UNIVERSITÉ POLYTECHNIQUE DE BINGERVILLE

DÉPARTEMENT - INFORMATIQUE

ADMINISTRATION ET SÉCURITÉ DES BASES DE DONNÉES

Triggers, Stored Procedure (A3)

Automatisation (A3)

Requêtes Paramétrées (A4)

Année Scolaire 2022-2023

Par

Professeur: **Robert Yavo**

Email: ryavo@hotmail.com WhatsApp: +225-07-88-63-26-58

Table des matières

1. Triggers & Stored Procedure

1.1 Triggers (Déclencheurs)

1.1.1 Définitions

Un Trigger (Déclencheur) est un ensemble d'instructions SQL qui résident dans la mémoire système avec un nom unique qui est appelée automatiquement lorsqu'un événement se produit dans une base de données.

1.1.2 Les Types de Triggers et leurs syntaxes

- DDL Triggers (Pour les évènements CREATE, ALTER, DROP)
- DML Triggers (Pour les évènements INSERT, UPDATE, DELETE (After et Instead Of)
- Logon Triggers (

Syntaxes:

-DDL Triggers

```
/* Trigger on a CREATE, ALTER, DROP, GRANT, DENY, REVOKE statement */

CREATE [ OR ALTER ] TRIGGER trigger_name
ON { ALL SERVER | DATABASE }
[ WITH <ddl_trigger_option> [ ,...n ] ]
{ FOR | AFTER } { event_type | event_group } [ ,...n ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier >
[ ; ] }
```

- DML Triggers

```
/* -- Trigger on an INSERT, UPDATE, or DELETE statement to a table or
view (DML Trigger)  */

CREATE [ OR ALTER ] TRIGGER [ schema_name . ]trigger_name
ON { table | view }
[ WITH <dml_trigger_option> [ ,...n ] ]
{ FOR | AFTER | INSTEAD OF }
{ [ INSERT ] [ , ] [ UPDATE ] [ , ] [ DELETE ] }
[ WITH APPEND ]
[ NOT FOR REPLICATION ]
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME <method specifier [ ;
] > }
```

-Logon Triggers

```
-- Trigger on a LOGON event (Logon Trigger)
CREATE [ OR ALTER ] TRIGGER trigger_name
ON ALL SERVER
[ WITH <logon_trigger_option> [ ,...n ] ]
{ FOR| AFTER } LOGON
AS { sql_statement  [ ; ] [ ,...n ] | EXTERNAL NAME < method specifier >
[ ; ] }
```

Les « Logon Triggers » sont utilisés pour auditer et contrôler les sessions du serveur, par exemple en suivant les activités de connexion, en limitant les connexions au Serveur SQL ou en limitant le nombre de sessions pour une connexion spécifique.

Pour supprimer les Triggers :

DROP TRIGGER IF EXISTS *trigger_name1*, *trigger_name2*, …., *last_trigger_name*
ON DATABASE | ALL SERVER;

Pour plus de details:
https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15
https://learn.microsoft.com/en-us/sql/relational-databases/triggers/logon-triggers?view=sql-server-ver15

1.1.3 Exemples

Suivre les exemples avec le Prof.

Nous allons créer un autre déclencheur pour stocker les enregistrements de transaction de chaque opération de suppression sur la table Employee dans la table **Employee_Audit_Test.** Nous pouvons créer le déclencheur de suppression à l'aide de l'instruction ci-dessous :

1.1.4 Travail Dirigé

Voir le document : **00_TravailDirigé-02.pdf**

1.2 Stored Procedure (Les Procédures stockées)

1.2.1 Définition et syntaxe

Une procédure stockée est un segment (ensemble) d'instructions SQL stockées dans le catalogue de la base de données. Il s'agit d'un code SQL préparé, enregistré, qui peut être réutilisé encore et encore. Ainsi, il peut être invoqué par des déclencheurs, par d'autres procédures stockées ou par des langages de programmation/scripts tels que Java, Python, PHP, VB.Net, C#, PowerShell, Windows batch, etc. On a 2 types de procédure stockée (user-defined et system stored procedure). La procédure stockée :

- Accepte les paramètres d'entrée (input) et renvoie plusieurs valeurs sous la forme de paramètres de sortie (output) au programme appelant.
- Contient des instructions de programmation qui effectuent des opérations dans la base de données. Celles-ci incluent l'appel d'autres procédures.
- Renvoie une valeur d'état (status) à un programme appelant pour indiquer le succès ou l'échec (et la raison de l'échec).

Avantages

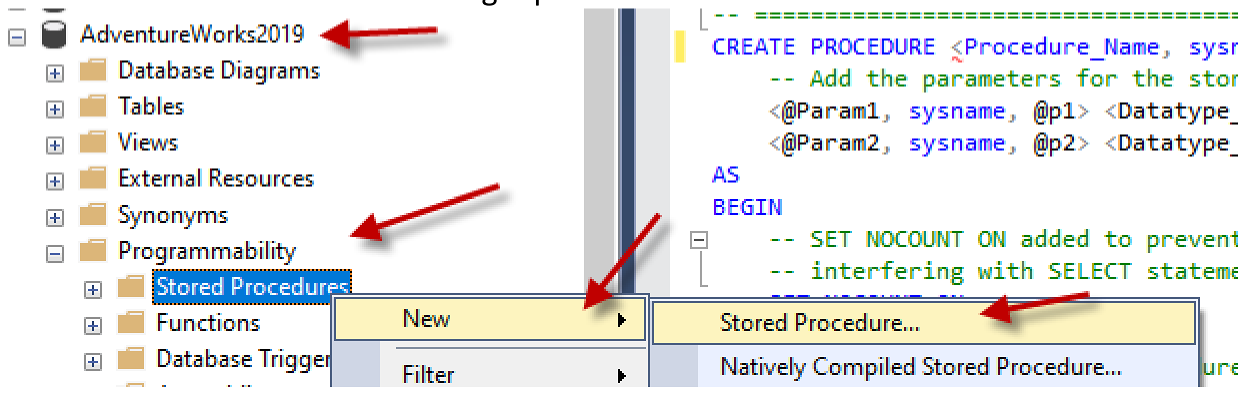
- Trafic réduit
- Sécurité renforcée
- Réutilisable
- Maintenance facile
- Performance Améliorée

Syntaxe

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }
    [schema_name.] procedure_name [ ; number ]
    [ { @parameter_name [ type_schema_name. ] data_type }
        [ VARYING ] [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]

<procedure_option> ::=
    [ ENCRYPTION ]
    [ RECOMPILE ]
    [ EXECUTE AS Clause ]
```

1.2.2 Création de procédures stockées

Dans SSMS, sélectionnez la base de données, naviguez dans Programmability, puis sélectionnez Stored Procedures avec le bouton droit de la souris puis choisir New=>Stored Procedure comme le montre l'image qui suit :



```
-- ==================================
CREATE PROCEDURE <Procedure_Name, sysr
    -- Add the parameters for the stor
    <@Param1, sysname, @p1> <Datatype_
    <@Param2, sysname, @p2> <Datatype_
AS
BEGIN
    -- SET NOCOUNT ON added to prevent
    -- interfering with SELECT stateme
```

/*Norme ISO pour l'Expression Booléenne*/

SET ANSI_NULLS ON

https://learn.microsoft.com/en-us/sql/t-sql/statements/set-ansi-nulls-transact-sql?view=sql-server-ver16

Boolean Expression	SET ANSI_NULLS ON	SET ANSI_NULLS OFF
NULL = NULL	UNKNOWN	TRUE
1 = NULL	UNKNOWN	FALSE
NULL <> NULL	UNKNOWN	FALSE
1 <> NULL	UNKNOWN	TRUE
NULL > NULL	UNKNOWN	UNKNOWN
1 > NULL	UNKNOWN	UNKNOWN
NULL IS NULL	TRUE	TRUE
1 IS NULL	FALSE	FALSE
NULL IS NOT NULL	FALSE	FALSE
1 IS NOT NULL	TRUE	TRUE

/* Norme ISO pour les guillemets*/

```
SET QUOTED_IDENTIFIER ON
GO
-- Create statement succeeds.
CREATE TABLE "select" (
   "identity" INT IDENTITY NOT NULL,
   "order" INT NOT NULL
);
GO
```

```
SET QUOTED_IDENTIFIER OFF
GO
-- Create statement fails.
CREATE TABLE "select" (
   "identity" INT IDENTITY NOT NULL,
   "order" INT NOT NULL
);
GO
```

Exemple

