Transact SQL

Les tables

- **Les types de données (sp_help type_de_donnees)**

Char	Chaine de caractères de longueur fixe d'un maximum de 8000 caractères.
Nchar	Chaine de caractères Unicode, d'un maximum de 4000 caractères.
Varchar	Chaine de caractères de longueur variable. Il est possible de préciser la valeur max, ce qui permet d'entrer des longueurs de chaines de caractères de 2^31 caractères.
Nvarchar	Chaine de caractères Unicode, d'un maximum de 4000 caractères. En spécifiant max, le texte peut avoir une longueur maximum de 2^31 caractères.
Int	Nombre entier compris entre -2^31 et 2^31-1.
Bigint	Nombre entier compris entre -2^63 et 2^63.
Smallint	Nombre entier compris entre -2^15 et 2^15-1.
Tinyint	Nombre positif compris entre 0 et 255.
Decimal/Numeric	Nom exact de précision C (nombre entier) et D chiffres après la virgule tel que : Decimal (entre 0 et 38, 2) =Un nombre (2 chiffres après la virgule). Les valeurs supportées vont de -99999,999 à 99999,999.
Float	Nom approché de N chiffres tel que pour Float(N), N vas de 1 à 53.
Real	Identique à Float(24).

Les tables

- **Les types de données**

Money	Supporte les nombres monétaires compris entre -922337203685477,5808 et 922337203685477,5807 donc des nombres sur 8 octets.
Smallmoney	Supporte les nombres monétaires compris entre -214748,3648 et 214748,3647 donc des nombres sur 4 octets.
Date	Permet de stocker une donnée de type date comprise entre le 01/01/0001 et le 31/12/9999 avec la précision d'une journée.
Datetime	Permet de stocker une date et une heure sur 8 octets. Datetime a une précision accrue par rapport à Smalldatetime (précision de 3,33 millisecondes).
Datetime2	Il permet de stocker une donnée de type date et heure comprise entre le 01/01/0001 et le 31/12/9999 avec une précision de 100 nanosecondes.
Smalldatetime	Permet de stocker une date et une heure sur 4 octets. Les dates possibles vont du 1er Janvier 1900 au 6 Juin 2079, avec une précision à la minute près.
Datetimeoffset	Il permet de stocker une donnée de type date et heure comprise entre le 01/01/0001 et le 31/12/9999 avec une précision de 100 nanosecondes. Les informations sont stockées au format UTC.

Les tables

- **Les types de données**

Binary	Supporte des données binaires sur n octets (De 1 à 255).
Varbinary	Supporte des données binaires sur n octets (De 1 à 8000). L'argument Max, permet de réserver 231 octet au maximum.
Bit	Valeur entière Booléenne pouvant prendre la valeur 0, 1 ou NULL.
Xml	Permet de stocker des documents Xml au sain d'une table dans une colonne.
Table	Type de données qui permet de renvoyer un ensemble de données en vue d'une utilisation future. Il est en particulier utilisé pour la création de tables temporaires.

- **Types de données définis par l'utilisateur**

```
CREATE TYPE Nom_Type
    FROM Type_existant NULL
```

Les tables

- Créer / Supprimer une table

```
CREATE TABLE Client
(
[Id_Client]  NUMERIC(8,0)  IDENTITY PRIMARY KEY,
[Nom_Client] varchar(50) NOT NULL,
[Prenom_Client] varchar(50) NOT NULL,
[Numero_Client] varchar(20) NOT NULL, [Adresse_Client] varchar(50) NOT NULL,
[Mail_Client] varchar(50) UNIQUE NOT NULL
);
```

IDENTITY : auto-incrémentation, une seule par table.

DROP TABLE Client

Remarque : l'accès à un objet : ***serveur.base_cible.schema.objet***
Ignorer le serveur et le schema : ***base_cible..objet***

Les tables

- **Les contraintes :** deverons porter un nom

  - **PRIMARY KEY :** CREATE TABLE MATABLE1 (COLONNE1 int CONSTRAINT PK_Nom_Contrainte PRIMARY KEY)
    Ou : ALTER TABLE MATABLE1 ADD CONSTRAINT PK_PRIMARY PRIMARY KEY (COLONNE1)
  - **UNIQUE** : CREATE TABLE MATABLE1 (COLONNE1 int CONSTRAINT Unq_Nom_Contrainte UNIQUE)
    Ou : ALTER TABLE MATABLE1 ADD CONSTRAINT Nom_Contrainte UNIQUE .
  - **REFERENCE :** CREATE TABLE MATABLE2 (COLONNE1 int CONSTRAINT FOREIGN KEY COLONNE1 REFERENCE MATABLE1 [COLONNE1] Options/NO OPTION)
    ou : ADD CONSTRAINT FOREIGN KEY COLONNE1 REFERENCE MATABLE1 [COLONNE1] Options)

