Support de cours SQL Server

Outils de SQL Server

 Moteur de base de données	Le moteur de base de données permet de stocker, traiter et sécuriser les données
 Analysis Services - Données multidimensionnelles	Analysis Services permet de créer et gérer des structures multidimensionnelles qui contiennent des données agrégées à partir d'autres sources de données, telles que des bases de données relationnelles.
 Integration Services	Integration Services est une plateforme permettant de créer des solutions d'intégration de données qui autorisent les processus d'extraction, de transformation et de chargement (ETL) pour le Data Warehouse.
 Reporting Services	Reporting Services fournit des fonctionnalités Web de création de rapports d'entreprise.

Types de bases de données

OLTP	OLTP (On-Line Transactional Processing) – Moteur SQL Server - est un service central qui permet de stocker, traiter et sécuriser les données
OLAP	OLAP (On-Line Analytical Processing) - Moteur Analysis Services - a pour but d'organiser les données à analyser par domaine/thème et d'en ressortir des résultats pertinents pour le décideur

SQL Server est un système de gestion de base de données relationnelle (SGBDR) qui :

- ✓ gère le stockage des données pour les transactions et l'analyse ;
- ✓ répond aux requêtes des applications clientes

Vous pouvez utiliser SQL Server pour traiter des transactions, stocker et analyser des données, et développer de nouvelles applications.

SQL Server est une suite de produits et de technologies qui répondent aux exigences de stockage des données des environnements OLTP et OLAP.

SQL Server gère les bases de données **OLTP** et **OLAP**.

Les données d'une base de données OLTP sont généralement organisées en tables relationnelles afin de réduire les informations redondantes et d'accroître la vitesse des mises à jour. SQL Server autorise un grand nombre d'utilisateurs à effectuer des transactions et à changer simultanément les données en temps réel dans les bases de données OLTP.

Liste des bases de données

MSDB	La base de données **msdb** est utilisée par l'Agent SQL Server pour planifier des alertes et des travaux, ainsi que par d'autres fonctionnalités telles que Service Broker et la messagerie de base de données.
MASTER	La base de données **master** contient l'intégralité des informations système relatives à un système SQL Server
MODEL	La base de données **model** fait office de modèle pour toutes les bases de données créées sur une instance de SQL Server
TEMPDB	La base de données système **tempdb** est une ressource globale disponible pour tous les utilisateurs connectés à l'instance de SQL Server et sert de conteneur aux éléments suivants :

Création d'une base de données

Fichier	Description
Primaire	Le fichier de données primaire contient les informations de **démarrage de la base de données, et il pointe vers les autres fichiers de la base de données**. Les objets et les données utilisateur peuvent être stockés dans ce fichier ou dans des fichiers de données secondaires. Chaque base de données comprend un fichier de données primaire.
Secondaire	Les fichiers de données secondaires sont **facultatifs**, définis par l'utilisateur, et ils stockent les données utilisateur. Les fichiers secondaires peuvent être utilisés pour répartir des données sur plusieurs disques en plaçant chaque fichier sur un lecteur de disque distinct. En outre, si la taille d'une base de données excède la taille maximale autorisée pour un fichier Windows, vous pouvez avoir recours aux fichiers secondaires afin que la base de données puisse continuer à croître.
Journal des transactions	Les fichiers journaux des transactions contiennent les informations du journal qui sont utilisées pour la restauration de la base de données. Chaque base de données **doit posséder** au moins un fichier journal. L'extension de fichier recommandée pour les journaux des transactions est .ldf.

Création de tables

Créer une table CLIENTS contenant les champs :

```
/* Création d'une table CLIENT simple sans Contraintes */
IF OBJECT_ID('Client', 'U') IS NOT NULL DROP TABLE Client

create table Client(
IdClient INT
,Civilite char(8)
,Nom varchar(25) NOT NULL
,Prenom varchar(25)
,Ville varchar(30)
,Tel char(14)
,EnCoursHT money
,EnCoursTTC AS (EnCoursHT*(1.196))
,NoSS VARCHAR(13)
,Modif ROWVERSION
,DatePremiereVisite smalldatetime NOT NULL
,DatePremiereCommande smalldatetime NULL
)
```

```
--Informations sur une table
--sp_help <NOM_TABLE> (ou tout objet comme une vue, proc stock,etc...
exec sp_help '2483dif'
```

Pour info, sp_help est une System Procedure placée dans chaque base de données (hérite de model)

Création de types

Créer un type pour le téléphone

Création de contraintes

```
IF OBJECT_ID('Client', 'U') IS NOT NULL DROP TABLE Client
create table Client(
/*Clé primaire (PRIMARY KEY)
  AutoIncrement de 1 (IDENTITY)*/
IdClient INT PRIMARY KEY IDENTITY

/*Contrainte (CHECK) n'acceptant que certaines valeurs (M, MME,...)*/
,Civilite char(8)
CONSTRAINT [CK Civilite]
CHECK(upper(Civilite)='M' OR upper(Civilite)='MME' OR upper(Civilite)='MLLE' OR
upper(Civilite)='M ET MME')

,Nom varchar(25) NOT NULL
,Prenom varchar(25)
,Ville varchar(30) DEFAULT 'Le Havre'

,EnCoursHT money

/*Valeur devant être unique sans être une clé */
,NoSS VARCHAR(13) UNIQUE

,EnCoursTTC AS (EnCoursHT*(1.196))

,Modif ROWVERSION
```

```
,DatePremiereVisite smalldatetime NOT NULL
,DatePremiereCommande smalldatetime NULL
)
ALTER TABLE Client
WITH CHECK
ADD CONSTRAINT [CK_DateVite]
CHECK([DatePremiereCommande]>=[DatePremiereVisite])
```

Exercice : Créer une table Devis comportant les champs et contraintes suivantes :

Champ	Type	Contrainte
Compteur	Int	N° automatique
Réf Devis	12 caract.	Clé
Date Création	Date	Date du jour par défaut
idClient	Int	Clé étrangère
Validité	Tinyint	
Date échéance	Date	Formule : ([Date Création] + [Validité])
Accord	Bit	
Date Accord	Date	
Mise à jour	timestamp	

Contraintes de tables :

Nom contrainte	Condition
CK_Devis	La date d'accord ne peut excéder la date de création et la validité
PK_Devis	Créer une clé primaire sur le no de devis

Exemple de contrainte de clé étrangère

```
CREATE TABLE MyOrders (OrderID int,CustID int
CONSTRAINT [CleEtrangere]  FOREIGN KEY
REFERENCES MyCustomers(CustID) ON DELETE CASCADE ON UPDATE CASCADE )
```

Ajouter un enregistrement à la table Client

Ajouter un enregistrement à la table Devis

Supprimer un client ayant un devis et constater la suppression en cascade

Le langage SQL – Extraction de données

Clause FROM et WHERE

```
SELECT orderid, empid, orderdate, freight
FROM Sales.Orders
WHERE custid = 71;
```

Clause DISTINCT

Afficher la liste des different couleurs des produits

```
SELECT DISTINCT Color FROM Production.Products
```

Critère TOP

N° et Date des 10 dernières commandes

```
SELECT TOP(10) ORDERID,ORDERDATE FROM SALES.ORDERS ORDER BY ORDERDATE DESC
```

Critère NULL

Afficher la liste des produits qui sont toujours commercialisés (TSQL)

```
SELECT [productid],[productname],[ProductNumber]
  FROM [TSQLFundamentals2008].[Production].[Products]
  WHERE (ProductNumber IS NULL)
```