```
USE AdventureWorks2012;
GO
CREATE PROCEDURE HumanResources.uspGetEmployeesTest2
    @LastName nvarchar(50),
    @FirstName nvarchar(50)
AS
   BEGIN
      SET NOCOUNT ON;
      SELECT FirstName, LastName, Department, EndDate
      FROM HumanResources.vEmployeeDepartmentHistory
      WHERE FirstName = @FirstName AND LastName = @LastName
      AND EndDate IS NULL;
   END
GO
```

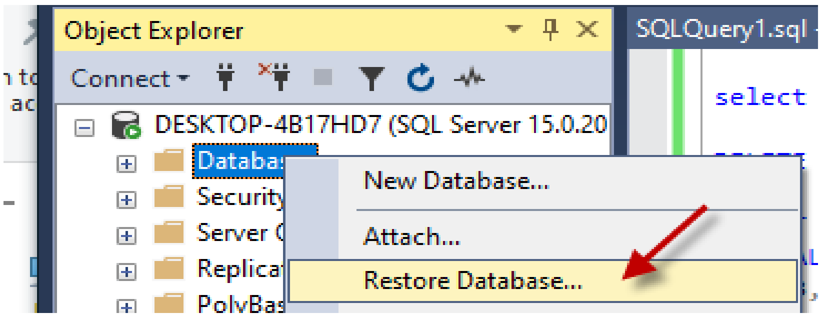
Pour plus de details: https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql?view=sql-server-ver15

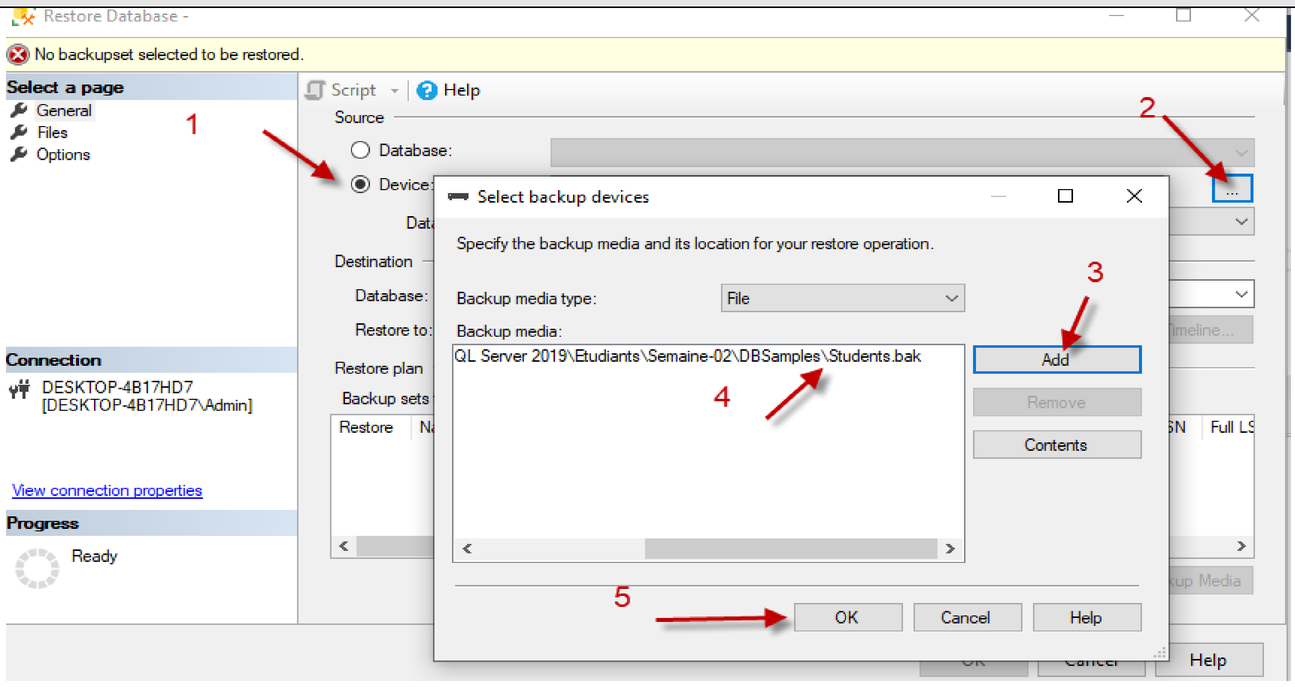
Téléchargez les fichiers de base de données suivantes :

- AdventureWorks2012
- AdventureWorks2016
- AdventureWorks2019

A partir du lien suivant : https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms

Ajoutez-les dans SSMS en suivant les étapes vues la dernière fois.




Restore Database -

No backupset selected to be restored.

Select a page: General, Files, Options

Source — Database: / Device:

Destination — Database: / Restore to:

Restore plan — Backup sets

Connection: DESKTOP-4B17HD7 [DESKTOP-4B17HD7\Admin]

View connection properties

Progress: Ready

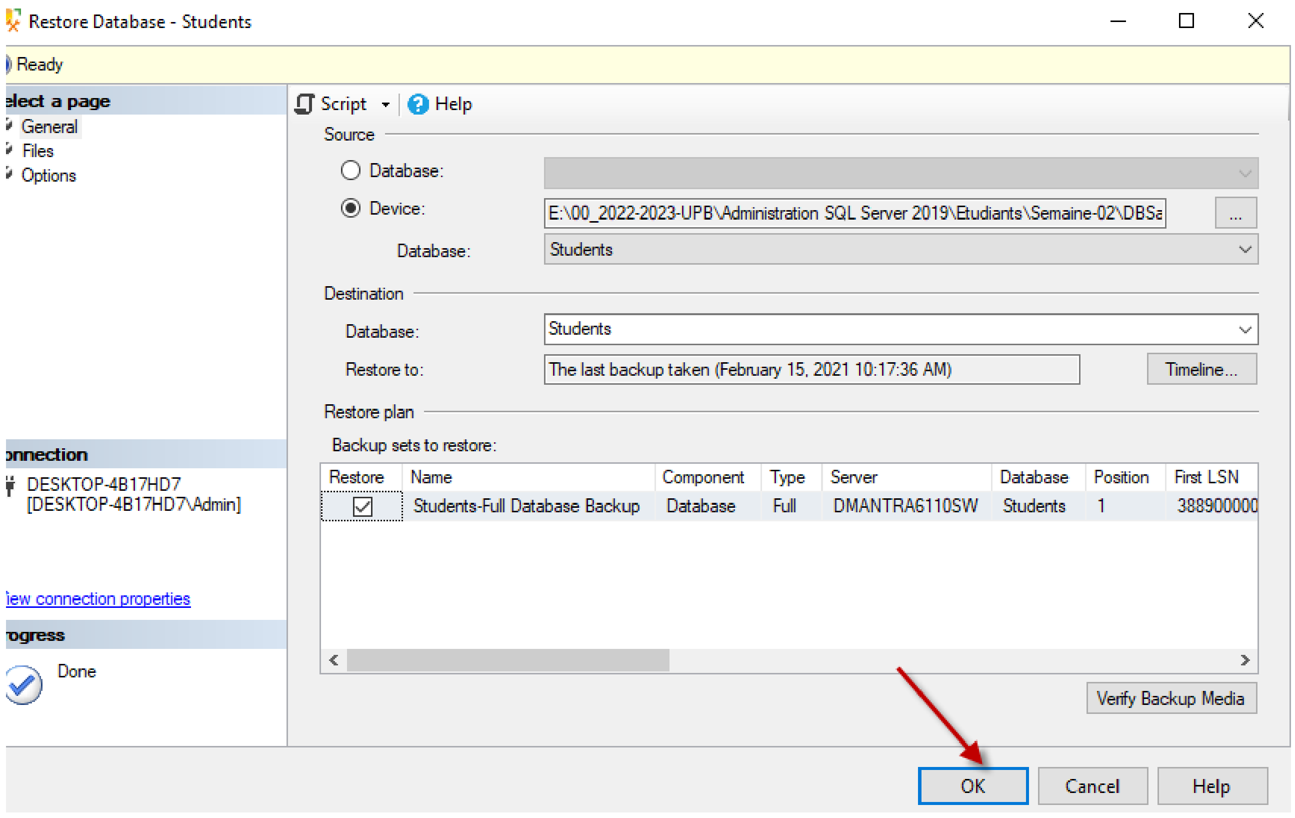
Select backup devices

Specify the backup media and its location for your restore operation.

Backup media type: File

Backup media:

QL Server 2019\Etudiants\Semaine-02\DBSamples\Students.bak


Restore Database - Students

Ready

Source

Database:

Device: E:\00_2022-2023-UPB\Administration SQL Server 2019\Etudiants\Semaine-02\DBSa

Database: Students

Destination

Database: Students

Restore to: The last backup taken (February 15, 2021 10:17:36 AM)

Restore plan

Backup sets to restore:

Restore	Name	Component	Type	Server	Database	Position	First LSN
☑	Students-Full Database Backup	Database	Full	DMANTRA6110SW	Students	1	388900000

Connection: DESKTOP-4B17HD7 [DESKTOP-4B17HD7\Admin]

Progress: Done

Comment exécuter une procédure :

EXEC procedure_name; ou bien **EXECUTE** procedure_name;

Exemple:

EXEC dbo.getEmployeeDetails 'California'; ou bien

EXEC dbo.getEmployeeDetails @States = 'New York';

2. Requêtes Paramétrées (A4)

2.1 Procédures paramétrées

2.1.1 Procédure avec les Paramètres INPUT

Créer une procédure qui reçoit en paramètres le Nom de Famille et le Prénom d'un employé et permet de lister (afficher) son Titre et son Département.

```
IF OBJECT_ID ('HumanResources.uspGetEmployees', 'P') IS NOT NULL
    DROP PROCEDURE HumanResources.uspGetEmployees;
GO
CREATE PROCEDURE HumanResources.uspGetEmployees
    @LastName NVARCHAR(50),
    @FirstName NVARCHAR(50)
AS
BEGIN
    SET NOCOUNT ON;
    SELECT FirstName, LastName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment
    WHERE FirstName = @FirstName AND LastName = @LastName;
END
GO
```

Pour exécuter cette procédure paramétrée :

```
EXECUTE HumanResources.uspGetEmployees N'Ackerman', N'Pilar';
-- Or
EXEC HumanResources.uspGetEmployees @LastName = N'Ackerman', @FirstName =
N'Pilar';
```

Créer une autre procédure mais générique qui fait la même chose sauf que le Nom de famille commence par une lettre et peu importe le Prénom ou vise-versa.

```
IF OBJECT_ID ( 'HumanResources.uspGetEmployees2', 'P' ) IS NOT NULL
    DROP PROCEDURE HumanResources.uspGetEmployees2;
GO
CREATE PROCEDURE HumanResources.uspGetEmployees2
    @LastName NVARCHAR(50) = N'D%',
    @FirstName NVARCHAR(50) = N'%'
AS
    SET NOCOUNT ON;
    SELECT FirstName, LastName, JobTitle, Department
    FROM HumanResources.vEmployeeDepartment
    WHERE FirstName LIKE @FirstName AND LastName LIKE @LastName;
```

EXECUTE HumanResources.uspGetEmployees2 N'H%', N'S%';
EXECUTE HumanResources.uspGetEmployees2 N'Hesse', N'Stefen';

2.1.2 Procédure avec les Paramètres OUTPUT

Les paramètres OUTPUT permettent à une procédure externe, à un lot (batch) ou à plusieurs instructions Transact-SQL d'accéder à une valeur définie lors de l'exécution de la procédure.
Exemple : Créer une procédure paramétrée appelée uspGetList avec 3 paramètres pour obtenir la liste des produits dont les prix ne dépassent pas un montant spécifié.

```sql
/* Output Parameters*/
IF OBJECT_ID ( 'Production.uspGetList', 'P' ) IS NOT NULL
    DROP PROCEDURE Production.uspGetList;
GO
CREATE PROCEDURE Production.uspGetList @Product VARCHAR(40)
    , @MaxPrice MONEY
    , @ComparePrice MONEY OUTPUT
    , @ListPrice MONEY OUT
AS
    SET NOCOUNT ON;
    SELECT p.[Name] AS Product, p.ListPrice AS 'List Price'
    FROM Production.Product AS p
    JOIN Production.ProductSubcategory AS s
      ON p.ProductSubcategoryID = s.ProductSubcategoryID
    WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice;
-- Populate the output variable @ListPprice.
SET @ListPrice = (SELECT MAX(p.ListPrice)
    FROM Production.Product AS p
    JOIN Production.ProductSubcategory AS s
      ON p.ProductSubcategoryID = s.ProductSubcategoryID
    WHERE s.[Name] LIKE @Product AND p.ListPrice < @MaxPrice);
-- Populate the output variable @compareprice.
SET @ComparePrice = @MaxPrice;
GO
```

Exécutez uspGetList pour renvoyer une liste de produits Adventure Works (bikes) qui coûtent moins de 700 $. Les paramètres OUTPUT @Cost et @ComparePrice sont utilisés avec le langage de contrôle de flux pour renvoyer un message dans la fenêtre Messages.

