

Transact_SQL est une extension du langage SQL.

LES COMMENTAIRES

Commentaire sur la ligne avec : --

Commentaire sur plusieurs lignes /*
 */

LES VARIABLES

Variables Locales : variable qui n'existe que le temps de la durée d'exécution du code dont elle fait partie, leur nom est précédé de : @

```
DECLARE    @datecde          DATETIME ,
           @qte              INT ,
           @prix             DECIMAL ( 6 , 2 ) ,
           @codarticle       CHAR ( 4 ) ,
           @designation      VARCHAR ( 50 )
```

Variables globales : variables prédéfinies fournies par le système, leur nom est précédé de : @@

Quelques variables globales

```
@@rowcount      nombres de ligne ramenées par une requête
@@error         numéro de l'état d'erreur de l'instruction exécutée
@@fetch_status  variable d'état d'un curseur
```

etc...

INSTRUCTION D’AFFICHAGE : PRINT

Affiche un message construit à partir d'une chaîne de caractères ou d'une variable de type chaîne de caractères

```
PRINT 'ceci est un message'
```

```
DECLARE @message1 VARCHAR(15)
DECLARE @message2 VARCHAR(15)
SET      @message1 = 'ceci est '
SET      @message2 = 'un message'
PRINT    @message1 + @message2
```

ceci est un message

INSTRUCTION D'AFFECTION

SET @nom_variable = {expression}
 SELECT @nom_variable = {expression | instruction Select}

```
DECLARE @a INT, @b INT
SET @a = 1
SET @b = @a
PRINT @b
```

on pourrait écrire aussi : `SELECT @a = 1`

1

```
DECLARE @message VARCHAR(40)
SET @message = 'coucou au revoir'
SET @message = substring(@message,1,6)
PRINT @message
```

coucou

```
DECLARE @date DATETIME
SET @date = getdate()
PRINT @date
```

avr 8 2015 10:13AM

```
DECLARE @date DATETIME
SET @date = '12/03/2015'
PRINT @date
```

mars 12 2015 12:00AM

```
DECLARE @compteur integer
SET @compteur = 12
PRINT 'le compteur vaut : ' + convert(char(4), @compteur)
```

Le compteur vaut : 12

Affectation à partir d'une valeur issue d'une table

```
DECLARE @nbclients INT, @ncli INT

SELECT @nbclients = COUNT(*) FROM client
SELECT @ncli = ncli FROM client WHERE societe = 'ACME MFG.'
```

On pourrait écrire aussi :

```
SET @nbclients = (SELECT COUNT(*) FROM client)
SET @ncli = (SELECT ncli FROM client WHERE societe = 'ACME MFG.')
```

INSTRUCTIONS DE CALCUL

```
DECLARE @somme INT
DECLARE @nombre1 INT, @nombre2 INT
DECLARE @resultat DEC(5,2)
SET @nombre1 = 6
SET @nombre2 = 5
SET @somme = @nombre1 + @nombre2
SET @resultat = ( @somme * 2 / 5. ) % 4
PRINT @resultat
```

0.40

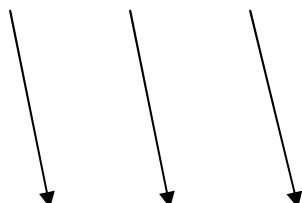
CURSOR - Récupération des lignes d'une requête dans un curseur

Déclaration des variables récupérant le contenu d'une ligne du curseur :

```
DECLARE    @ncli int
DECLARE    @societe varchar(20)
DECLARE    @nrep int
```

Allocation d'espace mémoire pour le curseur :

```
/*      le nom du curseur ne commence pas par @      */
DECLARE    c_client CURSOR FOR                          -- réserve un espace
SELECT      ncli, societe, nrep
FROM client
```



Instructions liées au curseur :

