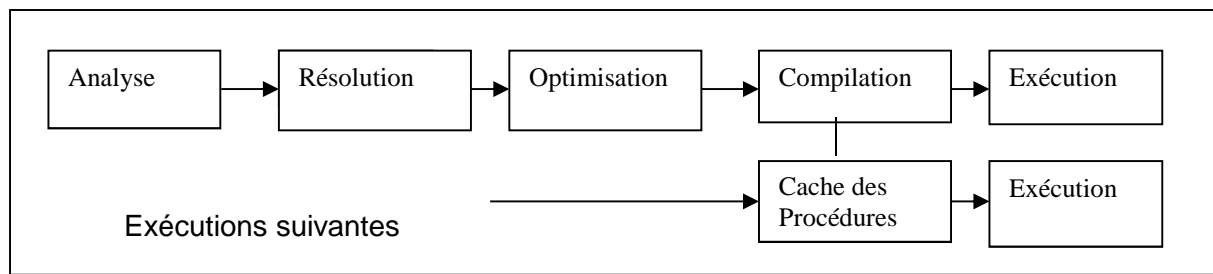
Ces traitements produisent un **plan de requête** qui sera ensuite exécuté (SQL Server peut fournir au « DBA », l'administrateur des bases de données, des informations sur les plans de requêtes utiles à leur optimisation).

Afin d'améliorer les performances, SQL Server enregistre en mémoire-cache 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 automatiquement du cache lorsqu'un besoin de mémoire supplémentaire se fait sentir. L'ordre de suppression des plans dépend de leur dernière utilisation et de leur coût calculé en fonction des ressources utilisées pour les 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 1ère et la 2ème instruction. La 3ème instruction utilisera le plan de requête de la 1ère instruction (\$12.5 est passé en donnée monétaire alors que 12.5 est passé en données à virgule flottante).

2.4 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** ou son abrégé **EXEC**.

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)
```

NB : 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 les données numériques doivent donc 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_executesql @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
```


CREDITS

ŒUVRE COLLECTIVE DE l'AFPA

Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)

E. Cattaneo - formateur

B. Hézard - Formateur

Ch. Perrachon – Ingénieure de formation

Date de mise à jour : 7/10/15

Reproduction interdite

Article L 122-4 du code de la propriété intellectuelle.

« Toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droits ou ayants cause est illicite. Il en est de même pour la traduction, l'adaptation ou la reproduction par un art ou un procédé quelconque. »