```sql
DECLARE @ComparePrice MONEY, @Cost MONEY;
EXECUTE Production.uspGetList '%Bikes%', 700,
    @ComparePrice OUT,
    @Cost OUTPUT
IF @Cost <= @ComparePrice
BEGIN
    PRINT 'These products can be purchased for less than
    $'+RTRIM(CAST(@ComparePrice AS VARCHAR(20)))+'.'
```

```
END
ELSE
    PRINT 'The prices for all products in this category exceed
    $'+ RTRIM(CAST(@ComparePrice AS VARCHAR(20)))+'.';
```

Résultat de la procedure exécutée:

```
Output

Product                     List Price
--------------------------  ----------
Road-750 Black, 58          539.99
Mountain-500 Silver, 40     564.99
Mountain-500 Silver, 42     564.99
...
Road-750 Black, 48          539.99
Road-750 Black, 52          539.99

(14 row(s) affected)

These items can be purchased for less than $700.00.
```

2.1.3 Procédure avec les paramètres Table

L'exemple suivant utilise un type de paramètre table pour insérer plusieurs lignes dans une table. L'exemple crée le type de paramètre, déclare une variable de table pour y faire référence, remplit la liste de paramètres, puis transmet les valeurs à une procédure stockée. La procédure stockée utilise les valeurs pour insérer plusieurs lignes dans une table.

```
/* Create a table type. */
CREATE TYPE LocationTableType AS TABLE
( LocationName VARCHAR(50)
, CostRate INT );
GO
/* Create a procedure to receive data for the table-valued parameter. */
CREATE PROCEDURE usp_InsertProductionLocation
    @TVP LocationTableType READONLY
    AS
    SET NOCOUNT ON
    INSERT INTO [AdventureWorks2012].[Production].[Location]
       ([Name]
       , [CostRate]
       , [Availability]
       , [ModifiedDate])
    SELECT *, 0, GETDATE()
    FROM @TVP;
GO
```

```
/* Declare a variable that references the type. */
DECLARE @LocationTVP
AS LocationTableType;
/* Add data to the table variable. */
INSERT INTO @LocationTVP (LocationName, CostRate)
    SELECT [Name], 0.00
    FROM
    [AdventureWorks2012].[Person].[StateProvince];
/* Pass the table variable data to a stored procedure. */
EXEC usp_InsertProductionLocation @LocationTVP;
GO
```

2.2 Procédure avec UPDATE

Exemple

```
IF OBJECT_ID ( 'HumanResources.Update_VacationHours', 'P' ) IS NOT NULL
    DROP PROCEDURE HumanResources.Update_VacationHours;
GO
CREATE PROCEDURE HumanResources.Update_VacationHours
@NewHours SMALLINT,
@Rowcount INT OUTPUT
AS
BEGIN
SET NOCOUNT ON;
UPDATE HumanResources.Employee
SET VacationHours =
    ( CASE
        WHEN SalariedFlag = 0 THEN VacationHours + @NewHours
        ELSE @NewHours
        END
    )
WHERE CurrentFlag = 1;
SET @Rowcount = @@rowcount;
END
GO
DECLARE @Rowcount INT
EXEC HumanResources.Update_VacationHours 40, @Rowcount OUTPUT
PRINT @Rowcount;
```

La procédure prend 2 paramètres: un d'entrée **@NewHours** et un de sortie **@RowCount.** La valeur @NewHours est utilisée avec UPDATE pour mettre à jour la colonne VacationHours dans la table Employee. Le paramètre de sortie @RowCount est utilisé pour renvoyer le nombre de lignes affectées à une variable locale. L'expression CASE est utilisée dans la clause SET pour déterminer conditionnellement la valeur définie pour VacationHours. Lorsque l'employé est payé à l'heure (SalariedFlag = 0), VacationHours est défini sur le nombre d'heures actuel plus la valeur spécifiée dans @NewHours ; sinon, VacationHours est défini sur la valeur spécifiée dans @NewHours.

2.3 Gestion des erreurs avec TRY … CATH

```
/*The following example using the TRY...CATCH construct to return error
information
caught during the execution of a stored procedure.
*/

IF OBJECT_ID ( 'Production.uspDeleteWorkOrder', 'P' ) IS NOT NULL
    DROP PROCEDURE Production.uspDeleteWorkOrder;
GO

CREATE PROCEDURE Production.uspDeleteWorkOrder ( @WorkOrderID INT )
AS
SET NOCOUNT ON;
BEGIN TRY
  BEGIN TRANSACTION
  -- Delete rows from the child table, WorkOrderRouting, for the
specified work order.
    DELETE FROM Production.WorkOrderRouting
    WHERE WorkOrderID = @WorkOrderID;
  -- Delete the rows from the parent table, WorkOrder, for the specified
work order.
    DELETE FROM Production.WorkOrder
    WHERE WorkOrderID = @WorkOrderID;
  COMMIT
END TRY

BEGIN CATCH
  -- Determine if an error occurred.
  IF @@TRANCOUNT > 0
    ROLLBACK

  -- Return the error information.
  DECLARE @ErrorMessage NVARCHAR(4000), @ErrorSeverity INT;
  SELECT @ErrorMessage = ERROR_MESSAGE(),@ErrorSeverity =
ERROR_SEVERITY();
  RAISERROR(@ErrorMessage, @ErrorSeverity, 1);
END CATCH;

GO
EXEC Production.uspDeleteWorkOrder 13;
GO
```

```sql
/* Intentionally generate an error by reversing the order in which rows
   are deleted from the parent and child tables. This change does not
   cause an error when the procedure definition is altered, but produces
   an error when the procedure is executed.
*/
ALTER PROCEDURE Production.uspDeleteWorkOrder ( @WorkOrderID INT )
AS

BEGIN TRY
  BEGIN TRANSACTION
  -- Delete the rows from the parent table, WorkOrder, for the specified
work order.
    DELETE FROM Production.WorkOrder
    WHERE WorkOrderID = @WorkOrderID;

  -- Delete rows from the child table, WorkOrderRouting, for the
specified work order.
    DELETE FROM Production.WorkOrderRouting
    WHERE WorkOrderID = @WorkOrderID;
  COMMIT TRANSACTION
END TRY

BEGIN CATCH
  -- Determine if an error occurred.
  IF @@TRANCOUNT > 0
    ROLLBACK TRANSACTION

  -- Return the error information.
  DECLARE @ErrorMessage NVARCHAR(4000), @ErrorSeverity INT;
  SELECT @ErrorMessage = ERROR_MESSAGE(),@ErrorSeverity =
ERROR_SEVERITY();
  RAISERROR(@ErrorMessage, @ErrorSeverity, 1);
END CATCH;
GO

-- Execute the altered procedure.
EXEC Production.uspDeleteWorkOrder 15;
GO
DROP PROCEDURE Production.uspDeleteWorkOrder;
```

Pour plus de details:
https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql?view=sql-server-ver15

2.4 Les Functions

Lorsque SET QUOTED_IDENTIFIER est activé (par défaut), les identificateurs peuvent être délimités par des guillemets doubles (" ") et les littéraux doivent être délimités par des guillemets simples (' ').

2.4.1 Les functions d'aggrégation (MIN, MAX, COUNT, SUM, AVG)

Exemple 1:

```
SELECT AVG(VacationHours)AS 'Average vacation hours',
    SUM(SickLeaveHours) AS 'Total sick leave hours'
FROM HumanResources.Employee
   WHERE JobTitle LIKE 'Vice President%';

SELECT TerritoryID, AVG(Bonus)as 'Average bonus', SUM(SalesYTD) as 'YTD
sales'
FROM Sales.SalesPerson
GROUP BY TerritoryID;
GO

SELECT AVG(DISTINCT ListPrice)
FROM Production.Product;
```

Exemple 2:

```
USE ssawPDW;
SELECT COUNT(*)
FROM dbo.DimEmployee;
SELECT COUNT(DISTINCT Title)
FROM dbo.DimEmployee;

SELECT COUNT(EmployeeKey) AS TotalCount, AVG(SalesAmountQuota) AS
[Average Sales Quota]
FROM dbo.FactSalesQuota
WHERE SalesAmountQuota > 500000 AND CalendarYear = 2001;


SELECT DepartmentName,
       COUNT(EmployeeKey)AS EmployeesInDept
FROM dbo.DimEmployee
GROUP BY DepartmentName
HAVING COUNT(EmployeeKey) > 15;
```

Exemple 3:

```sql
SELECT Color, SUM(ListPrice), SUM(StandardCost)
FROM Production.Product
WHERE Color IS NOT NULL
    AND ListPrice != 0.00
    AND Name LIKE 'Mountain%'
GROUP BY Color
ORDER BY Color;
GO

SELECT DISTINCT Name
       , MIN(Rate) OVER (PARTITION BY edh.DepartmentID) AS MinSalary
       , MAX(Rate) OVER (PARTITION BY edh.DepartmentID) AS MaxSalary
       , AVG(Rate) OVER (PARTITION BY edh.DepartmentID) AS AvgSalary
       ,COUNT(edh.BusinessEntityID) OVER (PARTITION BY edh.DepartmentID)
AS EmployeesPerDept
FROM HumanResources.EmployeePayHistory AS eph
JOIN HumanResources.EmployeeDepartmentHistory AS edh
     ON eph.BusinessEntityID = edh.BusinessEntityID
JOIN HumanResources.Department AS d
 ON d.DepartmentID = edh.DepartmentID
WHERE edh.EndDate IS NULL
ORDER BY Name;
```

2.4.2 Les fonctions de configurations

```sql
Select DATEDIFF(day,DueDate,ModifiedDate) as DifferenceDay from
Purchasing.PurchaseOrderDetail;
```

Options: year, month, day, week, quarter, hour, minute, second, millisecond, microsecond