Critère AND et OR

Afficher les noms des produits rouge et noir

```
SELECT PRODUCTID, NAME FROM Production.Product WHERE COLOR='RED' OR
COLOR='BLACK'
```

Afficher les noms des produits rouge et noir dont le ProductNumber commence par FR

```
SELECT  PRODUCTID, PRODUCTNAME, COLOR, PRODUCTNUMBER FROM  PRODUCTION.PRODUCT
WHERE (COLOR = N'RED' OR COLOR = N'BLACK') AND (PRODUCTNUMBER LIKE N'FR%')
```

Critère BETWEEN

Afficher la liste des commandes passées en 2008

```
SELECT * FROM  Sales.Orders  WHERE orderdate BETWEEN '20080101'AND  '20081231'
```

```
SELECT * FROM  Sales.Orders  WHERE orderdate BETWEEN '01/01/2008'  AND
'31/12/2008'
```

Critère LIKE

Afficher les noms des produits contenant ROAD, mais pas ceux dont le nom commence par ROAD

```
SELECT NAME FROM SALESLT.PRODUCT WHERE NAME LIKE '[^ROAD]%ROAD%'
```

Afficher les productNumber dont le code comporte un chiffre en 4e position

```
SELECT PRODUCTNUMBER FROM SALESLT.PRODUCT
WHERE PRODUCTNUMBER LIKE '___[0-9]%'
```

Afficher les employés dont le nom contient au moins 2 fois la lettre 'A'

```
SELECT empid, firstname, lastname FROM HR.Employees WHERE lastname LIKE '%a%a%';
```

Critère IN

La liste des clients de Mexico, du Canada ou des USA

```
SELECT * FROM Sales.Customers
WHERE Country IN ('Mexico', 'Canada','USA')
```

Liste des produits Rouge ou Jaune (ou Null)

```
SELECT  [productname],[Color],[ProductNumber]
  FROM [TSQLFundamentals2008].[Production].[Products]
  WHERE (Color in ('Black','Yellow') OR Color IS NULL)
```

Créer un champ calculé

Calcule le montant de la vente pour chaque ligne de commande

```
SELECT orderid, productid, qty, unitprice, discount,
  qty * unitprice * (1 - discount) AS val FROM Sales.OrderDetails;
```

Concatène le nom et le prénom des employés

```
SELECT empid, firstname + N' ' + lastname AS fullname
FROM HR.Employees;
```

Utiliser les fonctions dans des requêtes

Afficher les ventes et l'année de la vente

```
SELECT     Name, YEAR(SellStartDate) AS Annee, ListPrice - StandardCost AS Marge
FROM         SalesLT.Product
```

Afficher 'Pas de couleur' si NULL pour le champ Color

```
SELECT [productid]
      ,[productname]
      ,ISNULL([Color],'Pas de couleur')

  FROM [TSQLFundamentals2008].[Production].[Products]
```

Afficher les commandes ayant eu lieu le dernier jour du mois

```
SELECT orderid, orderdate, custid, empid
FROM Sales.Orders
WHERE orderdate = DATEADD(month, DATEDIFF(month, '19991231', orderdate),
'19991231');
```

Concatène plusieurs champs et si valeur nulle, on affiche un espace

```
SELECT custid, country, region, city,
  country + N',' + COALESCE(region, N'') + N',' + city AS location
FROM Sales.Customers;
```

Sépare le nom du prénom avec la fonction CHARINDEX

```
SELECT [custid]
      ,[contactname]
      ,LEFT(contactname,CHARINDEX(',',[contactname])-1)
  FROM [TSQLFundamentals2008].[Sales].[Customers]
```

Afficher le numéro du fournisseur avec 10 chiffres

```
SELECT supplierid,
RIGHT(REPLICATE('0',9) + CAST(supplierid AS VARCHAR(10)),10)
FROM Production.Suppliers
```

Convertir une date de chaîne vers un DateTime

```
SELECT CONVERT(DATETIME,'01/03/2009',103)
```

Formater un affichage de chiffres (2 décimales)

```
CAST(SUM((OD.qty * OD.unitprice) * (1 - OD.discount)) AS NUMERIC(12 , 2))
```

Générer un chiffre aléatoire

```
SELECT ABS(CAST(CAST(NEWID() AS VARBINARY) AS tinyint))
```

Les fonctions de SQL Server

Fonctions d'agrégat

COUNT (*)	Dénombre les lignes sélectionnées
COUNT ([ALL|DISTINCT] exp1)	Dénombre toutes les expressions non nulles ou les expressions non nulles distinctes
COUNT_BIG	Possède le même fonctionnement que la fonction COUNT, simplement, le type de données de sortie est de type bigint au lieu de int
SUM ([ALL|DISTINCT] exp1)	Somme de toutes les expressions non nulles ou des expressions non nulles distinctes
AVG ([ALL|DISTINCT] exp1)	Moyenne de toutes les expressions non nulles ou des expressions non nulles distinctes
MIN (exp1) OU MAX (exp1)	Valeur MIN ou valeur MAX d'exp1
STDEV ([ALL|DISTINCT] exp1)	Ecart type de toutes les valeurs de l'expression donnée
STDEVP ([ALL|DISTINCT] exp1)	Ecart type de la population pour toutes les valeurs de l'expression donnée
VAR ([ALL|DISTINCT] exp1)	Variance de toutes les valeurs de l'expression donnée

Les fonctions mathématiques :

ABS (exp1)	Valeur absolue d'exp1.
CEILING (exp1)	Plus petit entier supérieur ou égal à exp1.
FLOOR (exp1)	Plus grand entier supérieur ou égal à exp1.
SIGN (exp1)	Renvoie 1 si exp1 est positive, -1 si elle est négative, et 0 si elle est égale à 0.
SQRT (exp1)	Racine carrée d'exp1.
POWER (exp1, n)	Exp1 à la puissance n.
SQUARE (exp1)	Calcul du carré d'exp1.

Les fonctions trigonométriques :

PI ()	Valeur de PI.
DEGREES (exp1)	Conversion d'exp1 de radian vers degrés.
RADIANS (exp1)	Conversion d'exp1 de degrés vers radians.
SIN (exp1), COS (exp1), TAN (exp1), COT (exp1)	Sin, cos ou tangente d'exp1.
ACOS (exp1), ASIN (exp1), ATAN (exp1)	Arc cos, arc sin ou arc tan d'exp1.
ATN2 (exp1, exp2)	Angle dont la tangente se trouve dans l'intervalle exp1 et exp2.

Les fonctions logarithmiques :

EXP (exp1)	Exponentielle d'exp1.
LOG (exp1)	Logarithme d'exp1.
LOG10 (exp1)	Logarithme base 10 d'exp1.

Les fonctions de dates :

GETDATE ()	Date et Heure système.
DATENAME (format, exp1)	Renvoie la partie date sous forme de texte.
DATEPART (format, exp1)	Renvoie la valeur de la partie date selon le format donné.
DATEDIFF (format, exp1, exp2)	Différence entre les deux tables selon le format donné.
DATEADD (format, p, exp1)	Ajoute p format à la date exp1.
DAY (exp1)	Retourne le numéro du jour dans le mois.
MONTH (exp1)	Retourne le numéro du mois.
YEAR (exp1)	Retourne l'année.

