SQL-LMD Programmation SGBDR

association nationale
pour la formation professionnelle
des adultes

Vincent BOST

afpa
La formation professionnelle

Au sommaire

Les variables

Elles sont de deux types :

- Locale, déclarée par l'utilisateur
- Système (globale)

Déclaration d'une variable locale

- DECLARE @mavariable type [longueur]
- Le préfixe @ définit une variable utilisateur locale

Utilisation d'une variable système

- Une variable système a pour préfixe @@
- Exemple : @@rowcount

Les variables : Affectation

Affectées par l'opérateur SET

- SET @mavariable = 12

Affectées par une opération SELECT

- SELECT @variable=colonne1, @variable2=colonne2 FROM WHERE tableID = @ID

Le résultat assigné doit être une valeur unique (scalaire)

Si besoin de tableaux, utilisation de tables temporaires

Les variables système

Derrière les variables systèmes se cachent souvent des fonctions.

Pour accéder à la liste, tapez @@ dans l'index de l'aide

Quelques exemples :

- @@rowcount : nombre de lignes affectées par la dernière instruction. Permet de déterminer le succès ou l'échec. Associée à @nocount
- @@error : Retourne le numéro d'erreur pour la dernière instruction

Les principaux opérateurs

Arithmétiques

- Multiplication : *
- Division : /
- Modulo : %
- Addition : +
- Soustraction : -

Logiques

- NOT
- AND
- OR

Comparaison

- Egal à : =
- Supérieur : >
- Inférieur : <
- Différent : <> ou !=

Concaténation

- Concaténation : +

Les fonctions système : exemples

Date et heure

- Heure Syst. : GetDate()
- Ajout : DateAdd()
- Différence : DateDiff()
- Portion : DatePart()
- …

Chaînes

- Trim()
- Substring(), PatIndex()
- Len()

Mathématiques

- Arrondi : ROUND()
- Racine Carrée : SQUARE()

Système

- Conversion CAST ou CONVERT
- IsNull, Coalesce()

Les contrôles de flux

Niveau Instruction

- IF … ELSE

Bloc

- BEGIN …END
- WHILE…
  CONTINUE
  BREAK
  END

```
IF user_name() <> 'dbo'
   BEGIN
         RAISERROR('Rôle sysadmin requis',10,1)
         RETURN -1
   END
ELSE
    BEGIN
         DBCC CHECKDB(CasContacts)
         RETURN 0
    END
```

Création dynamique d'instructions

Utilisation de l'instruction Execute avec des chaînes de texte et des variables

Permet d'affecter dynamiquement une valeur à une variable lors de l'exécution

```
Declare @nomTable sysname, @ChaineSQL varchar(550)
SET @nomTable = 'Products'
SET @chaineSQL = ' SELECT * FROM ' + @nomTable
EXECUTE(@chaineSQL)
```

Utilisation des blocs Try - Catch

```
BEGIN TRY
    Lot d'instructions
END TRY
BEGIN CATCH
    Traitement de l'exception
END CATCH
```

L'exception est associée à la notion de message d'erreur.

Sont traitées par Try Catch les erreurs avec gravité >= 20

Les messages d'erreurs

L'instruction Raiserror permet d'émettre un message :

- Défini dynamiquement lors de l'exécution
- Extrait de la table des messages sysmessages
- Consigné ou non dans le journal des applications

Les messages peuvent être définis et stockés dans la table des messages afin apporter plus de rigueur à nos développements

Possibilité d'utiliser aussi l'instruction Throw

Exemple d’ajout de messages

Les messages doivent être ajoutés d’abord dans la version linguistique de base du système puis dans notre culture

- EXEC sp_addmessage @msgnum = 60001, @severity = 16,
  @msgtext = N'The user named %s already exists.',
  @lang = 'us_english'
- EXEC sp_addmessage @msgnum = 60001, @severity = 16,
  @msgtext = N'L''utilisateur nommé %1! existe déjà!',
  @lang = 'French'

Procédures stockées : introduction

Constituées d'un ensemble d'instructions Transact SQL elles sont :

- Compilées et stockées sur le serveur
- Constituent des méthodes encapsulées
- Autorisent les variables, l'exécution conditionnelle, …

De 3 types

- Système : résident dans la base de données Master et permettent d'effectuer des tâches d'administration. Préfixe sp_
- Locales : créées dans les DB utilisateurs
- Etendues : implémentées comme des dll, elles sont exécutées en dehors de l'environnement SQL. Préfixe xp_

Procédures stockées : avantages

Partage de la logique d'application

- Permettent d'encapsuler une fonctionnalité d'entreprise.
- Factorisation et centralisation du code

Apporte sécurité et cohérence

- Les applications clientes doivent passer par ces procédures stockées pour effectuer les transactions demandées
- Permet de s'assurer que les développeurs d'application respectent les règles définis par l'administrateur de la base de données (DBA)

Procédures stockées : avantages

Améliorent les performances

- Peuvent implémenter de nombreuses tâches en interne.
  La logique conditionnelle permet de limiter le nombre d'instructions à exécuter.

Elles contribuent à réduire le trafic sur le réseau.

- Au lieu d'envoyer de nombreuses instructions depuis l'application cliente, on utilisera une procédure stockée qui effectuera une opération complexe en son sein.
- Exemple : Suppression d'un ensemble de lignes, création d'un jeu de résultats complexe

Procédures stockées : Règles de création

Une tâche par procédure

- Comme pour les tâches de découpage fonctions / procédures dans un langage évolué, il est préférable de spécialiser les procédures à l'accomplissement d'une seul tâche.
- Mise au point et maintenance facilitées

Réaliser un test unitaire

- Les procédures sont testées avant d'être mises à disposition des développeurs d'applications

Respecter les conventions de nommage

Procédures stockées : définition du code

```
CREATE PROCEDURE NomProcedure

@EntierEntree int = 10,
@ChaineSortie varchar(150) out

AS
BEGIN

Select TOP (@EntierEntree) CustomerID,CompanyName
FROM Customers

SET @ChaineSortie = CASt(@@ROWCOUNT as varchar(10))
                    + ' Lignes traitées';
Return 1

END
GO
```

Nom de la procédure

Paramètres en Entrée-Sortie, Sortie avec valeur par défaut

Corps de la méthode

Valeur retour

Create, Alter, et Drop Procedure

Procédure stockée : exécution

```
Declare @Sortie varchar(150);
Declare @ret int;

EXECUTE @ret= Nomprocedure 20,@Sortie out

Print @Sortie;
Print @ret;
```

Exécution avec passage des paramètres par position

@Ret représente la valeur de retour

```
Declare @Sortie varchar(150);
Declare @ret int;

EXECUTE @ret= Nomprocedure @EntierEntree=20,
        @ChaineSortie= @Sortie out

Print @Sortie;
Print @ret;
```

Exécution avec passage de paramètres nommés : technique propre à Microsoft