```sql
SELECT SYSDATETIME();
SELECT DATEDIFF(year,'2005-12-31 23:59:59.9999999', SYSDATETIME());
SELECT @@MAX_CONNECTIONS AS 'Max Connections';  /*32767 usagers peuvent
se connecter simultanément*/
SET LOCK_TIMEOUT 1800;  /*En milliseconde */
SELECT @@LOCK_TIMEOUT AS [Lock Timeout];
GO
SELECT @@TEXTSIZE AS 'Text Size'
SELECT @@VERSION AS 'SQL Server Version';
SELECT @@SERVERNAME AS 'Server Name';
SELECT @@SERVICENAME AS 'Service Name';
SELECT @@REMSERVER AS 'Remote Server Name' ;
SELECT @@SPID AS 'ID', SYSTEM_USER AS 'Login Name', USER AS 'User Name';
-- Session ID
SELECT @@LANGUAGE AS 'Language Name';
```

Plus de details: https://learn.microsoft.com/en-us/sql/t-sql/functions/configuration-functions-transact-sql?view=sql-server-ver16

Convertion avec CONVERT et CAT

```
SELECT
   GETDATE() AS UnconvertedDateTime,
   CAST(GETDATE() AS NVARCHAR(30)) AS UsingCast,
   CONVERT(nvarchar(30), GETDATE(), 126) AS UsingConvertTo_ISO8601  ;
GO

SELECT 'The list price is ' + CAST(ListPrice AS VARCHAR(12)) AS ListPrice
FROM dbo.DimProduct
WHERE ListPrice BETWEEN 350.00 AND 400.00;

SELECT DISTINCT CAST(EnglishProductName AS CHAR(10)) AS Name, ListPrice
FROM dbo.DimProduct
WHERE EnglishProductName LIKE 'Long-Sleeve Logo Jersey, M';

DECLARE @myval DECIMAL (5, 2);
SET @myval = 193.57;
SELECT CAST(CAST(@myval AS VARBINARY(20)) AS DECIMAL(10,5));
-- Or, using CONVERT
SELECT CONVERT(DECIMAL(10,5), CONVERT(VARBINARY(20), @myval));

DECLARE @a INT = 45, @b INT = 40;
SELECT [Result] = IIF( @a > @b, 'TRUE', 'FALSE' );

USE AdventureWorks2012;
GO
SELECT ProductCategoryID, CHOOSE (ProductCategoryID, 'A','B','C','D','E')
AS Expression1
FROM Production.ProductCategory;

SELECT Name, ModifiedDate,
CHOOSE(MONTH(ModifiedDate),'Winter','Winter',
'Spring','Spring','Spring','Summer','Summer',
                          'Summer','Autumn','Autumn','Autumn','Winter')
AS Quarter_Modified
FROM SalesLT.ProductModel AS PM
WHERE Name LIKE '%Frame%'
ORDER BY ModifiedDate;

SELECT CONCAT ( 'Happy ', 'Birthday ', 11, '/', '25' ) AS Result;

SELECT LEFT(Name, 5)   FROM Production.Product
ORDER BY ProductID;
GO
SELECT RIGHT(FirstName, 5) AS 'First Name'  FROM Person.Person
WHERE BusinessEntityID < 5
ORDER BY FirstName;
GO
```

```
SELECT UPPER(RTRIM(LastName)) + ', ' + FirstName AS Name
FROM dbo.DimEmployee
ORDER BY LastName;

SELECT REPLACE('abcdefghicde','cde','xxx');
```

Voir Annexe 2 pour plus de détails

2.4.3 User-defined functions (Fonctions définies par l'usager)

Limitations et restrictions

- Les fonctions définies par l'utilisateur ne peuvent pas être utilisées pour effectuer des actions qui modifient l'état de la base de données.
- Les fonctions définies par l'utilisateur ne peuvent pas contenir une clause OUTPUT INTO ayant une table comme cible.
- Les fonctions définies par l'utilisateur ne peuvent pas renvoyer plusieurs ensembles de résultats. Utilisez une procédure stockée si vous devez renvoyer plusieurs ensembles de résultats.
- La gestion des erreurs est restreinte dans une fonction définie par l'utilisateur. Une UDF ne prend pas en charge TRY...CATCH, @ERROR ou RAISERROR.
- Les fonctions définies par l'utilisateur ne peuvent pas appeler une procédure stockée, mais peuvent appeler une procédure stockée étendue.
- Les fonctions définies par l'utilisateur ne peuvent pas utiliser de tables SQL dynamiques ou temporaires. Les variables de tableau sont autorisées.
- Les instructions SET ne sont pas autorisées dans une fonction définie par l'utilisateur.

Exemple :

```
use AdventureWorks2019;
IF OBJECT_ID (N'dbo.ufnGetInventoryStock', N'FN') IS NOT NULL
    DROP FUNCTION ufnGetInventoryStock;
GO
CREATE FUNCTION dbo.ufnGetInventoryStock(@ProductID int)
RETURNS int
AS
BEGIN -- Returns the stock level for the product.
    DECLARE @ret int;
    SELECT @ret = SUM(p.Quantity)
    FROM Production.ProductInventory p
    WHERE p.ProductID = @ProductID
        AND p.LocationID = '6';
     IF (@ret IS NULL)
        SET @ret = 0;
    RETURN @ret;
END;
GO
SELECT ProductModelID, Name, dbo.ufnGetInventoryStock(ProductID)AS
CurrentSupply
FROM Production.Product WHERE ProductModelID BETWEEN 75 and 80;
```

