LA PROGRAMMATION

La programmation nous donne la possibilité de créer :

- des procédures stockées
- des fonctions
- des déclencheurs

Et même d’écrire une suite de commande à effectuer sur notre serveur.

Un programme peut se décomposer en trois parties :

Une partie déclarative

Une partie traitement

Une partie gestion des erreurs

```
DECLARE
      ...
BEGIN
      ...
END;
```

LES CURSEURS

Un curseur est un élément qui permet de stocker le résultat d’une requête contenant plusieurs lignes.

Il faut le déclarer dans la section déclarative.

Il faut l’ouvrir par OPEN, l’exécuter par FETCH et le fermer par CLOSE.

Exemple

LE CONTRÔLE DES FLUX

Plusieurs commandes permettent d’ajouter des conditions d’exécutions à la liste des commandes à exécuter :

- Boucle WHILE
- Condition IF … ELSE
- Condition CASE

Boucle WHILE

Le WHILE permet de répéter un bout de code tant que la condition testée au début est vraie.

Si la condition est fausse, on sort directement de la boucle sans exécuter le code.

```
BEGIN
    DECLARE @i int;
    SET @i = 0;
     WHILE @i < 8
        BEGIN
            SET @i = @i + 1
        END
    print @i
END;
```

Les structures conditionnelles CASE et IF

CASE permet de tester une variable et, en fonction des valeurs possibles, de réaliser l'action adéquate.

Une seule action est vérifiée à la fois, chaque condition WHEN est exclusive.

```
BEGIN
     DECLARE @i int;
     SET @i = 0;

     WHILE @i < 8
        BEGIN
            SET @i = @i + 1
            SELECT CASE @i
                WHEN 5 THEN 'OK'
                ELSE 'NON OK'
            END
        END
     print @i
END;
```

Les structures conditionnelles CASE et IF

IF permet d'exécuter un bloc de code en fonction d'une condition testée

```
BEGIN
     DECLARE @i int;
     SET @i = 0;

     WHILE @i < 8
        BEGIN
            SET @i = @i + 1
            if @i = 5
                BREAK
            ELSE
                CONTINUE
        END
     print @i
END;
```

LA GESTION DES ERREURS EN TRANSACT SQL

La gestion des erreurs permet d’anticiper les problèmes qui peuvent se dérouler au cours de l’exécution d’un programme.

Le principe est de tester le code dans un premier bloc avec

```
BEGIN TRY
    …
END TRY
```

puis d’intercepter l’exception avec

```
BEGIN CATCH
    …
END CATCH.
```

Quelques exemples :

```
DECLARE @i int
BEGIN
   BEGIN TRY
      SET @i = 2
      SET @i = @i / 0
   END TRY
   BEGIN CATCH
      SELECT ERROR_NUMBER() AS ErrorNumber
           , ERROR_MESSAGE() AS ErrorMessage
           , ERROR_LINE() AS ErrorLine;
   END CATCH
END;
```

```
DECLARE @i int
BEGIN
   BEGIN TRY
      SET @i = 2
      SET @i = @i / 0
   END TRY
   BEGIN CATCH
      IF @@ERROR = 8134
         SET @i = @i / 1
      print @i
   END CATCH
END;
```

PROCEDURES STOCKEES

Le principal avantage de la procédure stockée est de garder, au niveau de la base de données, un suite d’opérations répétitives auxquelles on fait appel fréquemment.

Un autre avantage de la procédure stockée est que l’on peut lui passer des paramètres.

Squelette de procédure

```
CREATE OR ALTER PROCEDURE <nom procédure>
 [(@<variable 1> <format>,
   @<variable 2> <format>,
   ... ... )]
 AS

BEGIN

... ...

[BEGIN TRY
... ...
END TRY]
[BEGIN CATCH
... ...
END CATCH]
END;
```

FONCTIONS STOCKEES

Dans le même exemple, il est également possible de créer une fonction à la place de la procédure.

La différence entre une fonction et une procédure ?

La première renvoie une valeur.

Squelette de procédure

```
CREATE OR ALTER FUNCTION <nom fonction>
 [(@<variable 1> <format>,
  @<variable 2> <format>,
   ... ... )]
   RETURNS <format>
 AS

BEGIN

... ...

END;
```

SUPPRESSION

La commande pour supprimer une procédure ou une fonction est DROP

```
DROP <’PROCEDURE’ ou ’FUNCTION’> <Nom procédure ou fonction>;
```

Exemple :

```
DROP FUNCTION PRIX_CHAMBRE;
```

LES DECLENCHEURS (TRIGGERS)

Un déclencheur, ou trigger en anglais, permet de lancer des commandes qui vont s’exécuter à chaque fois qu’un événement se produit sur une table.

Les déclencheurs sont souvent utilisés pour gérer l’intégrité fonctionnelle d’une application. Ils permettent de réaliser des contrôles sur le contenu des tables en automatique.

En général, on code les contrôles dans les programmes applicatifs exécutés côté client. Les déclencheurs permettent d’ajouter d’autres contrôles qui seront exécutés côté serveur.

Un déclencheur peut se déclencher avant ou après l’ordre SQL demandé. On l’indique par AFTER ou BEFORE.

Dans SQL Server, il peut s’exécuter aussi à la place : INSTEAD OFF.

Dans un trigger BEFORE, on peut contrôler avant toute modification de la base certains éléments et empêcher ainsi les mises à jour.

Dans un trigger AFTER, la mise à jour a eu lieu et on déclenche les actions qui en découlent.

Syntaxe générale d’un déclencheur :

```
CREATE ou ALTER TRIGGER <nom du trigger> ON <TABLE>
[BEFORE ou AFTER] [INSERT ou DELETE ou UPDATE]
ON <nom de table>
(FOR | AFTER | INSTEAD OF )
{ [ DELETE ] [ , ] [ INSERT ] [ , ] [ UPDATE ] }
AS

DECLARE
... ...
BEGIN
... ...

END;
```

Création d’un déclencheur de contrôle et mise à jour dans une table

Il est possible, à l’insertion dans une table de vérifier une valeur dans une autre table et, en fonction du résultat, d’accepter ou de refuser l’insertion.

Par exemple, lors de la création d’une ligne dans la table CHAMBRES, on déclenche une vérification du type de chambre : il faut que celui-ci soit présent dans la table TYPESCHAMBRE.

Exemple :

Création d’un déclencheur de suivi des mises à jour

Prenons l’exemple suivant : on souhaite enregistrer dans une table annexe le nombre de mises à jour réalisées dans la journée sur les tables CHAMBRES, TARIFS et TYPESCHAMBRE.

Quelles sont les étapes à mettre en place ?

Création d’un déclencheur de suivi des mises à jour

1 – création de la table de suivi

2 – création des triggers sur les tables concernées

3 – contrôle des bons résultats

Exemple