Les fonctions de chaîne de caractères :

ASCII (exp1)	Valeur du code ASCII du premier caractère d'exp1.
UNICODE (exp1)	Valeur numérique correspondant au code UNICODE d'exp1.
CHAR (exp1)	Caractère correspondant au code ASCII d'exp1.
NCHAR (exp1)	Caractère UNICODE correspondant au code numérique d'exp1.
LTRIM (exp1), RTRIM (exp1)	Supprime les espaces à droit pour RTRIM et à gauche pour LTRIM d'exp1.
STR (exp1, n, p)	Convertit le nombre exp1, en chaine de longueur maximale n dont p caractères seront à droite de la marque décimale.
SPACE (n)	Renvoie n espaces.
REPLICATE (exp1, n)	Renvoie n fois exp1.
CHARINDEX ('masque', exp1) PATINDEX ('%masque%', exp1)	Renvoie la position de départ de la première expression 'masque' dans exp1. PATINDEX permet d'utiliser des caractères génériques et de travailler avec certains type comme TEXT, CHAR ou encore VARCHAR.
LOWER (exp1), UPPER (exp1)	Change la casse. LOWER va convertir exp1 en minuscules et UPPER va convertir exp1 en majuscules.
REVERSE (exp1)	Retourne les caractères d'exp1 dans le sens inverse.
RIGHT (exp1, n)	Renvoie les n caractères les plus à droite d'exp1.
LEFT (exp1, n)	Renvoie les n caractères les plus à gauche d'exp1.
SUBSTRING (exp1, n, p)	Renvoie p caractères d'exp1 à partir de n.
STUFF (exp1, n, p, exp2)	Supprime p caractères d'exp1, à partir de n, puis insère exp2 à la position n.

Les fonctions de chaîne de caractères (suite) :

SOUNDEX (exp1)	Renvoie le code phonétique d'exp1.
DIFFERENCE (exp1, exp2)	Compare les SOUDEX des deux expressions. La valeur,qui peut être renvoyée va de 1 à 4,4, valeur pour laquelle, les deux expressions possèdent la plus grande similitude.
LEN (exp1)	Retourne le nombre de caractères d'exp1.
QUOTENAME (exp1)	Permet de transformer exp1 en identifiant valide pour SQL Server.
REPLACE (exp1, exp2, exp3)	Permet de remplacer dans exp1 toutes les occurrences d'exp2 par exp3.

Les fonctions conversion de types :

CAST (exp1 AS types_données)	Permet de convertir une valeur dans le type spécifié en argument
CONVERT (types_données, exp1, style)	Conversion de l'expression dans le type de données spécifié. Un style peut être spécifié dans le cas d'une conversion date ou heure

Les fonctions diverses :

RAND (exp1)	Nombre aléatoire compris en 0 et 1. Exp1 est la valeur de départ
ROUND (exp1, n)	Arrondis exp1 à n chiffres après la virgule

COALESCE (exp1, exp2…)	Renvoie la première expression non NULL.
COL_LENGTH (nom_table, nom_colonne)	Longueur de la colonne.
COL_NAME (id_table, id_colonne)	Nom de la colonne.
DATALENGTH (exp1)	Longueur en octet de l'expression.
DB_ID (Nom_base)	Numéro d'identification de la base de données.
DB_NAME (id_base)	Nom de la base.
GETANSINULL (nom_base)	Renvoie 1 si l'option 'ANSI NULL DEFAULT' est positionné pour la base.
HOST_ID ()	Numéro d'identification du poste.
HOST_NAME ()	Nom du poste.
IDENT_INCR (nom_table)	Valeur de l'incrémentation définit pour la colonne identité de la table spécifiée.
IDENT_SEED (nom_table)	Valeur initiale définie pour la colonne identité de la table indiquée.
IDENT_CURRENT (nom_table)	Retourne la dernière valeur de type identité utilisé par cette table.
INDEX_COL (nom_table, id_index, id_cle)	Nom de la colonne indexé correspondant à l'index.
ISDATE (exp1)	Renvoie 1 si l'expression de type varchar possède un format date valide.
ISNULL (exp1, valeur)	Renvoie valeur si exp1 est NULL.
ISNUMERIC (exp1)	Renvoie 1 si l'expression de type varchar a un format numérique valide.
NULLIF (exp1, exp2)	Renvoie NULL si exp1 = exp2.
STATS_DATE (id_table, id_index)	Date de la dernière mise à jour de l'index.
SUSER_SID (nom_acces)	Numéro d'identification correspondant au nom_acces.
SUSER_SNAME (id)	Nom d'accès identifié par l'id.
CURRENT_TIMESTAMP	Date et heure système, équivalent à GETDATE ().
SYSTEM_USER	Nom d'accès.
CURRENT_USER, USER, SESSION_USER	Nom de l'utilisateur de la session.
OBJECT_PROPERTY (id, propriété)	Permet de retrouver les propriétés de la base.
ROW_NUMBER	Permet de connaitre le numéro d'une ligne issue d'une partition depuis un jeu de résultats.
RANK	Permet de connaitre le rang d'une ligne issue d'une partition dans une série de résultats.
DENSE_RANK	Fonctionne comme RANK, mais ne s'applique qu'aux lignes de la série de résultat.
KILL	Cette fonction permet de mettre fin à une session utilisateur.
NEWID ()	Permet de gérer une valeur de type UniqueIdentifier.
NEWSEQUENTIALID ()	Permet de gérer la prochaine valeur de type UniqueIdentifier.

Créer une condition avec CASE

Suivant le montant des frais, afficher un libellé

```
SELECT ShipDate,SalesOrderNumber,CustomerID,Subtotal,TaxAmt,TotalDue,
    CASE WHEN Freight<25 THEN 'Soft'
        WHEN Freight between 25 AND 700 THEN 'Normal'
        ELSE 'Hard'
    END FreightType
FROM Sales.Orders
```

Permet de remplacer la fonction IF dans une requête : Afficher un message si une valeur est nulle :

```
select orderid,
'Région' =
     CASE
          WHEN shipregion IS NULL THEN 'Pas de région
          ELSE shipregion
     END
FROM Sales.Orders
```

Remarque : on peut aussi utiliser la fonction ISNULL

```
ISNULL(shipregion,'Pas de région'),
```

Exercice :Afficher le nom du trimestre (Trim1 à 4) suivant le mois de la date de commande

```
SELECT ORDERDATE,
CASE WHEN MONTH(ORDERDATE)<=3 THEN 'TRIM1'
        WHEN MONTH(ORDERDATE)<=6 THEN 'TRIM2'
        WHEN MONTH(ORDERDATE)<=9 THEN 'TRIM3'
        ELSE 'TRIM4'
    END 'TRIMESTRE'
FROM SALES.Orders
```

Le langage SQL - Agrégats

GROUP BY

Nombre de clients par pays

```
SELECT Country,COUNT(*) [Clients par pays] FROM Sales.Customers
GROUP BY Country
```

WHERE

Compter le nombre de commande, la somme du frêt et l'année de la commande pour un client particulier (le n° 71)

```
SELECT empid, YEAR(orderdate) AS orderyear, SUM(freight) AS totalfreight,
  COUNT(*) AS numorders
FROM Sales.Orders WHERE custid = 71 GROUP BY empid, YEAR(orderdate)
```