```
IF OBJECT_ID (N'Sales.ufn_SalesByStore', N'IF') IS NOT NULL
    DROP FUNCTION Sales.ufn_SalesByStore;
GO
CREATE FUNCTION Sales.ufn_SalesByStore (@storeid int)
RETURNS TABLE
AS
RETURN
(
    SELECT P.ProductID, P.Name, SUM(SD.LineTotal) AS 'Total'
    FROM Production.Product AS P
    JOIN Sales.SalesOrderDetail AS SD ON SD.ProductID = P.ProductID
    JOIN Sales.SalesOrderHeader AS SH ON SH.SalesOrderID =
SD.SalesOrderID
    JOIN Sales.Customer AS C ON SH.CustomerID = C.CustomerID
    WHERE C.StoreID = @storeid
    GROUP BY P.ProductID, P.Name
);
GO
SELECT * FROM Sales.ufn_SalesByStore (602);
SELECT * FROM Sales.Customer
```

Suivez les explications du Prof.

3. Automatisation (A3)

3.1 SQL Server PowerShell

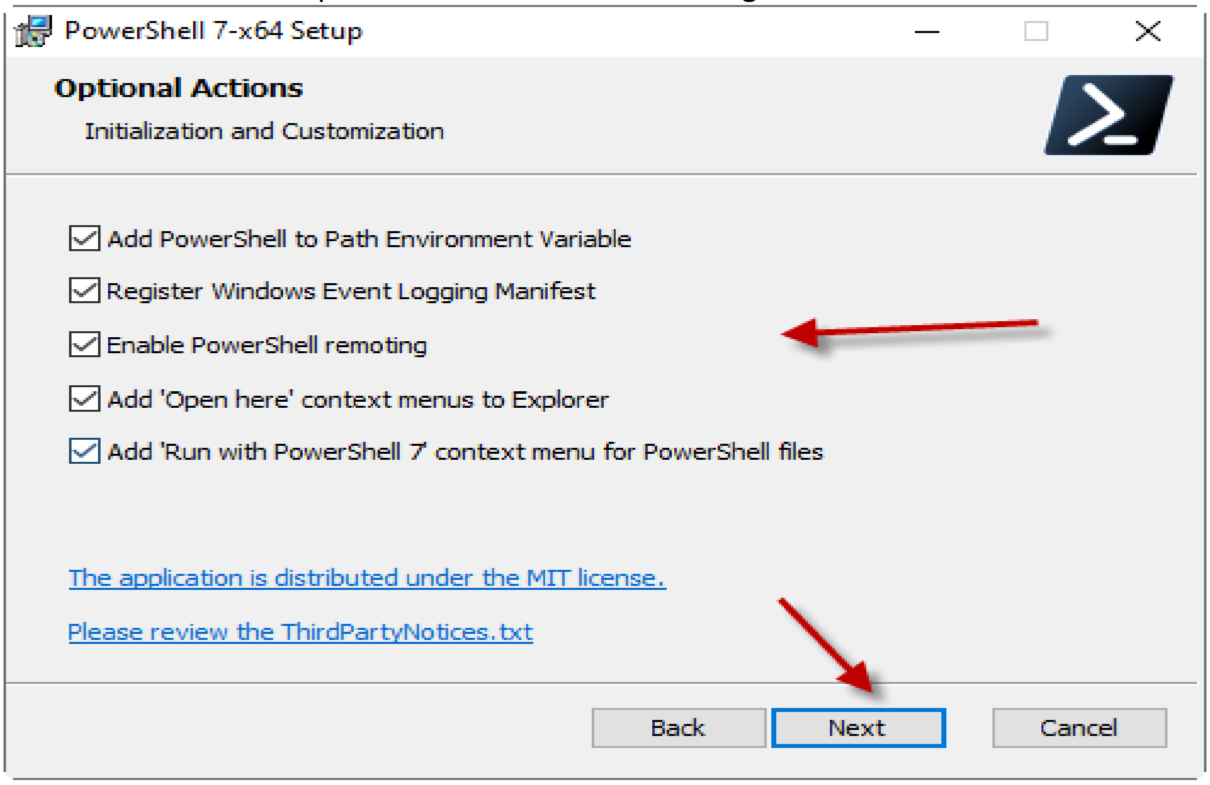
Le « SQL Server Database Engine » prend en charge l'environnement de script PowerShell pour gérer les instances du moteur de base de données et les objets dans les instances. On peut donc créer et exécuter des requêtes de moteur de base de données contenant Transact-SQL et XQuery dans des environnements très similaires aux environnements de script.

3.1.1 Installation des Outils nécessaires

A) PowerShell 5.1 vient avec Windows 10. Téléchargez et Installez la version la plus recente 7.2.7 64 bit. Le PowerShell ISE (Editeur de Script) n'est plus existant dans les versions 6 et 7.

PowerShell-7.2.7-win-x64.msi à partir de ce lien :
https://github.com/PowerShell/PowerShell/releases/download/v7.2.7/PowerShell-7.2.7-win-x64.msi

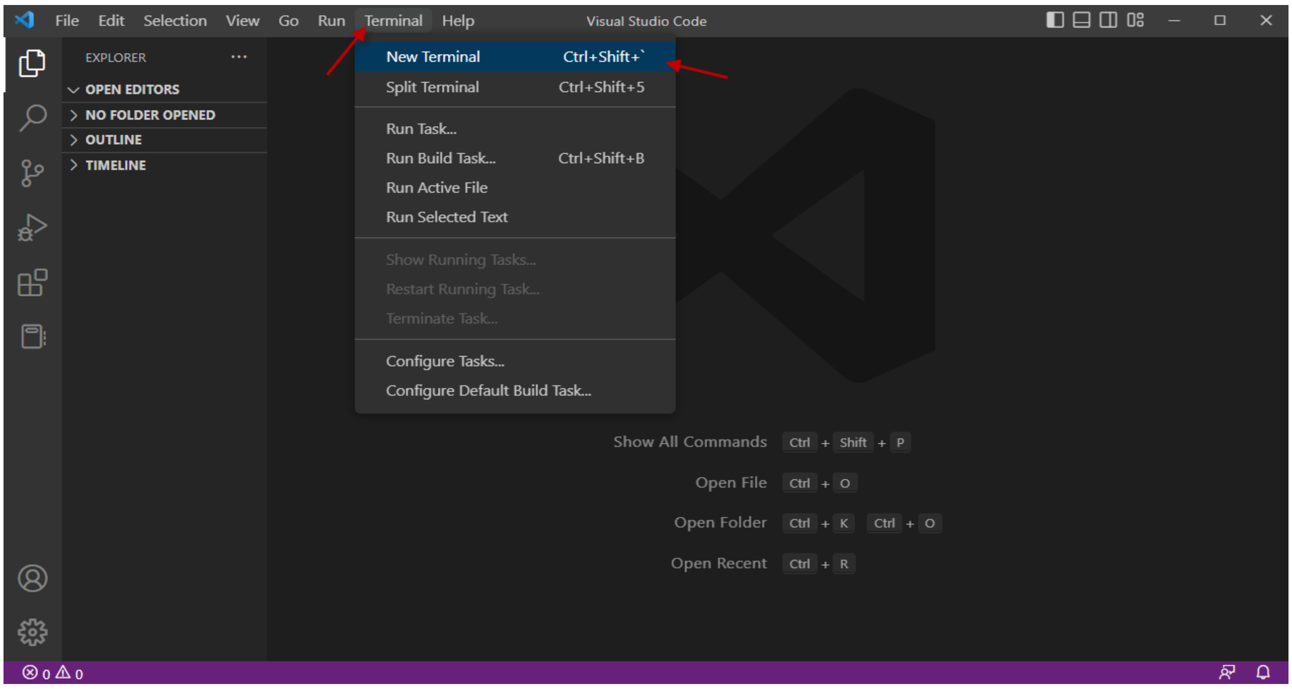
Choisissez toutes les options comme le montre l'image si-dessous :

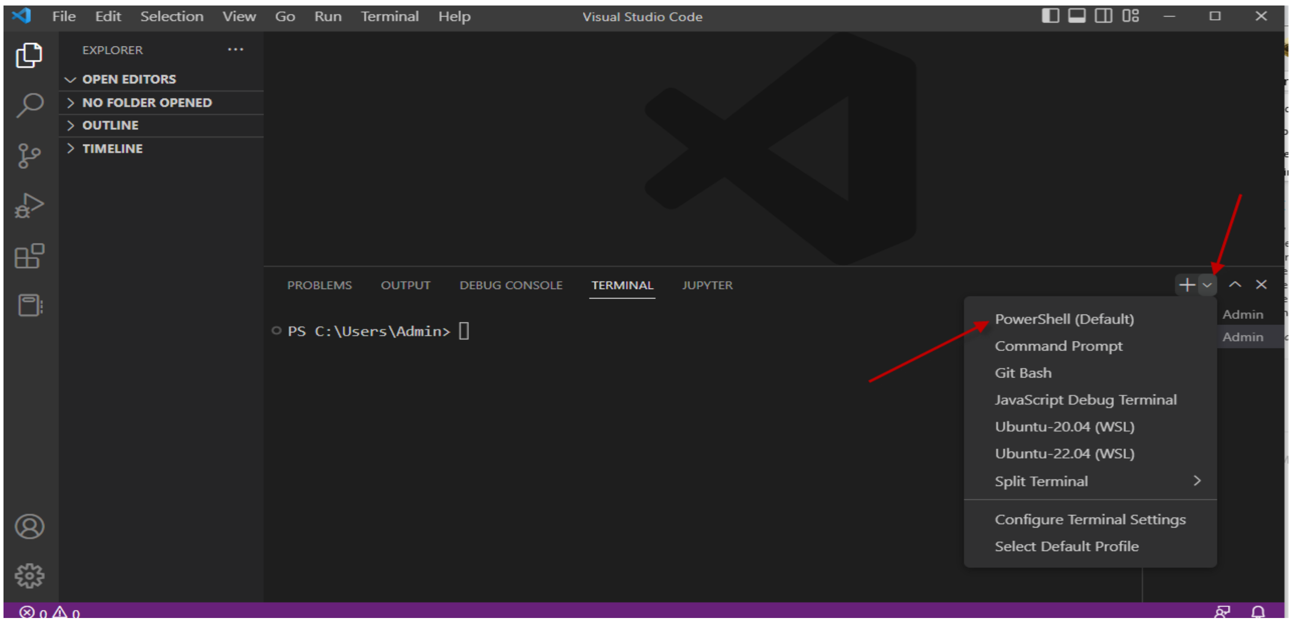

PowerShell 7-x64 Setup
Optional Actions
Initialization and Customization
The application is distributed under the MIT license.
Please review the ThirdPartyNotices.txt
Back
Next
Cancel

☑ Add PowerShell to Path Environment Variable
☑ Register Windows Event Logging Manifest
☑ Enable PowerShell remoting
☑ Add 'Open here' context menus to Explorer
☑ Add 'Run with PowerShell 7' context menu for PowerShell files

B) Téléchargez et Installer l'éditeur Visual Studio Code pour remplace le PowerShell ISE Version 1.73.0
https://code.visualstudio.com/docs/?dv=win64

Cliquez 2 fois le fichier téléchargé VSCodeSetup-x64-1.73.0.exe pour lancer l'installation. Suivre les instructions juqu'à la fin, et Ouvrir Visual Studio Code 1.73.0



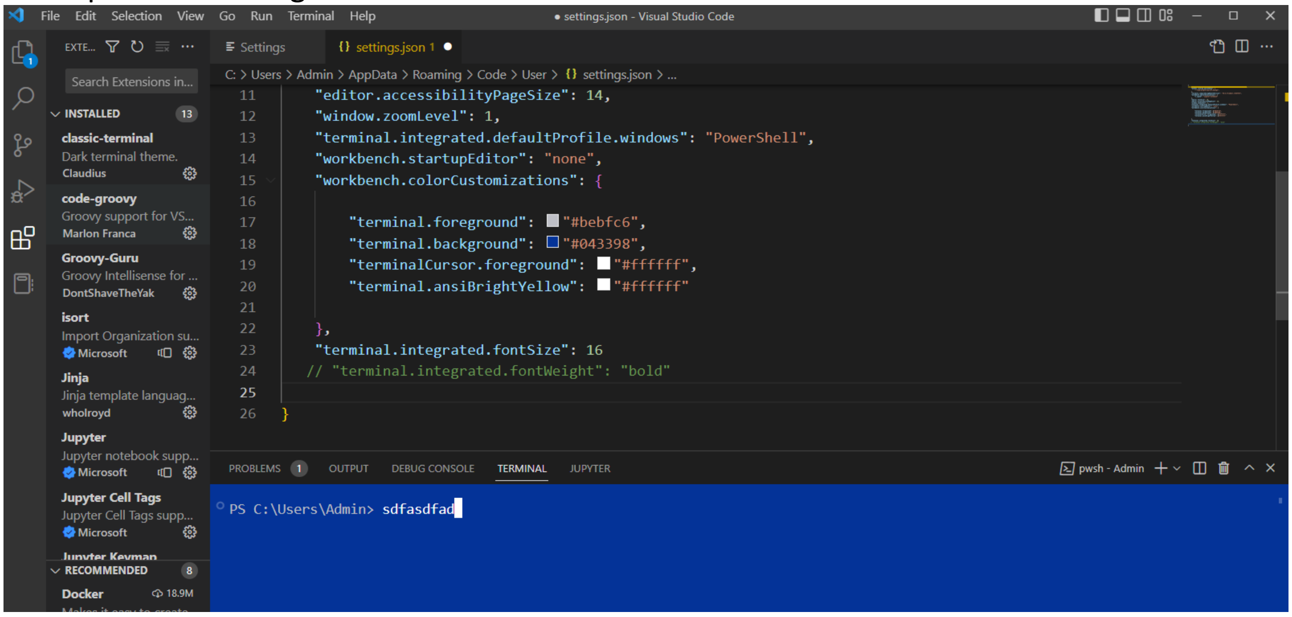


```
PS C:\Users\Admin> 
```

File=>Preferences=>Settings et rechercher “workbench color”
Workbench: Color Customizations cliquez sur Edit in settings.json et entrez ce code puis sauvegardez et sortez.