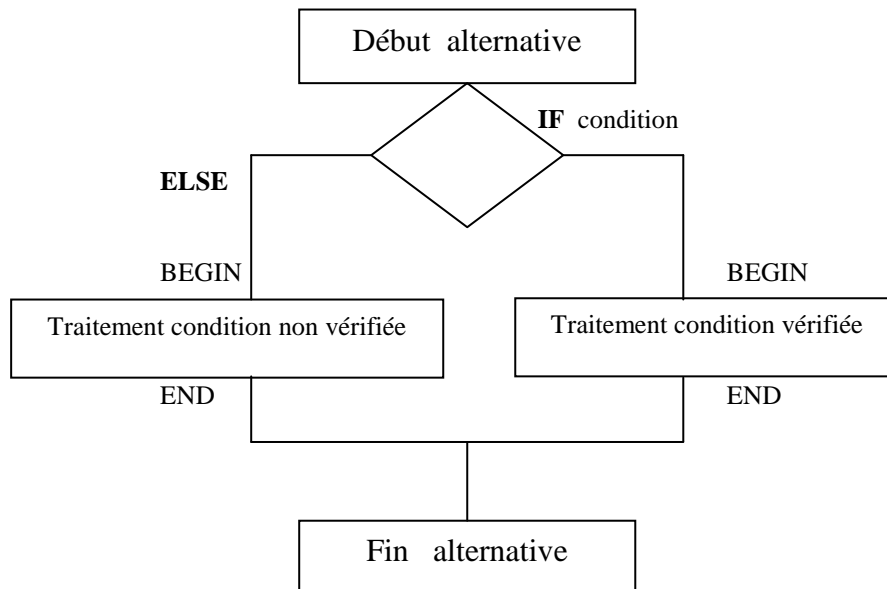
```
OPEN        c_client                                -- alimente le curseur
FETCH       c_client INTO @ncli, @societe, @nrep    -- lit une ligne
CLOSE       c_client                                -- vide le curseur
```

Variable liée au FETCH :

```
@@FETCH_STATUS      -- ( = 0 une ligne a été lue , <> 0 fin du curseur)
```

Libération des allocations du curseur :

```
DEALLOCATE c_client                                -- libère l'espace
```

L'ALTERNATIVE**FORME GENERALE**

```

--      début alternative

IF      condition
--      traitement condition vérifiée
        Instruction 1
ELSE
--      traitement condition non vérifiée
        BEGIN
        instruction1
        instruction2
        END

--      fin alternative
  
```

Dans le cas où il n'y a qu'une seule instruction dans un pavé (branche de l'alternative), il est possible de supprimer :
 BEGIN
 END

Exemples d'alternatives

1^{er} exemple : Comparaison de valeurs du code

Ce code détermine s'il y a assez de produits à vendre (100 minimum).
Il affiche 'NB PRODUITS SUFFISANT' ou 'NB PRODUITS INSUFFISANT'.

```
DECLARE      @nbproduits  INTEGER

--      début alternative
/*      chercher le nombre de produits dans la table      */
SELECT      @nbproduits = count(*)
FROM Produit
IF          @nbproduits < 100

--      Branche condition vérifiée
      BEGIN
      PRINT      'NB PRODUITS INSUFFISANT'
      END
ELSE

--      Branche condition non vérifiée
      BEGIN
      PRINT      'NB PRODUITS SUFFISANT'
      END

--      fin alternative
```

2^{ème} exemple : Tester le code retour d'une requête SQL

Ce code détermine s'il y a des produits dans la table des produits.
Il affiche : 'il y a des produits' ou 'il n'y a pas de produit'

```
--      début alternative
IF      EXISTS      (SELECT 'x'
                     FROM produit)

--      Traitement condition vérifiée
PRINT      'il a des produits'
ELSE

--      Traitement condition non vérifiée
PRINT      'il n''y a pas de produit'

--      fin alternative
```

3^{ème} exemple : Comparer des valeurs obtenues à partir des tables

Ce code détermine si la moyenne des montants des commandes du bureau 11 est supérieure à la moyenne des commandes.

Il affiche 'RESULTATS SATISFAISANTS' ou 'RESULTATS INSATISFAISANTS'

```
--      Début alternative
IF      (SELECT      AVG(montant)
FROM      COMMANDE C
JOIN      REPRESENTANT R      ON C.nrep = R.nrep
WHERE     nbur = 11)
>
  (SELECT      AVG(montant)
FROM      COMMANDE)

--      Traitement  moyenne de commandes du bureau > moyenne
BEGIN
PRINT      'RESULTATS SATISFAISANTS'
END
ELSE

--      Traitement  moyenne de commandes du bureau <= moyenne
BEGIN
PRINT      'RESULTATS INSATISFAISANTS'
END

--      Fin alternative
```