HAVING

Liste des ventes > 10000 et trié par ventes décroissant

```
SELECT orderid, SUM(qty*unitprice) AS TotalDetailCommande
FROM Sales.OrderDetails
GROUP BY orderid
HAVING SUM(qty*unitprice) > 10000
ORDER BY TotalDetailCommande DESC;
```

Remarque : on peut utiliser le champ TotalDetailCommande pour le tri mais par pour filtrer

Afficher des enregistrements en doublon (ici le nom du produit)

```
SELECT MIN(ProductID) AS Product_ID, LEFT(ProductName,12)
  FROM [TSQLFundamentals2008].[Production].[Products]
GROUP BY LEFT(ProductName,12)
HAVING COUNT(*) > 1
```

Jointures entre tables

Quantité vendue par année de commande

```
SELECT SUM(Qty) AS Qte,YEAR(OrderDate) AS Annee
FROM Sales.Orders AS OH
    JOIN Sales.OrderDetails AS OD
    ON OH.OrderID = OD.OrderID
GROUP BY YEAR(OrderDate)
```

Liste des employés avec le nombre de ventes et le total général

```
SELECT     E.lastname + ' ' + E.firstname AS Salarié,
COUNT(P.productname) AS [Nb Produits]
FROM         Production.Products AS P INNER JOIN
                      Sales.OrderDetails AS OD ON P.productid =
OD.productid INNER JOIN
                      Sales.Orders AS O ON OD.orderid = O.orderid INNER
JOIN
                      HR.Employees AS E ON O.empid = E.empid
GROUP BY E.lastname + ' ' + E.firstname
ORDER BY [Nb Produits]
COMPUTE SUM(COUNT(P.ProductName))
```

Jointures LEFT ou RIGHT

Affiche la liste des clients y compris s'ils n'ont pas passé de commande

```
SELECT C.custid, C.companyname, O.orderid, O.orderdate
FROM Sales.Customers AS C
  LEFT JOIN Sales.Orders AS O
    ON O.custid = C.custid;
```

Jointures > ou <

Affiche des paires uniques d'employés

```
SELECT
  E1.empid, E1.firstname, E1.lastname,
  E2.empid, E2.firstname, E2.lastname
FROM HR.Employees AS E1
  JOIN HR.Employees AS E2
    ON E1.empid < E2.empid;
```

CROSS JOIN

Permet d'effectuer une multiplication matricielle des enregistrements :

Affiche 5 copies de chaque employé en utilisant la table Categories

```
  SELECT E.empid, E.firstname, E.lastname, C.Categoryid FROM HR.Employees
AS E
  CROSS JOIN Production.Categories AS C
WHERE C.Categoryid <= 5
ORDER BY Categoryid, empid;
```

Génère des données aléatoires

```
SELECT TOP 1000000
        SomeNumber = ROW_NUMBER() OVER(ORDER BY (SELECT NULL))*0.001
   FROM sys.all_columns ac1
  CROSS JOIN sys.all_columns ac2
```

Requête UNION

Tous les clients et fournisseurs répertoriés (UNION).

```
SELECT     City, CompanyName, ContactName, 'Customers' AS Relationship
FROM         dbo.Sales.Customers
UNION
SELECT     City, CompanyName, ContactName, 'Suppliers' AS Expr1
FROM         dbo.Production.Suppliers
```

Sous-requêtes imbriquées

Afficher la commande avec le plus grand numéro

```
SELECT orderid, orderdate, empid, custid
FROM Sales.Orders
WHERE orderid = (SELECT MAX(O.orderid)
                 FROM Sales.Orders AS O);
```

Effectuer un filtre sur un champ calculé avec SELECT FROM

```
SELECT *  FROM
(
SELECT OrderID,UnitPrice*qty AS Total FROM Sales.OrderDetails
) SR
WHERE Total>500
```

Afficher les % de chaque vente par client

```
SELECT orderid, custid, val,
  CAST(100. * val / (SELECT SUM(O2.val)
                     FROM Sales.OrderValues AS O2
                     WHERE O2.custid = O1.custid)
       AS NUMERIC(5,2)) AS pct
FROM Sales.OrderValues AS O1
ORDER BY custid, orderid;
```

Afficher les produits dont le prix de vente est supérieur à la moyenne

```
SELECT  ProductName,UnitPrice
FROM Production.Products
WHERE UnitPrice > (SELECT AVG(UnitPrice) FROM Production.Products)
```

Afficher les 3 dernières ventes de chaque client

```
SELECT SO.*
FROM Sales.Orders AS SO
WHERE OrderDate IN
(SELECT TOP 3 OrderDate
FROM
Sales.Orders
WHERE CustID=SO.CustID
ORDER BY CustID, OrderDate DESC)
ORDER BY CustID, OrderDate DESC
```

La liste des supérieurs hiérarchiques des employés

```
SELECT * FROM HR.EMPLOYEES
WHERE EmpID IN (SELECT DISTINCT mgrid FROM HR.EMPLOYEES)
```

Liste des produits n'étant pas utilisé dans les commandes

```
SELECT Productid FROM Production.Products
WHERE productid NOT IN
(
SELECT DISTINCT [productid]
  FROM [TSQLFundamentals2008].[Sales].[OrderDetails]
  )
```

Les CTE

Les CTE (Common Table Expression), appelée aussi expression de table courante ressemblent à une vue temporaire (non persistante).

C'est un ensemble de résultats temporaire et qui sera utilisé par la clause FROM de la requête.

Syntaxe :

```
WITH <nom de votre CTE>(<nom colonne>,…)
AS
(<requête>)
SELECT * FROM <nom de votre CTE>
```

Exemple : Effectuer un filtre sur un champ calculé sans utiliiser de sous-requête

```
WITH CTE AS
(
SELECT OrderID,UnitPrice*Quantity AS Total FROM [Order Details]
)
SELECT * FROM CTE WHERE Total>500
```

Exemple : Crée une table CTE qui permet d'afficher la moyenne générale à partir d'un champ calculé

```
WITH TotalUnitesParCategorie AS
(
SELECT
      dbo.Categories.CategoryID,
      dbo.Categories.CategoryName,
      SUM(dbo.Products.UnitsInStock) AS TotalUnit

FROM         dbo.Categories
      INNER JOIN dbo.Products ON dbo.Categories.CategoryID =
dbo.Products.CategoryID

GROUP BY
      dbo.Categories.CategoryID, dbo.Categories.CategoryName
)
--Affichage de la moyenne générale
SELECT AVG(TotalUnit) FROM TotalUnitesParCategorie
```

Fonctions de classement RANK(), DENSE_RANK(), NTILE(n) et PARTITION BY

ROW_NUMBER()

La fonction de classement de base est ROW_NUMBER().