```
"workbench.colorCustomizations": {

    "terminal.foreground": "#bebfc6",
     "terminal.background": "#043398",
     "terminalCursor.foreground": "#ffffff",
     "terminal.ansiBrightYellow": "#ffffff"
},
"terminal.integrated.fontSize": 16
```

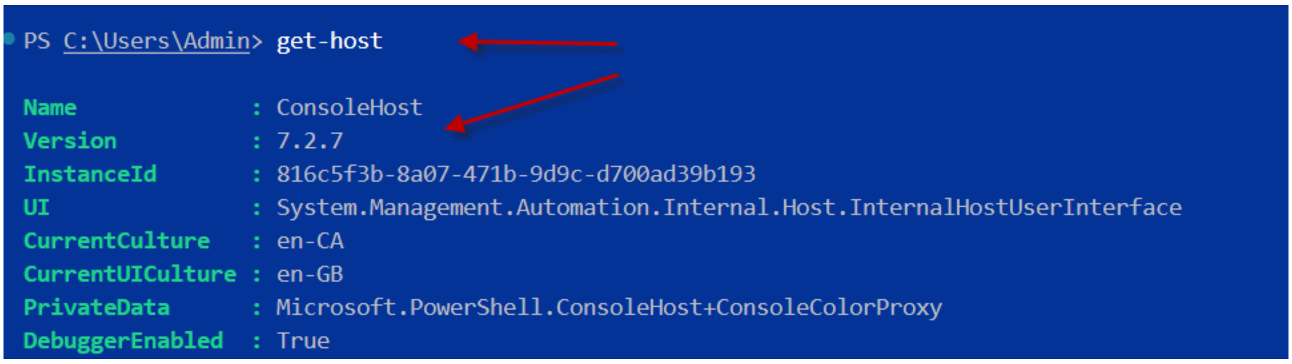
Ctrl+Space for Intelligencia


File Edit Selection View Go Run Terminal Help
● settings.json - Visual Studio Code
Settings
settings.json 1
C: > Users > Admin > AppData > Roaming > Code > User > settings.json > ...
INSTALLED 13
classic-terminal
Dark terminal theme.
Claudius
code-groovy
Groovy support for VS...
Marlon Franca
Groovy-Guru
Groovy Intellisense for ...
DontShaveTheYak
isort
Import Organization su...
Microsoft
Jinja
Jinja template languag...
wholroyd
Jupyter
Jupyter notebook supp...
Microsoft
Jupyter Cell Tags
Jupyter Cell Tags supp...
Microsoft
RECOMMENDED 8
Docker
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
pwsh - Admin

```
    "editor.accessibilityPageSize": 14,
    "window.zoomLevel": 1,
    "terminal.integrated.defaultProfile.windows": "PowerShell",
    "workbench.startupEditor": "none",
    "workbench.colorCustomizations": {

        "terminal.foreground": "#bebfc6",
        "terminal.background": "#043398",
        "terminalCursor.foreground": "#ffffff",
        "terminal.ansiBrightYellow": "#ffffff"

    },
    "terminal.integrated.fontSize": 16
   // "terminal.integrated.fontWeight": "bold"

}
```

```
PS C:\Users\Admin> sdfasdfad
```

Tapez get-host pour voir la version de votre PowerShell installé. Elle doit être 7.2.7 comme le montre l'écran suivant :



```
PS C:\Users\Admin> get-host

Name             : ConsoleHost
Version          : 7.2.7
InstanceId       : 816c5f3b-8a07-471b-9d9c-d700ad39b193
UI               : System.Management.Automation.Internal.Host.InternalHostUserInterface
CurrentCulture   : en-CA
CurrentUICulture : en-GB
PrivateData      : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
DebuggerEnabled  : True
```

Ensuite vous passez à l'Étape C pour installer le module SQL Server PowerShell.

C) Téléchargez et Installez l'outil SQLServer PowerShell à partir de ce lien : https://learn.microsoft.com/en-us/sql/powershell/download-sql-server-ps-module?view=sql-server-ver15
Le fichier téléchargé doit être : **sqlserver.21.1.18256.nupkg**

Il y a 2 modules dans le SQL Server Powershell qui sont **SQLServer** et **SQLPS**
SQLPS est discontinu. Pour les versions SSMS 17.0 et plus récentes il faudra installer l'outil. Nous allons donc principalement utiliser le module SQLServer.

Pour utiliser le SQL Agent Job step il faut importer le module en tapant ce script de PoweShell :

```
#NOSQLPS
```

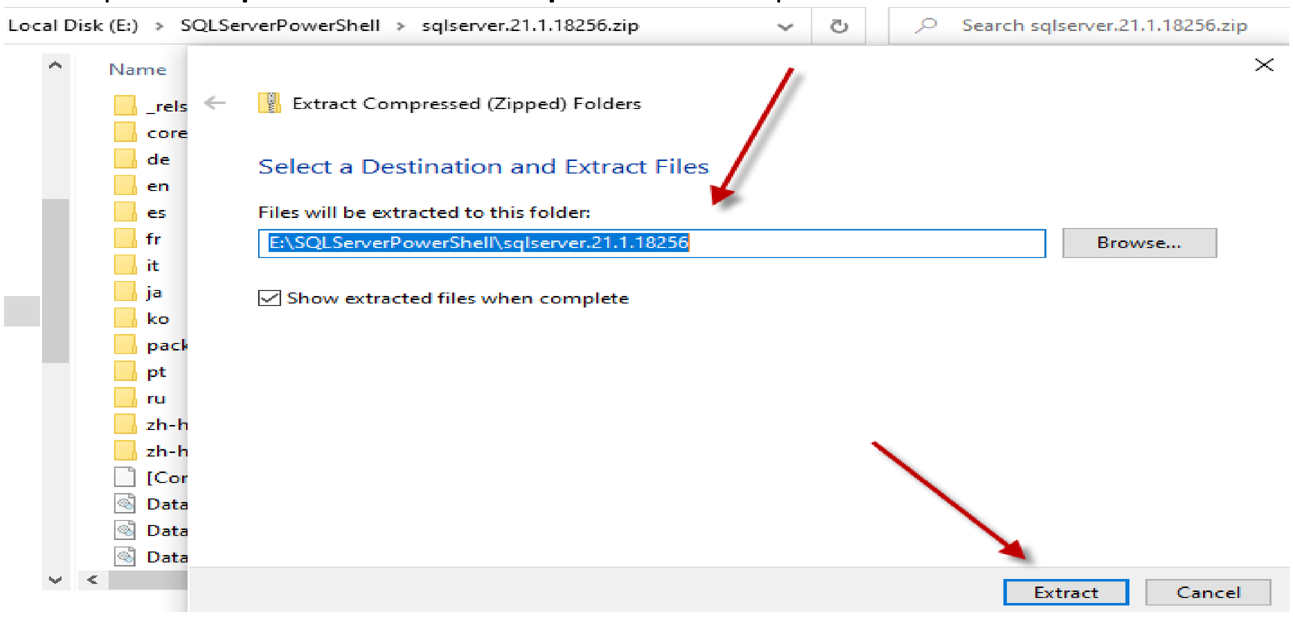
```
Import-Module -Name SqlServer

/* From PowerShell Command Line or .SP1 script*/
-- To install or update:
Install-Module -Name SqlServer;
--To view the version of SQLServer Module installed
Get-Module SqlServer -ListAvailable
ou (Get-Module SqlServer).Version
-- To Update the installed version of the SqlServer module
Update-Module -Name SqlServer -AllowClobber
--To overwrite the previous version of SQLServer Module installed
Install-Module -Name SqlServer -AllowClobber
--To remove older versions:
Uninstall-module -Name SQLServer -RequiredVersion "<version number>"
-- To discover pre-release versions of SQLServer Module
Find-Module SqlServer -AllowPrerelease
-- Liste des SqlServer CMDLETS ds PowerShell:
Voir en Annexe 3:
URL: https://learn.microsoft.com/en-us/sql/powershell/download-sql-server-ps-module?view=sql-server-ver15
```

Si vous avez une connexion Internet, lancer la commande suivant dans le Terminal PowerShell de VS Code pour installer le module SQLServer PowerShell : `Import-Module -Name SqlServer`

Sinon, procédez à une installation Manuelle.

3.1.2 Installation Manuelle de SQLServer PowerShell

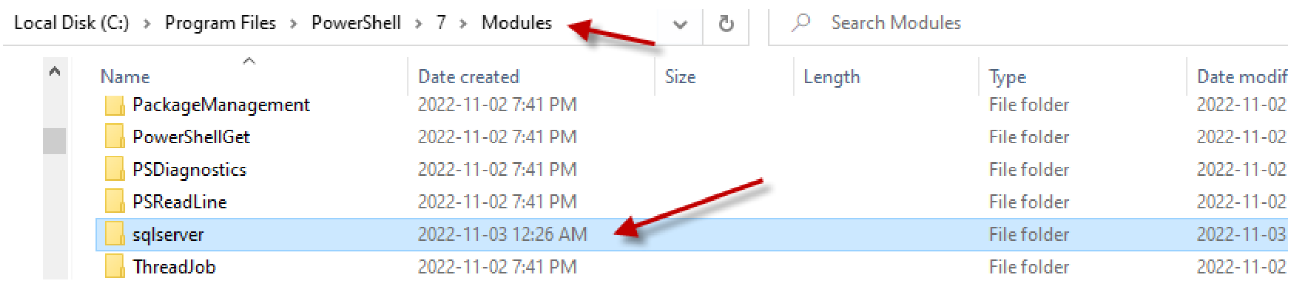
1. Débloquez le fichier **sqlserver.21.1.18256.nupkg** en tapant la commande suivante dans le Terminal PowerShell de VSCode(PS E:\SQLServerPowerShell>) :
   Unblock-File -Path E:\SQLServerPowerShell\sqlserver.21.1.18256.nupkg
2. Renommez le fichier **sqlserver.21.1.18256.nupkg** en **sqlserver.21.1.18256.zip**
3. Décompressez **sqlserver.21.1.18256.zip** à l'aide de l'Explorateur Windows


Local Disk (E:) > SQLServerPowerShell > sqlserver.21.1.18256.zip
Search sqlserver.21.1.18256.zip
Name
_rels
core
de
en
es
fr
it
ja
ko
pack
pt
ru
zh-h
zh-h
[Cor
Data
Data
Data
Extract Compressed (Zipped) Folders
Select a Destination and Extract Files
Browse...
Extract
Cancel

Files will be extracted to this folder:
E:\SQLServerPowerShell\sqlserver.21.1.18256
☑ Show extracted files when complete

4. Supprimez les 4 éléments spécifiques suivant dans le Dossier **sqlserver.21.1.18256**
   - A folder named **_rels** - contains a .rels file that lists the dependencies
   - A folder named **package** - contains the NuGet-specific data
   - A file named **[Content_Types].xml** - describes how extensions like PowerShellGet work with NuGet
   - A file named **<name>.nuspec** - contains the bulk of the metadata

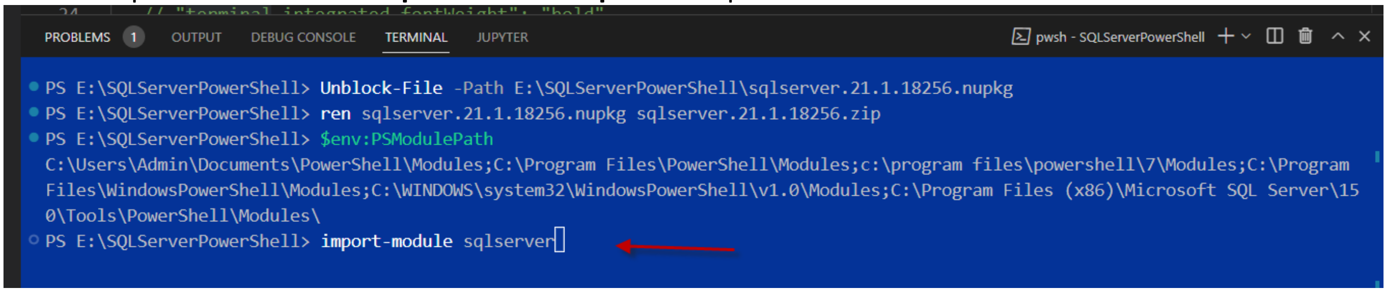
5. Renommez le dossier **sqlserver.21.1.18256** par **sqlserver**.
   Rename the folder. The default folder name is usually <name>.<version>. The version can include -prerelease if the module is tagged as a prerelease version. Rename the folder to just the module name. For example, azurerm.storage.5.0.4-preview becomes azurerm.storage.
6. Copiez le dossier **sqlserver** dans l'un des chemins d'accès aux Modules de PowerShell. Il faut taper la commande suivante pour découvrir ce chemin d'accès :
   $env:PSModulePath
   Copy the folder to one of the folders in the $env:PSModulePath value. $env:PSModulePath is a semicolon-delimited set of paths in which PowerShell should look for modules.
   Par exemple moi j'ai choisi ce chemin d'accès parceque j'ai la version 7 de PowerShell :
   c:\program files\powershell\7\Modules
   Donc copiez le dossier **sqlserver** dans le dossier **c:\program files\powershell\7\Modules**



Local Disk (C:) › Program Files › PowerShell › 7 › Modules

Name	Date created	Size	Length	Type	Date modif
PackageManagement	2022-11-02 7:41 PM			File folder	2022-11-02
PowerShellGet	2022-11-02 7:41 PM			File folder	2022-11-02
PSDiagnostics	2022-11-02 7:41 PM			File folder	2022-11-02
PSReadLine	2022-11-02 7:41 PM			File folder	2022-11-02
sqlserver	2022-11-03 12:26 AM			File folder	2022-11-03
ThreadJob	2022-11-02 7:41 PM			File folder	2022-11-02

7. Tapez la commande **import-module sqlserver** à partir du Terminal PowerShell de VSCode :