4^{ème} exemple : Tester si une requête a ramené une valeur

```
DECLARE      @numero int

--      Début alternative
SELECT      @numero = ncli
FROM      client
WHERE     societe = 'ACME MFG.'

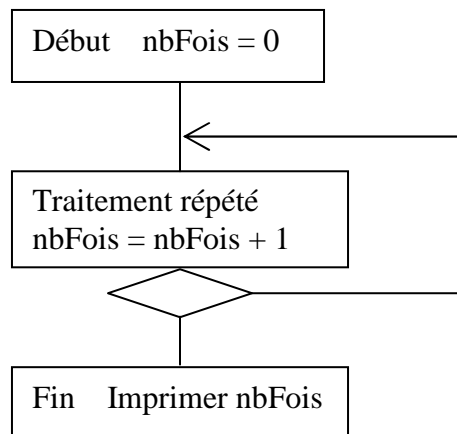
IF      @numero IS null
--      Traitement  aucun numéro trouvé
BEGIN
PRINT      '@numero n'est pas renseigné'
END

ELSE
--      Traitement  du numéro trouvé
BEGIN
PRINT      CAST(@numero AS char(4)) + ' renseigné'
END

--      Fin alternative
```

LA REPETITIVE

FORME « LOGIQUE » GENERALE

**LE BRANCHEMENT INCONDITIONNEL****GOTO**

L'exemple réalise la répétitive ci-dessus

```
DECLARE      @nbfois integer

--  début répétitive
SET          @nbfois = 0

trtimprimer:
--          traitement répétition
SET          @nbfois = @nbfois + 1
PRINT        @nbfois
IF           @nbfois < 2
GOTO         trtimprimer

--  fin répétitive
```

1
2

WHILE condition

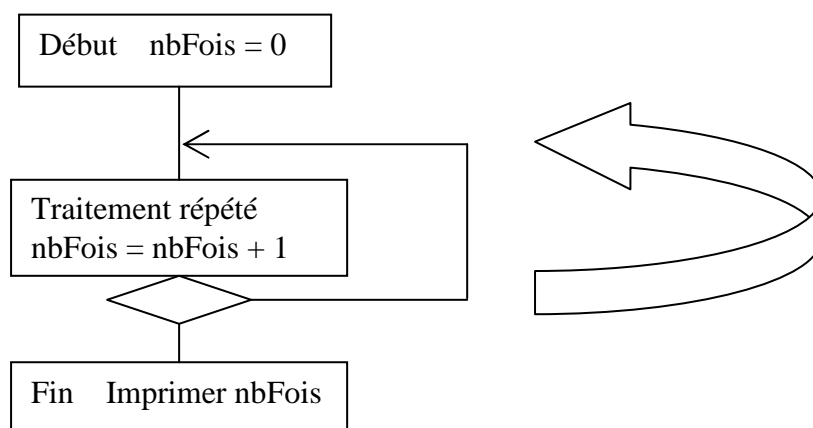
Instruction créée pour ne pas gérer la boucle avec un GOTO

```
--          début répétitive

          WHILE condition
--          traitement répété
              BEGIN
                  instruction1
                  instruction2
              END

--          fin répétitive
```

Dans le cas où le pavé ne contient qu'une seule instruction, il est possible de supprimer : BEGIN, END

PASSAGE AU CODE PAR « ABUS » DE REPRESENTATION**1^{er} exemple de répétitive avec WHILE**

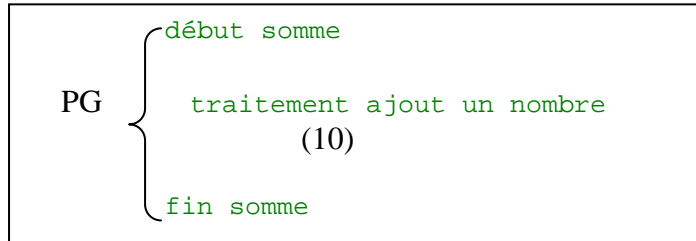
```
DECLARE      @nbfois integer

--          début répétitive
SET          @nbfois = 0

WHILE        @nbfois < 2
--          traitement répétitif
            BEGIN
                SET      @nbfois = @nbfois + 1
                PRINT    @nbfois
            END

--          fin répétitive
```

