Programmation de bases de données:
T-SQL, Introduction

Plan de la séance:

- Retour sur la dernière séance:
  - Point de vue des enseignants.
  - Point de vue des étudiants.
- Introduction
- Les variables
- Les mots BEGIN - END
- Les structures de contrôle
- Exemples
- Points clés
- Suite du Laboratoire 1

Introduction:

- Retour sur la dernière séance:
  - Point de vue des étudiants
  - Points de vue des enseignants.
- Le PowerPoint ….. C’est bien, le cours sur le site c’est mieux et c’est recommandé ! **Le PowerPoint est un guide et non un contenu.**
- Le SQL d’oracle est le même que celui de SQL Server
- Interface conviviale pour SSMS
- Les commandes SQL doivent-elles écrites en majuscules ? C’est la norme. Mais comme vous avez une interface conviviale, vous pouvez coder en minuscules.

Introduction

- La plupart des SGBDs relationnels offrent une extension du SQL, en y ajoutant des déclarations de variables, des structures de contrôles (alternatives et les répétitives) pour améliorer leurs performances

- Transact-SQL ou T-SQL ou TSQL est l'extension du langage SQL pour Microsoft SQL Server et Sybase.
- Transact-SQL est un langage **procédural** permettant d'implémenter des fonctionnalités de bases de données que SQL seul ne peut implémenter.
- Nous présenterons rapidement les blocs « anonymes ». Les éléments du langage seront utilisés dans les procédures stockées.

Les variables, définition

- Dans Transact SQL, on utilise le mot réservé **DECLARE** pour déclarer des variables.
- Un seul DECLARE est suffisant si vos déclarations sont suivies par une virgule. La dernière variable déclarée se termine par point virgule.
- Les noms de variables sont précédés **du symbole @**
- Les types de variables, sont les types SQL
- Les variables peuvent être initialisées avec des valeurs en utilisant la fonction **SET** .

Les variables, déclaration

```
DECLARE
@CHOIX int ;
SET @CHOIX =1;

DECLARE
@variable  INT =12,
@nom VARCHAR(30) ='Patoche',
@date DATE ='2020-08-21';
PRINT @variable; PRINT @nom; PRINT @date;
```

Les variables: Affectation de valeurs

Par le mot réservé SET

```
DECLARE
@variable  INT =12,
@nom VARCHAR(30),
@date DATE ;
SET @nom ='Patoche';
SET @date ='2020-09-30';

PRINT @variable; PRINT @nom; PRINT @date;
```

Les variables: Affectation de valeurs

Par des données provenant de la base de données: **SELECT**

```
USE bdEmployesClg
DECLARE
@salaire_min MONEY;

BEGIN
SELECT @salaire_min =MIN(SALAIRE) FROM Employes;
PRINT @salaire_min;
END;
```

BEGIN….END

Les mots réservés : BEGIN …END

Ces mots réservés permettent de définir un bloc ou un groupe d’instructions qui doivent être exécutées. Un peu comme { } en C#.

L’instruction IF..ELSE

IF Boolean_expression

{ sql_statement | statement_block }

[ELSE

{ sql_statement | statement_block }]

L’instruction IF.. ELSE IF

```
DECLARE
@vsalaire MONEY;
BEGIN
SELECT  @vsalaire =AVG(SALAIRE) FROM Employes WHERE codeDep ='inf’;

        IF (@vsalaire>60000)
        UPDATE Employes SET Salaire = Salaire + 0.01*salaire  WHERE codeDep ='inf';
        ELSE
         UPDATE  Employes SET Salaire = Salaire + 0.02*salaire   WHERE codeDep ='inf';
END;
```

L’instruction IF.. ELSE IF

IF Boolean_expression

{ sql_statement | statement_block }

[ELSE IF Boolean_expression

{ sql_statement | statement_block }]

[ELSE

{ sql_statement | statement_block }]

L’instruction CASE

```
CASE input_expression
    WHEN when_expression THEN result_expression [ ...n ]
    [ ELSE else_result_expression ]
END
```

Ou

```
CASE
   WHEN Boolean_expression THEN result_expression [ ..n ]
    [ ELSE else_result_expression ]
END
```

L’instruction CASE

```
SELECT    nom,prenom, codeDep =
        CASE   codeDep
              WHEN 'inf' THEN 'Informatique'
              WHEN 'rsh' THEN 'Ressources humaines'
              WHEN 'ges' THEN 'Gestion’
              WHEN 'rec' THEN 'Recherche et developpement'
             ELSE 'aucun département connu'
        END, salaire
FROM employes
```

Important: Une jointure est beaucoup moins couteuse qu’un CASE pour ce cas.