```
PS E:\SQLServerPowerShell> Unblock-File -Path E:\SQLServerPowerShell\sqlserver.21.1.18256.nupkg
PS E:\SQLServerPowerShell> ren sqlserver.21.1.18256.nupkg sqlserver.21.1.18256.zip
PS E:\SQLServerPowerShell> $env:PSModulePath
C:\Users\Admin\Documents\PowerShell\Modules;C:\Program Files\PowerShell\Modules;c:\program files\powershell\7\Modules;C:\Program
Files\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules;C:\Program Files (x86)\Microsoft SQL Server\15
0\Tools\PowerShell\Modules\
PS E:\SQLServerPowerShell> import-module sqlserver
```

Tapez get-module pour voir si notre module sqlserver est listé.
Measure-Object cmdlet : https://learn.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/measure-object?view=powershell-7.2

Références

https://learn.microsoft.com/en-us/sql/samples/adventureworks-install-configure?view=sql-server-ver16&tabs=ssms
https://learn.microsoft.com/en-us/sql/t-sql/statements/create-trigger-transact-sql?view=sql-server-ver15
https://learn.microsoft.com/en-us/sql/relational-databases/triggers/logon-triggers?view=sql-server-ver15
https://learn.microsoft.com/en-us/sql/t-sql/statements/create-procedure-transact-sql?view=sql-server-ver15
https://learn.microsoft.com/en-us/sql/ssms/scripting/database-engine-scripting?view=sql-server-ver16

https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver15

https://go.microsoft.com/fwlink/p/?linkid=866662

https://aka.ms/ssmsfullsetup

https://github.com/iCodeMechanic/Essentials-of-Sql-Server-Performance-for-Every-Developer/find/master

https://github.com/dbcli/mssql-cli/blob/main/doc/usage_guide.md

https://learn.microsoft.com/en-us/sql/tools/mssql-cli?view=sql-server-ver15

https://learn.microsoft.com/en-us/sql/tools/sqlcmd-utility?view=sql-server-ver15

https://go.microsoft.com/fwlink/?linkid=2142258

https://learn.microsoft.com/en-us/sql/ssms/scripting/sqlcmd-run-transact-sql-script-files?view=sql-server-ver15

https://www.powershellgallery.com/packages/SqlServer/21.1.18256

https://learn.microsoft.com/en-us/sql/relational-databases/tables/rename-columns-database-engine?source=recommendations&view=sql-server-ver16

https://learn.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql?view=sql-server-ver16

https://learn.microsoft.com/en-us/sql/relational-databases/tables/create-tables-database-engine?view=sql-server-ver16

https://www.sqlrx.com/find-and-modify-file-growth-settings-for-all-databases/

Annexes

Annexe-01: Best practices

Although this isn't an exhaustive list of best practices, these suggestions may improve procedure performance.

- Use the SET NOCOUNT ON statement as the first statement in the body of the procedure. That is, place it just after the AS keyword. This turns off messages that SQL Server sends back to the client after any SELECT, INSERT, UPDATE, MERGE, and DELETE statements are executed. This keeps the output generated to a minimum for clarity. There is no measurable performance benefit however on today's hardware. For information, see SET NOCOUNT (Transact-SQL).
- Use schema names when creating or referencing database objects in the procedure. It takes less processing time for the Database Engine to resolve object names if it doesn't have to search multiple schemas. It also prevents permission and access problems caused by a user's default schema being assigned when objects are created without specifying the schema.
- Avoid wrapping functions around columns specified in the WHERE and JOIN clauses. Doing so makes the columns non-deterministic and prevents the query processor from using indexes.
- Avoid using scalar functions in SELECT statements that return many rows of data. Because the scalar function must be applied to every row, the resulting behavior is like row-based processing and degrades performance.
- Avoid the use of `SELECT *`. Instead, specify the required column names. This can prevent some Database Engine errors that stop procedure execution. For example, a `SELECT *` statement that returns data from a 12 column table and then inserts that data into a 12 column temporary table succeeds until the number or order of columns in either table is changed.
- Avoid processing or returning too much data. Narrow the results as early as possible in the procedure code so that any subsequent operations performed by the procedure are done using the smallest data set possible. Send just the essential data to the client application. It is more efficient than sending extra data across the network and forcing the client application to work through unnecessarily large result sets.
- Use explicit transactions by using BEGIN/COMMIT TRANSACTION and keep transactions as short as possible. Longer transactions mean longer record locking and a greater potential for deadlocking.

- Use the Transact-SQL TRY...CATCH feature for error handling inside a procedure. TRY...CATCH can encapsulate an entire block of Transact-SQL statements. This not only creates less performance overhead, it also makes error reporting more accurate with significantly less programming.
- Use the DEFAULT keyword on all table columns that are referenced by CREATE TABLE or ALTER TABLE Transact-SQL statements in the body of the procedure. This prevents passing NULL to columns that don't allow null values.
- Use NULL or NOT NULL for each column in a temporary table. The ANSI_DFLT_ON and ANSI_DFLT_OFF options control the way the Database Engine assigns the NULL or NOT NULL attributes to columns when these attributes aren't specified in a CREATE TABLE or ALTER TABLE statement. If a connection executes a procedure with different settings for these options than the connection that created the procedure, the columns of the table created for the second connection can have different nullability and exhibit different behavior. If NULL or NOT NULL is explicitly stated for each column, the temporary tables are created by using the same nullability for all connections that execute the procedure.
- Use modification statements that convert nulls and include logic that eliminates rows with null values from queries. Be aware that in Transact-SQL, NULL isn't an empty or "nothing" value. It is a placeholder for an unknown value and can cause unexpected behavior, especially when querying for result sets or using AGGREGATE functions.
- Use the UNION ALL operator instead of the UNION or OR operators, unless there is a specific need for distinct values. The UNION ALL operator requires less processing overhead because duplicates aren't filtered out of the result set.

Annexe-02 – CONVERT Date/Time

Using CONVERT with datetime data in different formats

Starting with `GETDATE()` values, this example uses `CONVERT` to display of all the date and time styles in section Date and Time styles of this article.

Format #	Example query	Sample result
0	`SELECT CONVERT(NVARCHAR, GETDATE(), 0)`	Aug 23 2019 1:39PM
1	`SELECT CONVERT(NVARCHAR, GETDATE(), 1)`	08/23/19
2	`SELECT CONVERT(NVARCHAR, GETDATE(), 2)`	19.08.23
3	`SELECT CONVERT(NVARCHAR, GETDATE(), 3)`	23/08/19
4	`SELECT CONVERT(NVARCHAR, GETDATE(), 4)`	23.08.19
5	`SELECT CONVERT(NVARCHAR, GETDATE(), 5)`	23-08-19
6	`SELECT CONVERT(NVARCHAR, GETDATE(), 6)`	23 Aug 19
7	`SELECT CONVERT(NVARCHAR, GETDATE(), 7)`	Aug 23, 19
8 or 24 or 108	`SELECT CONVERT(NVARCHAR, GETDATE(), 8)`	13:39:17
9 or 109	`SELECT CONVERT(NVARCHAR, GETDATE(), 9)`	Aug 23 2019 1:39:17:090PM
10	`SELECT CONVERT(NVARCHAR, GETDATE(), 10)`	08-23-19
11	`SELECT CONVERT(NVARCHAR, GETDATE(), 11)`	19/08/23
12	`SELECT CONVERT(NVARCHAR, GETDATE(), 12)`	190823
13 or 113	`SELECT CONVERT(NVARCHAR, GETDATE(), 13)`	23 Aug 2019 13:39:17:090
14 or 114	`SELECT CONVERT(NVARCHAR, GETDATE(), 14)`	13:39:17:090
20 or 120	`SELECT CONVERT(NVARCHAR, GETDATE(), 20)`	2019-08-23 13:39:17
21 or 25 or 121	`SELECT CONVERT(NVARCHAR, GETDATE(), 21)`	2019-08-23 13:39:17.090
22	`SELECT CONVERT(NVARCHAR, GETDATE(), 22)`	08/23/19 1:39:17 PM
23	`SELECT CONVERT(NVARCHAR, GETDATE(), 23)`	2019-08-23
101	`SELECT CONVERT(NVARCHAR, GETDATE(), 101)`	08/23/2019
102	`SELECT CONVERT(NVARCHAR, GETDATE(), 102)`	2019.08.23
103	`SELECT CONVERT(NVARCHAR, GETDATE(), 103)`	23/08/2019
104	`SELECT CONVERT(NVARCHAR, GETDATE(), 104)`	23.08.2019
105	`SELECT CONVERT(NVARCHAR, GETDATE(), 105)`	23-08-2019
106	`SELECT CONVERT(NVARCHAR, GETDATE(), 106)`	23 Aug 2019
107	`SELECT CONVERT(NVARCHAR, GETDATE(), 107)`	Aug 23, 2019
110	`SELECT CONVERT(NVARCHAR, GETDATE(), 110)`	08-23-2019
111	`SELECT CONVERT(NVARCHAR, GETDATE(), 111)`	2019/08/23

Format #	Example query	Sample result
112	`SELECT CONVERT(NVARCHAR, GETDATE(), 112)`	20190823
113	`SELECT CONVERT(NVARCHAR, GETDATE(), 113)`	23 Aug 2019 13:39:17.090
120	`SELECT CONVERT(NVARCHAR, GETDATE(), 120)`	2019-08-23 13:39:17
121	`SELECT CONVERT(NVARCHAR, GETDATE(), 121)`	2019-08-23 13:39:17.090
126	`SELECT CONVERT(NVARCHAR, GETDATE(), 126)`	2019-08-23T13:39:17.090
127	`SELECT CONVERT(NVARCHAR, GETDATE(), 127)`	2019-08-23T13:39:17.090
130	`SELECT CONVERT(NVARCHAR, GETDATE(), 130)`	22 ذو الحجة 1440 1:39:17.090P
131	`SELECT CONVERT(NVARCHAR, GETDATE(), 131)`	22/12/1440 1:39:17.090PM

Annexe-03 – SQLServer PowerShell CmdLets

CmdLets	Description
Add-RoleMember	Adds a member to a specific Role of a specific database.