Elle retourne une colonne qui contient le numéro de la ligne

```
SELECT val, ROW_NUMBER() OVER(ORDER BY val) AS rownum
FROM Sales.OrderValues
```

Exemple : Afficher les 20 lignes à partir de la 10e ligne

```
SELECT TOP 20 * FROM
(
SELECT val, ROW_NUMBER() OVER(ORDER BY val) AS rownum
FROM Sales.OrderValues
) A
WHERE rownum>10
```

RANK() et DENSE_RANK()

Afficher les vendeurs classés par leur nombre de ventes

```
SELECT RANK() OVER (ORDER BY TotCnt DESC) AS TopCustomers, CustID, TotCnt
FROM (SELECT CustID, COUNT(*) AS TotCnt FROM Sales.Orders Group BY CustID) AS
Cust
```

La différence entre RANK et DENSE_RANK est que RANK affiche les valeurs des ex-aequo mais continue la numérotation alors que DENSE_RANK tient compte aussi des valeurs identiques mais reprend la numérotation sans décalage

Exemple

Classement	*RANK()*	*DENSE_RANK()*
12	*1*	*1*
10	*2*	*2*
10	*2*	*2*
10	*2*	*2*
9	*5*	*3*

Problème : si 2 vendeurs ont les mêmes nombre de ventes, la numérotation continue. Pour corriger ce problème, utiliser DENSE_RANK()

DENSE_RANK – FONCTION DE CLASSEMENT

```
SELECT DENSE_RANK() OVER (ORDER BY TotCnt DESC) AS TopCustomers, CustID, TotCnt
FROM (SELECT CustID, COUNT(*) AS TotCnt FROM Sales.Orders Group BY CustID) AS
Cust
```

NTILE()

Divise les lignes retournées en groupes de taille similaire, dont on spécifie le nombre sous la forme d'un paramètre transmis à la fonction.

Exemple : Découpe la liste des commandes en 3 blocs et affiche un texte correspondant

```
SELECT orderid, custid, val
  CASE NTILE(3) OVER(ORDER BY val)
    WHEN 1 THEN 'Low'
    WHEN 2 THEN 'Medium'
    WHEN 3 THEN 'High'
    ELSE 'Unknown'
  END AS titledesc
FROM Sales.OrderValues ORDER BY val;
```

Affiche les noms des produits et leurs quantité vendues par produit.

Utilisation avec une CTE

```
--Création de la CTE
WITH CTE_TotalQte AS (
SELECT Sales.Product.ProductID,SalesLT.Product.Name, SUM(OrderQty) As TotalQte
FROM Sales.OrderDetails
INNER JOIN Sales.Products ON
Sales.Products.ProductID=Sales. OrderDetails.ProductID
GROUP BY Sales.Products.ProductID,Sales.Products.ProductName
)
--Utilisation de la CTE
SELECT *,
 RANK() OVER(ORDER BY TotalQte DESC ) AS Rang,
 DENSE_RANK() OVER(ORDER BY TotalQte DESC) AS DenseRang
NTILE(100) OVER(ORDER BY TotalQte DESC)
FROM CTE_TotalQte
```

Utilisation avec SELECT et SELECT() AS T

```
SELECT *,
 RANK() OVER(ORDER BY TotalQte DESC ) AS Rang,
 DENSE_RANK() OVER(ORDER BY TotalQte DESC) AS DenseRang
FROM
(
SELECT SalesLT.Product.ProductID,SalesLT.Product.Name, SUM(OrderQty) As TotalQte
FROM SalesLT.SalesOrderDetail
INNER JOIN
SalesLT.Product
ON
SalesLT.Product.ProductID
=
SalesLT.SalesOrderDetail.ProductID
GROUP BY
SalesLT.Product.ProductID,SalesLT.Product.Name

) AS CTE_TotalQte
```

PARTITION BY

Partition By permet un classement par groupe (effectue une rupture)

Exemple : Numérote les ventes avec une rupture par client et trié par date décroissant

```
SELECT custid, orderdate, orderid, ROW_NUMBER() OVER(PARTITION BY custid ORDER
BY orderdate DESC, orderid) AS rownum
FROM Sales.Orders
```

Remarque : Puisque chaque vente est numérotée de 1 à N pour chaque client, il est aisé de n'afficher que les N dernières/premières ventes en englobant le SELECT dans un SELECT :

```
SELECT * FROM(
SELECT custid, orderdate, orderid, ROW_NUMBER() OVER(PARTITION BY custid ORDER
BY orderdate DESC, orderid) AS SNO
FROM Sales.Orders
) AS T
WHERE SNO<=3
```

Récapitulatif des commandes

```
SELECT orderid, custid, val,
  ROW_NUMBER() OVER(ORDER BY val) AS rownum,
  RANK()       OVER(ORDER BY val) AS rank,
  DENSE_RANK() OVER(ORDER BY val) AS dense_rank,
  NTILE(10)    OVER(ORDER BY val) AS ntile
FROM Sales.OrderValues
ORDER BY val;
```

Les opérateurs PIVOT et UNPIVOT

Les utilisateurs souhaitent afficher les données sous forme tabulaire, mais les données sont le plus souvent stockées sous une forme relationnelle.

PIVOT est un nouvel opérateur T-SQL que l'on peut mentionner dans une clause FROM pour transformer les lignes en colonnes et créer une requête à tabulation croisée.

Exemple avec une CTE

```
WITH OrderSummary
AS
(

SELECT SUM(Qty) AS Qte,ProductId,YEAR(OrderDate) AS Annee
FROM Sales.Orders AS OH
    JOIN Sales.OrderDetails AS OD ON OH.OrderID = OD.OrderID
GROUP BY ProductId, YEAR(OrderDate)
)
SELECT * FROM OrderSummary PIVOT (SUM(Qte) FOR Annee IN ( [2008], [2009], [2010]
)

) AS pvt
```

Même exemple sans CTE

```
SELECT * FROM
(
SELECT SUM(Qty) AS Qte,ProductId,YEAR(OrderDate) AS Annee
FROM Sales.Orders AS OH
    JOIN Sales.OrderDetails AS OD ON OH.OrderID = OD.OrderID
GROUP BY ProductId, YEAR(OrderDate)
) UP
PIVOT (SUM(Qte) FOR Annee IN ( [2008], [2009], [2010])

) AS pvt
```

Exemple : Comparer ensuite les années pour afficher les ventes ayant progressé sur les 3 dernières années

```
SELECT * FROM
(
SELECT * FROM
(
SELECT SUM(Qty) AS Qte,ProductId,YEAR(OrderDate) AS Annee
FROM Sales.Orders AS OH
    JOIN Sales.OrderDetails AS OD ON OH.OrderID = OD.OrderID

GROUP BY ProductId, YEAR(OrderDate)
) UP
PIVOT (SUM(Qte) FOR Annee IN ( [2008], [2009], [2010])

) AS pvt
) AS SSR WHERE [2009]>[2008] AND [2010]>[2009]
```

Opérations sur les données

Créer une table temporaire

```
USE tempdb;

IF OBJECT_ID('dbo.Customers', 'U') IS NOT NULL DROP TABLE dbo.Customers;

CREATE TABLE dbo.Customers
(
  custid      INT          NOT NULL PRIMARY KEY,
  companyname NVARCHAR(40) NOT NULL,
  country     NVARCHAR(15) NOT NULL,
  region      NVARCHAR(15) NULL,
  city        NVARCHAR(15) NOT NULL
);
GO
```

Méthode INSERT

Insertion de valeurs :

Insérer une valeur dans la table Customers :

```
INSERT INTO dbo.Customers(custid, companyname, country, region, city)
  VALUES(100, N'Company ABCDE', N'USA', N'WA', N'Redmond');
```

Insertion de données

Ajouter à cette table tous les enregistrements de la table Customers qui ont des commandes

```
INSERT INTO dbo.Customers(custid, companyname, country, region, city)
  SELECT custid, companyname, country, region, city
  FROM TSQLFundamentals2008.Sales.Customers AS C
  WHERE EXISTS
    (SELECT * FROM TSQLFundamentals2008.Sales.Orders AS O
     WHERE O.custid = C.custid);
```