  Options : *ON DELETE CASCADE / ON UPDATE CASCADE / SET NULL / SET DEFAULT*

  - **DEFAULT :** CREATE TABLE MATABLE1 (COLONNE1 int DEFAULT Valeur)
    Ou : ALTER TABLE MATABLE2 ADD CONSTRAINT DEFAULT Valeur FOR COLONNE1
  - **CHECK :** CREATE TABLE MATABLE1 (COLONNE1 int CHECK (expression_booleenne))
    Ou : ALTER TABLE MATABLE2 ADD CONSTRAINT CHECK (expression_booleenne)

Les tables

- Manipuletr une table

  - **Ajout** : INSERT INTO Entrepos (Nom_Entrepos) VALUES ('Entrepos1') ;

  - **Modification :** UPDATE [Client]
    SET [Nom_Client] = <Nom_Client>
    ,[Prenom_Client] = <Prenom_Client>
    ,[Numero_Client] =< Numero_Client>
    ,[Adresse_Client] = <Adresse_Client>
    WHERE <Conditions de recherche>

  - **Retrait :** DELETE FROM [Client] WHERE <Conditions de recherche>
    Toutes les lignes : TRUNCATE TABLE Nom_table

SQL Procédurale

- Les variables
  - Variables : précédée par @ (variables système : @@)
    - *Définition : DECLARE* @Nom_variable type_variable;
    - Utilisation :

      set @Nom_variable valeur
      SELECT @Nom = nomcl, @Adresse = adresse FROM client (une seule ligne)
    - Regrouper plusieurs lignes :

```
DECLARE @Colonne  varchar(8000)
SET @Colonne = ' '
SELECT @Colonne = @Colonne +CategoryName +', ‘  FROM  dbo.Categories ;
print @Colonne
```

SQL Procédural

- Transaction :
  - Démarrage :BEGIN TRANSACTION nom_transaction
  - Validation : COMMIT TRANSACTION nom_transaction
  - Déclaration d'un point de contrôle : SAVE TRANSACTION nom_point_de_retour
  - Annulation : ROLLBACK TRANSACTION nom_transaction OR nom_point_de_controle
- Exemple :

```
BEGIN TRANSACTION Transaction1
        UPDATE dbo.Client  SET Nom_Client = 'ANDREO'  WHERE Nom_Client = 'CASA'
        BEGIN TRANSACTION Transaction2
                UPDATE dbo.Client  SET Nom_Client = 'VASSELON'  WHERE Nom_Client = HOLLEBECQ'
        COMMIT TRANSACTION Transaction2
ROLLBACK TRANSACTION Transaction1
```

SQL Procédural

- Lot et script :
  - Un lot est une suite de transactions et d'instructions qui seront exécutées en un seul et unique bloc.
  - Se termine par l'instruction GO.
  - Une simple erreur entraine l'annulation du bloc.
  - Pas possible de rassembler deux des instructions suivantes dans un même lot : CREATE PROCEDURE, CREATE RULE, CREATE DEFAULT, CREATE TRIGGER, CREATE VIEW.

SQL Procédural

- Curseur
  - Un curseur est un nom symbolique associé à une instruction select. Il se compose de :
    - Les résultats du curseur : ensemble (table) de lignes résultant de l'exécution d'une requête associée au curseur.
    - Position du curseur : pointeur sur une ligne de résultats curseur (fetch).
  - Déclaration

Declare nom_curseur cursor for instruction_select

[for {read only | update [of liste_noms_colonnes]}]

  - Ouverture

OPEN nom_curseur

  - Lecture

fetch nom_curseur [into liste_extraction_cible]

  - Fermeture

CLOSE curseur

  - Libération du curseur

Deallocate cursor nom_curseur

SQL Procédural

- Curseur

```
declare cur_clients cursor
    for
    select code_client,societe_client,nom_client,prenom_client from table_client order by
    code_client
    /* ouverture du curseur */
    open cur_clients
    /* récupération de la dernière ligne */
        fetch last from cur_clients                    -- first
    /* lecture des différentes lignes du curseur ("en remontant") */
        while @@fetch_status=0
                begin
                        fetch prior from cur_clients    -- next
                end
    /* fermeture et libération du curseur */
    close cur_clients
    deallocate cur_clients
    GO
```