L’instruction CASE

```
UPDATE employes SET salaire =
  (
   CASE
     WHEN (salaire < 25000 ) THEN salaire + 0.05*salaire
      ELSE (salaire + 0.01*salaire)
     END
  ) ;
```

L'instruction WHILE

Syntaxe:

WHILE Boolean_expression

{ sql_statement | statement_block | BREAK | CONTINUE }

Exemple:

```
BEGIN
        WHILE (SELECT  AVG (salaire) FROM employes ) <= 150000
                BEGIN UPDATE employes SET salaire = salaire +1000;
                        IF (SELECT MAX (salaire) FROM employes) > 500000
                        BREAK;
                        ELSE CONTINUE;
                END;
END;
```

L'instruction WHILE

Pour un livre donné, insérer plusieurs exemplaires:

Table Livres:

```
create table Livres (
idLivre char(3) constraint pk_livres primary key,
anneeEdition int ,
maisonEdition varchar(50),
auteur varchar(50),
);
```

Table Exemplaires

```
create table Exemplaires (
numEx varchar(20) CONSTRAINT PK_EXEMPLAIRES primary key,
idLivre char(3) not null,
constraint fk_exemplaires_livres foreign key(idLivre) references Livres(idLivre)
);
```

L’instruction WHILE

Pour un livre donné, insérer plusieurs exemplaires:

On obtient: Dans la table Livres:

idLivre	anneeEdition	maisonEdition	auteur
PHP	2020	CLG	Patoche

Dans la table Exemplaires:

numEx	idLivre
EXPHP-1	PHP
EXPHP-2	PHP
EXPHP-3	PHP
EXPHP-4	PHP
EXPHP-5	PHP

L’instruction WHILE

Le code est: (attention, ici nous n’avons pas mis en œuvre les Transactions)

```
begin
declare @idLivre char(3)='PHP',
        @cnt int = 1,
        @anneeEdition int =2020,
        @nbExemplaires smallint =5,
        @maisonEdition varchar(50) ='CLG’,
        @auteur varchar(30)='Patoche’;

          insert into Livres(idLivre, anneeEdition,maisonEdition,auteur)
          values(@idLivre,@anneeEdition,@maisonEdition,@auteur);
             while @cnt <= @nbExemplaires
                    begin
                    insert into Exemplaires(numEx,idLivre) values(concat('EX',@idLivre,'-',@cnt), @idLivre);
                    set @cnt = @cnt + 1;
                    end;
end;
```

Le mot réservé GO

Le mot réservé : GO

GO ne fait pas partie du SQL. C'est un mot réservé **optionnel** qui permet de finir un lot d'instructions.

Exemple:

Les procédures et les fonctions stockées sont comme vos fonctions C#. En principe, à moins que ce soit dans un programme principal, vous ne pouvez pas mettre les fonctions et les procédures dans un même fichier.

Si vous finissez votre procédure ou votre fonction par GO dans ce cas, vous pouvez mettre vos procédures et ou fonction dans un même fichier sans qu'il y soit des erreurs. Évidement ces procédure s'exécutent séparément.

Points clés:

- Dans Transact-SQL la déclaration de variable se fait par le mot DECLARE
- Les variables sont précédées du symbole @
- Les types de variable sont les type SQL de SQL Server
- L'affectation de valeurs aux variables se fait par = ou le SET
- L'affectation de valeurs issues de la base de données à une variable se fait par un SELECT : SELECT @variable = donnée issue de la BD
- Chaque bloc d'instructions commence par BEGIN et fini par END
- Le IF a le même rôle de que ce que vous connaissez en C#, cependant il n' a pas la même syntaxe.
- Le WHILE a le même rôle de que ce que vous connaissez en C#, cependant il n' a pas la même syntaxe.
- Le GO est optionnel (n'est pas du SQL) et permet de terminer un lot d'instructions.
- Surtout, évitez de faire des IF imbriqués à l'infini. Utiliser la fonction RETURN pour sortir d'un IF.

CONCLUSION

QUESTIONS ??