Méthode UPDATE

Met à jour la table des Produits en générant des valeurs aléatoires

```
UPDATE [TSQLFundamentals2008].[Production].[Products]
   SET [UnitsInStock] = ABS(CAST(CAST(NEWID() AS VARBINARY) AS int)) % 100
```

Augmente la remise de 5% pour un produit particulier

```
UPDATE dbo.OrderDetails SET discount = discount + 0.05 WHERE productid=51
```

Méthode DELETE

```
ALTER PROCEDURE [dbo].[DeleteProduct]
(@ProductID INT)
AS
DELETE FROM ShoppingCart WHERE ProductID=@ProductID
```

Exemple : On crée 2 tables temporaires Orders et Customers et on efface de la table Orders les commandes pour lesques les Clients sont américains

```
USE tempdb;

IF OBJECT_ID('dbo.Orders','U') IS NOT NULL DROP TABLE dbo.Orders;
IF OBJECT_ID('dbo.Customers','U') IS NOT NULL DROP TABLE dbo.Customers;

SELECT * INTO dbo.Orders FROM TSQLFundamentals2008.Sales.Orders
SELECT * INTO dbo.Customers FROM TSQLFundamentals2008.Sales.Customers

DELETE FROM O FROM dbo.Orders AS O
JOIN dbo.Customers AS C
ON O.Custid=C.Custid

WHERE C.country='USA'
```

Mise en œuvre des fonctions

Fonctions scalaires : Ne retournent qu'une seule valeur

Affiche la moyenne des ventes pour un client passé en paramètre

```
CREATE FUNCTION MoyenneClient(@CustomerID int) RETURNS money
AS
BEGIN
      DECLARE @Montant MONEY
      SET @Montant=0
      SELECT @Montant = AVG(TotalDue)
    FROM Sales. Orders
      WHERE CustID=@CustID
      RETURN ISNULL(@Montant,0)
END
```

Fonction qui affiche une liste séparée par des virgules :

```
ALTER FUNCTION [dbo].[fn_ListeModuleScenario]
(
            @idScenario int
)
RETURNS VarChar(8000)
AS
BEGIN
            Declare @buffer VarChar(8000)
            Select   @buffer = IsNull(@buffer + ',','') + LibelleModule
                        From    dbo.qryScenarioModuleListe
                        Where idScenario = @idScenario
            RETURN @buffer
END
```

Calcule l'âge d'une personne

```
CREATE FUNCTION [dbo].[ufn_GetAge] ( @pDateOfBirth    DATETIME,
                                     @pAsOfDate       DATETIME )
RETURNS INT
AS
BEGIN

    DECLARE @vAge          INT

    IF @pDateOfBirth >= @pAsOfDate
        RETURN 0

    SET @vAge = DATEDIFF(YY, @pDateOfBirth, @pAsOfDate)

    IF MONTH(@pDateOfBirth) > MONTH(@pAsOfDate) OR
      (MONTH(@pDateOfBirth) = MONTH(@pAsOfDate) AND
       DAY(@pDateOfBirth)   > DAY(@pAsOfDate))
        SET @vAge = @vAge - 1

    RETURN @vAge
END
GO
```

Créer une fonction qui retourne une nombre aléatoire suivant une tranche

```
create view v_newid as
select newid() as newid
go

CREATE FUNCTION fctAlea
(
      -- Add the parameters for the function here
      @Tranche int
)
RETURNS int
AS
BEGIN
      -- Declare the return variable here
      DECLARE @Resultat int

    declare @x uniqueidentifier
    select @x = newid from v_newid

      SET @Resultat =ABS(CAST(CAST(@x  AS VARBINARY) AS int))%@Tranche


      RETURN @Resultat

END
```

Fonctions en ligne

Une UDF en ligne est comme une vue paramétrée, et ne peut contenir qu'une seule instruction SELECT

```
CREATE FUNCTION [dbo].[ufnGetCustomerInformation](@CustomerID int)
RETURNS TABLE
AS
RETURN (
    SELECT
        CustID,
        FirstName,
        LastName
    FROM [Sales].[Customer]
    WHERE [CustID] = @CustomerID
);
```

Fonctions tables incluses

Fonction qui crée une table à partir d'une chaîne

```
CREATE FUNCTION [dbo].[CreeTableList] (@list nvarchar(MAX))
   RETURNS @tbl TABLE (number int NOT NULL) AS
BEGIN
   DECLARE @pos        int, @nextpos    int,  @valuelen   int
   SELECT @pos = 0, @nextpos = 1

   WHILE @nextpos > 0
   BEGIN
      SELECT @nextpos = charindex(',', @list, @pos + 1)
      SELECT @valuelen = CASE WHEN @nextpos > 0
                              THEN @nextpos
                              ELSE len(@list) + 1
                         END - @pos - 1
      INSERT @tbl (number)
         VALUES (convert(int, substring(@list, @pos + 1, @valuelen)))
      SELECT @pos = @nextpos
   END
  RETURN
END
```

Appel de cette fonction :

```
UPDATE [dbo].[tblAnneePlan]
   SET [Valide] = @Valide
FROM [dbo].[tblAnneePlan]
 JOIN   CreeTableList (@NoCE) i ON dbo.[tblAnneePlan].CodeCE = i.number
```

Procédures stockées

Convention de nommage des procédures stockées

sp_<TypeAction>_NomProc

Exemple d'utilisation de TOP avec une variable

```
CREATE PROCEDURE usp_SEL_ReturnTopOrders(@NbLignes bigint)
AS
BEGIN
      SELECT TOP (@NbLignes) SalesOrderID
      FROM Sales. Orders
      ORDER BY SalesOrderID
END
```

Exemple TSQL d'ajout d'un enregistrement si l'enregistrement n'existe pas :

```
IF EXISTS(SELECT * FROM Table1 WHERE Id = @id)
BEGIN
   UPDATE Table1 SET Column1 = @newValue WHERE Id = @id
END
ELSE
BEGIN
   INSERT INTO Table1 (Id, Column1) VALUES (@id, @newValue)
END
```

- Insére une nouvelle commande, récupère le no de la commande pour insérer une ligne de commande et affiche le no de la commande

```
DECLARE @Ident int
INSERT INTO Sales.Orders (CustID,OrderDate)
VALUES (@NoClient, GETDATE())

SELECT @Ident = @@IDENTITY

INSERT INTO Sales.[OrderDetails] (OrderID, ProductID, UnitPrice, Quantity)
VALUES (@Ident,@NoProduit, @Prix, @Qte)
SELECT 'La commande porte le n° ' + CONVERT(varchar(8),@Ident)
```

- On améliore la procédure en ajoutant une transaction qui vérifie que le stock est suffisant. Si ce n'est pas le cas, on annule toute la transaction

```
…(on place ici le code vu précédemment)
IF(SELECT UNITSINSTOCK FROM PRODUCTS WHERE ProductID=@NoProduit)> @Qte
      UPDATE PRODUCTS SET UNITSINSTOCK=UNITSINSTOCK-@Qte
      WHERE PRODUCTID= NoProduit
      PRINT 'Mise à jour effectuée'
ELSE
      PRINT 'Pas assez de stock !'
      END
```