1
2

2^{ème} exemple de répétitive avec WHILE : Somme des 10 premiers nombres

```
DECLARE @somme integer, @compteur int, @nombre int
DECLARE @affichageresultat varchar(50)

--- début répétitive
SET @somme = 0
SET @nombre = 0
SET @compteur = 0

--- boucle de répétition tant que
WHILE @compteur < 10
BEGIN
    SET @nombre = @nombre + 1
    SET @somme = @somme + @nombre
    SET @compteur = @compteur + 1
END

--- fin répétitive
SET @affichageresultat = 'somme = ' + CONVERT(char,@somme)
PRINT @affichageresultat
```

INSTRUCTIONS DIVERSES

BREAK permet de sortir de la répétitive (le traitement enchaîne avec la première instruction qui suit le END du WHILE)

CONTINUE permet d'itérer une nouvelle fois en ignorant les instructions qui suivent (retour au début de la boucle WHILE)

RETURN arrête le déroulement des instructions (fonctionnement similaire à *BREAK*)

EXECUTE

permet de fabriquer des ordres SQL (SQL dynamique)

```
DECLARE    @ordresql VARCHAR(80)
SET        @ordresql = 'SELECT * FROM '
EXECUTE    (@ordresql + 'client')      -- lance : SELECT * FROM client
```

permet d'exécuter une procédure stockée (une procédure stockée permet un stockage dans le SGBD d'une succession d'ordres Transact_SQL)

```
EXECUTE    p_essai                    -- appelle la procedure : p_essai
```

RAISEERROR permet de sortir des messages d'erreur du même style que ceux de SQLServer

```
DECLARE    @errno INT,
            @errmsg VARCHAR(80)
-- Début bureau
IF        (SELECT count(*)
FROM      bureau
WHERE     region = 'SOUTH') = 0

    BEGIN                                     -- traitement table vide
    SET @errno = 30002
    SET @errmsg = 'Aucun bureau dans la région'
    RAISEERROR @errno @errmsg
    END
ELSE
    BEGIN                                     -- traitement table pleine
    PRINT 'Traitement en cours'
    END
-- Fin bureau
```

```
Msg 30002, Level 16, State 1, Line 14
Aucun bureau dans la région
```

LE BLOC TRY- CATCH : gestion des erreurs par exception

Soit cette suite d'instructions

```
INSERT INTO client VALUES (2101, 'DISNEY PRODUCTION', 102, 70000)

IF @@ERROR <> 0
    PRINT 'erreur insertion'
ELSE
    PRINT 'insertion faite'
```

```
Msg 2627, Level 14, State 1, Line 1
Violation de la contrainte PRIMARY KEY 'PK_CLIENT__7ADC2F5E'. Impossible d'insérer une clé en
double dans l'objet 'dbo.client'.
L'instruction a été arrêtée.
erreur insertion
```

Peut être traitée comme cela :

```
BEGIN TRY
    INSERT INTO client VALUES (2101, 'DISNEY PRODUCTION', 102, 70000)
    PRINT 'Insertion faite'
END TRY

BEGIN CATCH
    PRINT 'Erreur d''insertion dans la table des clients'
END CATCH
```

```
Erreur d'insertion dans la table des clients
```

Conversion explicite

```
PRINT 'le numero ' + 1
```

Msg 245, Level 16, State 1, Line 2

Échec de la conversion de la valeur varchar « le numero » en type de données int.

correction :

```
PRINT 'le numero ' + '1'
```

```
DECLARE @numero INT
```

```
SET @numero = 2
```

```
PRINT 'le numero ' + @numero
```

Msg 245, Level 16, State 1, Line 3

Échec de la conversion de la valeur varchar « le numero » en type de données int.

correction :

```
DECLARE @numero INT
```

```
SET @numero = 2
```

```
PRINT 'le numero ' + convert(char(4),@numero)
```

```
DECLARE @date DATETIME
```

```
SET @date = '12/03/2015'
```

```
PRINT @date
```

```
PRINT convert(char(10),@date ,103)
```

mars 12 2015 12:00AM 12/03/2015

Conversion implicite

```
declare @compteur int
```

```
set @compteur = 1
```

```
set @compteur = @compteur + '1'
```

```
print @compteur
```

2

Conversion non autorisée

```
declare @compteur int
```

```
set @compteur = 1
```

```
set @compteur = @compteur + convert(INT,'A')
```

Msg 245, Level 16, State 1, Line 3

Échec de la conversion de la valeur varchar « A » en type de données int.