SQL Procédural

- Procédure stockée
  - Des ensembles d'instructions du LMD, exécutés par simple appel de leur nom ou par l'instruction EXEC.
  - L'exécution est très rapide: les instructions SQL contenues dans les procédures stockées sont précompilées.
  - Alléger le traitement du côté client.
  - Peut avoir des paramètres.
  - Il existe une multitude de procédures stockées prédéfinies qui servent principalement à la maintenance des bases de données utilisateur « sp_ ».
  - Définition:

```
CREATE PROCEDURE <Procedure_Name>
        <@Param> <Datatype_For_Param> = <Default_Value_For_Param>
        AS
        BEGIN
                utilisation de DML avec  <@Param>
        END
```

SQL Procédural

- Procédure stockée
  - Suppression

  DROP PROCEDURE <Procedure_Name>
  - Exemple

```
CREATE PROCEDURE EXEMPLE_1
AS
begin
        select * from client3
        declare @effectif int
        select @effectif=count(*) from client3
        if @effectif >= 10
                print 'Nb. d''enregistrements >= 10'
        else
                print ' Nb. d''enregistrements < 10'
end
```

SQL Procédural

- Fonction utilisateur
  - Définie par le concepteur de la base pour des besoins de traitement.
  - Renvoie une valeur/table
  - Syntaxe :

```
CREATE FUNCTION [ schema. ] nom_fonction     --
(
[ { @parametre1[AS] type [ = valeur_défaut ] } [ , @parametre2 ... ]
)

RETURNS type_résultant
[ AS ]
BEGIN                                   -- begin et end que pour les fonctions de type valeur
        code

 RETURN valeur_résultante
END
```

SQL Procédural

- Fonction utilisateur

```
CREATE FUNCTION FCT_EXEMPLE_1
(@valeur decimal)
RETURNS decimal(7,2) AS
BEGIN
return power(@valeur,3)/4
END
```

```
CREATE FUNCTION FCT_EXEMPLE_2
(@Valeur_DateCommande datetime)
RETURNS table AS
return(select
ncom,date1,client.ncli,nom,prenom from
commande,client
where commande.ncli=client.ncli and
date1>@Valeur_DateCommande)
```

select valeur, **dbo.fct_exemple_1(valeur) from table_exemple_1**

select * from **fct_exemple_2('05/03/2016')**

Déclencheur

- Permet d'effectuer automatiquement des taches administratives de maintient de la base de données.
- Utilise le retour d'une manipulation donnée pour déclencher un traitement.
- Se lance en arrière plan.

- Trigger de LMD: insert /update / delete
  - Désigné pour une table / vue
  - CREATE TRIGGER *TriggerName*
    ON *TableName* AFTER/ AFTER/INSTEAD OF
    INSERT / UPDATE / DELETE
    AS BEGIN *TriggerCode [*FROM UPDATED,
    FROM INSERTED, FROM DELETED *] END*

```
CREATE TRIGGER Soustraction_Stock
ON Commande  AFTER INSERT
AS
BEGIN
    DECLARE @Id_Stock int, @Quantite int
    SELECT @Id_Stock = Id_stock FROM INSERTED
    SELECT @Quantite = Quantite FROM INSERTED
    UPDATE Stock
    SET  Quantite =  Quantite - @Quantite
    WHERE  Id_Stock = @Id_Stock
END
```

- Trigger de LDD: create / alter / drop
  - Désigné pour une BD / serveur
  - CREATE TRIGGER TriggerName
    ON DATABASE/ALL SERVER FOR/AFTER *WhatEvent*
    AS *TriggerCode*

```
CREATE TRIGGER LogNewTableCreation
ON DATABASE FOR CREATE_TABLE
AS
BEGIN
    INSERT INTO DatabaseOperations
    VALUES(SUSER_SNAME(), 'A new table was
    created', GETDATE())
END
```

Exercices

- Créer une procédure stockée permettant de vérifier si une table, dont on passera le nom en paramètre, existe déjà (sysobjects pour les objets d'une BD).
- Définir une fonction qui permet de donner la taille de fichier de donnée d'une base passée en paramètre (sysfiles pour les info des fichiers d'une base donnée).
- Définir un curseur permettant de calculer la taille de l'ensembles des bases dans le système. (sysaltfiles)
- Définir un déclencheur qui indique la date de création et l'identité de l'utilisateur créant une nouvelle table.
- Un déclencheur qui vérifie la taille de fichier de données de toute base de données nouvellement ajoutée. Le déclencheur annule la création si la taille dépasse 8MB.

  EVENTDATA().value('(/EVENT_INSTANCE/DatabaseName)[1]', 'varchar(256)')

  EVENTDATA().value('(/EVENT_INSTANCE/TSQLCommand/CommandText)[1]','nvarchar(max)')