- Supprime un client qui n'a pas de commandes, si le client n'existe pas, on affiche un message

```
alter PROCEDURE EffaceClient @NoClient int
AS
BEGIN
IF NOT EXISTS(SELECT * FROM Sales.Customers WHERE CustID=@NoClient)
      PRINT '** Ce client n''existe pas !!!'
ELSE
BEGIN
      IF EXISTS(SELECT * FROM ORDERS
            WHERE custID=@NoClient)
            PRINT '** Ce client ne peut être supprimé'
      ELSE
            BEGIN
                  DELETE Sales.customers WHERE custid=@NoClient
                  PRINT '*** Ce client a été supprimé ***'
            END
      END
END
```

Procédure stockée : Simuler l'affichage d'une page Web, en précisant le nombre d'enregistrements par page, ains que la page désirée

```
      Set rowcount @NbElements

      SELECT  * FROM
  (SELECT ROW NUMBER() OVER(ORDER BY ProductID) AS RN,* FROM
    [TSQLFundamentals2008].[Production].[Products]) A WHERE
RN>@NoPage*@NbElements

    set rowcount 0
```

Déclencheurs

Si l'on modifie la quantité du detail d'une commande, diminuer le stock du produit correspondant

```
CREATE TRIGGER upd_Quantity ON Sales.[OrderDetails]
AFTER UPDATE NOT FOR REPLICATION
AS
BEGIN
UPDATE Production.Products SET
UnitsInStock=UnitsInStock-inserted.Quantity
FROM inserted
WHERE inserted.ProductID=Products.ProductID
END
GO
```

Déclenche une erreur si on tente de supprimer plusieurs lignes d'employés

```
CREATE TRIGGER [Empl_Delete] ON [dbo].[Employees]
FOR DELETE
AS
IF (SELECT COUNT(*) FROM Deleted)>1
BEGIN
      RAISERROR('Vous ne pouvez pas supprimer plus d''un employé à la
fois.',16,1)
ROLLBACK TRANSACTION
END
```

A la suppression d'une catégorie, on place la valeur 1 sur le champ DISCONTINUED

```
CREATE TRIGGER [delCategory] ON Production.[Categories]
INSTEAD OF DELETE AS
BEGIN
 UPDATE C SET [discontinued] = 1
  FROM Production.Categories C
  INNER JOIN deleted
  ON C.[CategoryID]=deleted.CategoryID
END
```

Calcule le total de la commande lorsque l'on modifie un enr

```
CREATE TRIGGER [tg_CalculeCommande] ON [Sales].[OrderDetails]
AFTER INSERT, DELETE, UPDATE AS
BEGIN
    DECLARE @Count int;

    SET @Count = @@ROWCOUNT;
    IF @Count = 0
        RETURN;

    SET NOCOUNT ON;

    BEGIN TRY
        -- If inserting or updating these columns
        IF UPDATE([ProductID]) OR UPDATE([Qty]) OR UPDATE([UnitPrice]) OR
UPDATE([Discount])

        UPDATE [Sales].Orders
        SET [Sales].Orders.[SubTotal] =
            (SELECT SUM(qty*unitprice*(1-discount))
                FROM [Sales].[OrderDetails]
                WHERE [Sales].Orders.[OrderID] =
[Sales].[OrderDetails].[OrderID])
        WHERE [Sales].Orders.[OrderID] IN (SELECT inserted.[OrderID] FROM
inserted);
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
        BEGIN
            ROLLBACK TRANSACTION;
        END

    END CATCH;
   END;
```

Fonctions utiles

Générer une table temporaire de 10 000 valeurs

```
DECLARE @NUMBERS TABLE(NUMBER INT)

INSERT INTO @NUMBERS(NUMBER)

SELECT NUMBER FROM

(
SELECT ROW_NUMBER() OVER (ORDER BY S1.NAME) AS NUMBER
FROM SYSOBJECTS S1 CROSS JOIN SYSOBJECTS S2
) AS NUMBERS

WHERE NUMBER BETWEEN 1 AND 10000
```

Nom de l'utilisateur en cours

```
SELECT 'L''utilisateur courant est : ' + SUSER_NAME()
```

Nb d'utilisateurs connectés à la base en cours

```
SELECT COUNT(*)
FROM master..sysprocesses
WHERE dbid=db_id()
```

Restaurer une base avec les fichiers MDF et LDF

```
EXEC sp_attach_db @dbname ='AdventureWorks',
@filename1 = 'D:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\AdventureWorks_Data.mdf',
@filename2 = 'D:\Program Files\Microsoft SQL
Server\MSSQL.1\MSSQL\Data\AdventureWorks_Log.ldf'
```

Vérifier l'existence d'une table :

```
IF EXISTS(SELECT name FROM sys.tables WHERE name = 'tblExportHRAccess')
DROP TABLE tblExportHRAccess
```

*Sélectionner un objet et **ALT + F1** pour en connaître la structure*

Liste des tables d'une base de données :

```
use XXX
SELECT * FROM information_schema.tables WHERE table_type='BASE TABLE'
```

Affiche les noms et tables et le nombre de lignes :

```
SELECT O.name AS Table_Name, SUM(I.rows) AS Rows_Count
FROM   sys.sysobjects AS O INNER JOIN sys.sysindexes AS I ON O.id = I.id
WHERE (O.xtype = 'U') GROUP BY O.name
```

Liste des bases de données du serveur :

```
EXEC sp_databases
```

Afficher le contenu TSQL d'une procédure stockée :

```
use XXX ; exec sp_helptext 'NomProcStock'
```

Nb d'utilisateurs connectés à la base en cours :

```
SELECT COUNT(*) FROM master..sysprocesses WHERE dbid=db_id()'
```

Taille occupée par une base de données

```
USE DB_ESSAI
GO
EXEC sp_spaceused
GO
```

Taille occupée par une table

```
USE pubs
EXEC sp_spaceused 'titles'
```

Rendre propriétaire l'utilisateur "sa" pour la prise en charge des schémas

```
ALTER AUTHORIZATION ON DATABASE::AdventureWorksLT TO sa
```

Compacter le fichier .LDF

```
DBCC SHRINKFILE (TSQLFundamentals2008,20)

BACKUP LOG TSQLFundamentals2008 WITH TRUNCATE_ONLY
```

Infos sur le serveur

```
SELECT SERVERPROPERTY('ProductVersion') AS ProductVersion
,SERVERPROPERTY('ProductLevel') AS ProductLevel
,SERVERPROPERTY('Edition') AS Edition
,SERVERPROPERTY('EngineEdition') AS EngineEdition
,SERVERPROPERTY('InstanceName') AS InstanceName
,SERVERPROPERTY('IsClustered') AS IsClustered
,SERVERPROPERTY('IsIntegratedSecurityOnly') AS IsIntegratedSecurityOnly
,SERVERPROPERTY('IsSingleUser') AS IsSingleUser
,SERVERPROPERTY('LicenseType') AS LicenseType
,SERVERPROPERTY('MachineName') AS MachineName
,SERVERPROPERTY('NumLicenses') AS NumLicenses
,SERVERPROPERTY('ServerName') AS ServerName
,SERVERPROPERTY('SqlCharSetName') AS SqlCharSetName
,SERVERPROPERTY('IsFullTextInstalled') AS IsFullTextInstalled
,SERVERPROPERTY('Collation') AS Collation
```

Création d'une table et chargement d'un million de lignes

```
CREATE TABLE dbo.T_TEST
(
  id int identity(1,1),
  val varchar(10),
  creation_date datetime
)

--> Chargement d'1 million de lignes dans la table
DECLARE @counter int;
SET @counter = 1;
WHILE @counter <= 1000000
BEGIN
INSERT INTO T_TEST(val,creation_date)
VALUES(convert(varchar(10),(LEFT(convert(bigint,RAND()*10000000),6))),getdate())
;
SET @counter = @counter + 1
END;
```

Structure WHILE

```
DECLARE @i AS INT;
SET @i = 1;
WHILE @i <= 10
BEGIN
  PRINT @i;
  SET @i = @i + 1;
END;
GO
```

Exemple :

```
declare @n tinyint
SET @n=5
IF(@n BETWEEN 4 AND 6)
BEGIN
      WHILE(@n>0)
      BEGIN
            SELECT @n AS 'Nombre'
            ,CASE WHEN (@n % 2)=1
                  THEN 'IMPAIR'
                  ELSE 'PAIR'
            END
      SET @n = @n - 1
      END
END
ELSE
      PRINT 'Analyse Impossible'
```

Etude de cas

Ajouter des enregistrements à une table DESTINATION si des enregistrements de la table SOURCE sont modifiés

Créer 2 tables

```
IF OBJECT_ID('TableSource', 'U') IS NOT NULL DROP TABLE TableSource
CREATE TABLE TableSource
(
    Id int identity(1,1) PRIMARY KEY,
    Nom varchar(50),
    Suivi timestamp
)

IF OBJECT_ID('TableDestination', 'U') IS NOT NULL DROP TABLE TableDestination
CREATE TABLE TableDestination
(
    Id int identity(1,1) PRIMARY KEY,
    AncId int,
    Nom varchar(50),
    Suivi binary(8)
)
```

Insérer des données dans la table source

```
INSERT INTO TableSource(Nom) VALUES('A')
INSERT INTO TableSource(Nom) VALUES('B')
INSERT INTO TableSource(Nom) VALUES('C')
INSERT INTO TableSource(Nom) VALUES('D')
```

On insère les enr de la table SOURCE vers DESTINATION si les enregistrements sont différents

```
INSERT INTO TableDestination(AncId, Nom, Suivi)
SELECT SRC.Id, SRC.Nom, SRC.Suivi
FROM TableSource AS SRC
WHERE NOT EXISTS(SELECT * FROM TableDestination AS DEST WHERE SRC.Id =
DEST.AncId AND
    SRC.Suivi = DEST.Suivi)
```

On modifie les données de la table SOURCE

```
UPDATE TableSource
SET Nom = 'XXX'
WHERE Id = 3
```

Les données du champ timestamp sont modifiés

OBJECTIF : PERMETTRE LA NUMEROTATION AUTOMATIQUE DE N° DE COMMANDE SOUS LA FORME AA 00000 OU AA DESIGNE L'ANNEE EN COURS ET 00000 LA NUMEROTATION PAR ANNEE.

Techniques utilisées

Fonctions UDF, Déclencheurs

Procédure :

- ✓ Créer la table des commandes
- ✓ Créer la table de référence des n° de commande
- ✓ Créer la fonction retournant le prochain numéro de commande suivant la date de la commande
- ✓ Mise en place du déclencheur pour créer le numéro automatique lors de la création d'une nouvelle commande

Créer la table des commandes

Descriptif : Table avec un identifiant pour le n° de commande, ainsi que la date de commande qui permettra d'affecter le bon numéro de commande

Script de création de la table :

```
CREATE TABLE [dbo].[Commande](
	[NoCommande] [char](10) NOT NULL,
	[DateCommande] [smalldatetime] NULL,
	[MontantCommande] [smallmoney] NULL,
 CONSTRAINT [PK_Commande] PRIMARY KEY CLUSTERED
(
	[NoCommande] ASC
)
```

Créer la table de référence des n° de commande

```
CREATE TABLE [dbo].[NoCommande](
	[NoCommande] [char](10) NULL,
	[AnneeCommande] [smallint] NULL)
```

Créer la fonction retournant le prochain numéro de commande suivant la date de la commande

```
ALTER FUNCTION [dbo].[fn_NoCommande]( @Annee smallint)
RETURNS char(10)
AS
BEGIN
      -- Declare the return variable here
      DECLARE @Resultat char(10)

      IF EXISTS(SELECT NoCommande FROM NoCommande WHERE
AnneeCommande=@Annee)
            SELECT @Resultat=NoCommande+1 FROM NoCommande WHERE
AnneeCommande=@Annee
      ELSE
            SET @Resultat=1

      SET @Resultat=REPLICATE('0',5 -LEN(@Resultat)) + @Resultat

      -- Return the result of the function
      RETURN   RIGHT(@Annee,2) + ' ' +  @Resultat

END
```

Mise en place du déclencheur pour créer le numéro automatique lors de la création d'une nouvelle commande

```
CREATE TRIGGER [dbo].[tr_NoCommande]  ON  [dbo].[Commande]
  INSTEAD OF INSERT
AS
BEGIN
    DECLARE @DateComm DATETIME
    SET @DateComm=(SELECT DateCommande FROM inserted)
  INSERT INTO [dbo].[Commande]
           ([NoCommande]
           ,[DateCommande]
           ,[MontantCommande])
    SELECT
    dbo.fn_NoCommande(YEAR(DateCommande)),
     inserted.DateCommande,inserted.MontantCommande FROM inserted

      IF EXISTS(SELECT * FROM NoCommande WHERE
AnneeCommande=YEAR(@DateComm))
            BEGIN
            UPDATE NoCommande SET [NoCommande]
=dbo.fn_NoCommande(YEAR(@DateComm)) WHERE AnneeCommande=YEAR(@DateComm)
            END
      ELSE
            BEGIN
            INSERT INTO [NoCommande] ([NoCommande],[AnneeCommande]) VALUES
(dbo.fn_NoCommande(YEAR(@DateComm)),YEAR(@DateComm))
         END

END
```

TP :

```
USE SalarieFormation
GO
IF OBJECT_ID('T_COUTFORMATION','U') IS NOT NULL
DROP TABLE T_COUTFORMATION
GO
SELECT * INTO T_COUTFORMATION
 FROM
(
SELECT * FROM
(
SELECT     tblSociete.LibelleSociete, tblBesoinsSalaries.AnneePlan,
SUM((tblBesoinsSalaries.NbHeuresTravail09 +
tblBesoinsSalaries.NbHeuresTravail05)
* tblSalariesPlan.TauxHoraire) AS CoutFormation
FROM         tblBesoinsSalaries INNER JOIN
tblEtablissement ON tblBesoinsSalaries.CodeEts = tblEtablissement.CodeEts
INNER JOIN
tblSociete ON tblEtablissement.CodeSociete = tblSociete.CodeSociete INNER
JOIN
tblSalariesPlan ON tblBesoinsSalaries.Matricule =
tblSalariesPlan.Matricule AND tblBesoinsSalaries.AnneePlan =
tblSalariesPlan.AnneePlan AND
tblBesoinsSalaries.CodeEts = tblSalariesPlan.CodeEts
GROUP BY tblSociete.LibelleSociete, tblBesoinsSalaries.AnneePlan
) AS TBL
PIVOT (SUM(CoutFormation) FOR AnneePlan IN ([2009],[2010],[2011])) AS PVT
) AS Export

UPDATE [SalarieFormation].[dbo].[T_COUTFORMATION]
   SET [2009] = ISNULL([2009],0)
      ,[2010] =  ISNULL([2010],0)
      ,[2011] =  ISNULL([2011],0)
GO
```