Add-SqlAvailabilityDatabase	Adds primary databases to an availability group or joins secondary databases to an availability group.
Add-SqlAvailabilityGroupListenerStaticIp	Adds a static IP address to an availability group listener.
Add-SqlAzureAuthenticationContext	Performs authentication to Azure and acquires an authentication token.
Add-SqlColumnEncryptionKeyValue	Adds an encrypted value for an existing column encryption key object in the database.
Add-SqlFirewallRule	Adds a Windows Firewall rule to allow connections to a specific instance of SQL Server.
Add-SqlLogin	Creates a Login object in an instance of SQL Server.
Backup-ASDatabase	Enables a database administrator to take the backup of Analysis Service Database to a file.
Backup-SqlDatabase	Backs up SQL Server database objects.
Complete-SqlColumnMasterKeyRotation	Completes the rotation of a column master key.
Convert-UrnToPath	Converts a SQL Server Management Object URN to a Windows PowerShell provider path.
ConvertFrom-EncodedSqlName	Returns the original SQL Server identifier when given an identifier that has been encoded into a format usable in Windows PowerShell paths.
ConvertTo-EncodedSqlName	Encodes extended characters in SQL Server names to formats usable in Windows PowerShell paths.

Disable-SqlAlwaysOn	Disables the Always On Availability Groups feature for a server.
Enable-SqlAlwaysOn	Enables the Always On Availability Groups feature.
Export-SqlVulnerabilityAssessmentBaselineSet	Exports a Vulnerability Assessment baseline set to a file.
Export-SqlVulnerabilityAssessmentScan	Exports a Vulnerability Assessment scan to a file.
Get-SqlAgent	Gets a SQL Agent object that is present in the target instance of SQL Server.
Get-SqlAgentJob	Gets a SQL Agent Job object for each job that is present in the target instance of SQL Agent.
Get-SqlAgentJobHistory	Gets the job history present in the target instance of SQL Agent.
Get-SqlAgentJobSchedule	Gets a job schedule object for each schedule that is present in the target instance of SQL Agent Job.
Get-SqlAgentJobStep	Gets a SQL JobStep object for each step that is present in the target instance of SQL Agent Job.
Get-SqlAgentSchedule	Gets a SQL job schedule object for each schedule that is present in the target instance of SQL Agent.
Get-SqlAssessmentItem	Gets SQL Assessment best practice checks available for a chosen SQL Server object.
Get-SqlBackupHistory	Gets backup information about databases and returns SMO BackupSet objects for each Backup record found based on the parameters specified to this cmdlet.
Get-SqlColumnEncryptionKey	Gets all column encryption key objects defined in the database, or gets one column encryption key object with the specified name.
Get-SqlColumnMasterKey	Gets the column master key objects defined in the database or gets one column master key object with the specified name.
Get-SqlCredential	Gets a SQL credential object.
Get-SqlDatabase	Gets a SQL database object for each database that is present in the target instance of SQL Server.
Get-SqlErrorLog	Gets the SQL Server error logs.
Get-SqlInstance	Gets a SQL Instance object for each instance of SQL Server that is present on the target computer.
Get-SqlLogin	Returns Login objects in an instance of SQL Server.
Get-SqlSensitivityClassification	Get the sensitivity label and information type of columns in the database.

Get-SqlSensitivityRecommendations	Get recommended sensitivity labels and information types for columns in the database.
Get-SqlSmartAdmin	Gets the SQL Smart Admin object and its properties.
Grant-SqlAvailabilityGroupCreateAnyDatabase	Grants the CREATE ANY DATABASE permission to an Always On Availability Group.
Import-SqlVulnerabilityAssessmentBaselineSet	Imports a Vulnerability Assessment baseline set from a file.
Invoke-ASCmd	Enables database administrators to execute an XMLA script, TMSL script, Data Analysis Expressions (DAX) query, Multidimensional Expressions (MDX) query, or Data Mining Extensions (DMX) statement against an instance of Analysis Services.
Invoke-PolicyEvaluation	Invokes one or more SQL Server policy-based management policy evaluations.
Invoke-ProcessASDatabase	Conducts the Process operation on a specified Database with a specific ProcessType or RefreshType depending on the underlying metadata type.
Invoke-ProcessCube	Conducts the Process operation on a specified Cube of a specific database with a specific ProcessType value.
Invoke-ProcessDimension	Conducts the Process operation on a specified Cube of a specific database with a specific ProcessType value.
Invoke-ProcessPartition	Conducts the Process operation on a specific Partition of a specific database having a specific Cube name and a MeasureGroup name with a specific ProcessType value.
Invoke-ProcessTable	Conducts the Process operation on a specified Table with a specific RefreshType.
Invoke-SqlAssessment	Runs SQL Assessment best practice checks for a chosen SQL Server object and returns their results.
Invoke-Sqlcmd	Runs a script containing statements supported by the SQL Server SQLCMD utility.
Invoke-SqlColumnMasterKeyRotation	Initiates the rotation of a column master key.
Invoke-SqlNotebook	Executes a SQL Notebook file (.ipynb) and outputs the materialized notebook.
Invoke-SqlVulnerabilityAssessmentScan	Invokes a new Vulnerability Assessment scan.
Join-SqlAvailabilityGroup	Joins the local secondary replica to an availability group.

Merge-Partition	This cmdlet merges the data of one or more source partitions into a target partition and deletes the source partitions.
New-RestoreFolder	Restores an original folder to a new folder.
New-RestoreLocation	Used to add a restore location to the server.
New-SqlAvailabilityGroup	Creates an availability group.
New-SqlAvailabilityGroupListener	Creates an availability group listener and attaches it to an availability group.
New-SqlAvailabilityReplica	Creates an availability replica.
New-SqlAzureKeyVaultColumnMasterKeySettings	Creates a SqlColumnMasterKeySettings object describing an asymmetric key stored in Azure Key Vault.
New-SqlBackupEncryptionOption	Creates the encryption options for the **Backup-SqlDatabase** cmdlet or the **Set-SqlSmartAdmin** cmdlet.
New-SqlCertificateStoreColumnMasterKeySettings	Creates a SqlColumnMasterKeySettings object referencing the specified certificate.
New-SqlCngColumnMasterKeySettings	Creates a **SqlColumnMasterKeySettings** object describing an asymmetric key stored in a key store supporting the CNG API.
New-SqlColumnEncryptionKey	Crates a column encryption key object in the database.
New-SqlColumnEncryptionKeyEncryptedValue	Creates the encrypted value of a column encryption key.
New-SqlColumnEncryptionSettings	Creates a SqlColumnEncryptionSettings object that encapsulates information about a single column's encryption, including CEK and encryption type.
New-SqlColumnMasterKey	Creates a column master key object in the database.
New-SqlColumnMasterKeySettings	Creates a SqlColumnMasterKeySettings object describing a master key stored in an arbitrarily specified key store provider and path.
New-SqlCredential	Creates a SQL Server credential object.
New-SqlCspColumnMasterKeySettings	Creates a SqlColumnMasterKeySettings object describing an asymmetric key stored in a key store with a CSP supporting CAPI.
New-SqlHADREndpoint	Creates a database mirroring endpoint on a SQL Server instance.

New-SqlVulnerabilityAssessmentBaseline	Creates a new instance of Microsoft.SQL.VulnerabilityAssessment.SecurityCheckBaseline.
New-SqlVulnerabilityAssessmentBaselineSet	Creates a new instance of Microsoft.SQL.VulnerabilityAssessment.SecurityCheckBaselineSet.
Read-SqlTableData	Reads data from a table of a SQL database.
Read-SqlViewData	Reads data from a view of a SQL database.
Read-SqlXEvent	Reads SQL Server XEvents from XEL file or live SQL XEvent session.
Remove-RoleMember	Removes a member from the specific Role of a specific database.
Remove-SqlAvailabilityDatabase	Removes an availability database from its availability group.
Remove-SqlAvailabilityGroup	Removes an availability group.
Remove-SqlAvailabilityReplica	Removes a secondary availability replica.
Remove-SqlColumnEncryptionKey	Removes the column encryption key object from the database.
Remove-SqlColumnEncryptionKeyValue	Removes an encrypted value from an existing column encryption key object in the database.
Remove-SqlColumnMasterKey	Removes the column master key object from the database.
Remove-SqlCredential	Removes the SQL credential object.
Remove-SqlFirewallRule	Disables the Windows Firewall rule that allows connections to a specific instance of SQL Server.
Remove-SqlLogin	Removes Login objects from an instance of SQL Server.
Remove-SqlSensitivityClassification	Remove the sensitivity label and/or information type of columns in the database.
Restore-ASDatabase	Restores a specified Analysis Service database from a backup file.
Restore-SqlDatabase	Restores a database from a backup or transaction log records.
Resume-SqlAvailabilityDatabase	Resumes data movement on an availability database.
Revoke-SqlAvailabilityGroupCreateAnyDatabase	Revokes the CREATE ANY DATABASE permission on an Always On Availability Group.
Save-SqlMigrationReport	Generates In-Memory OLTP Migration Checklist
Set-SqlAuthenticationMode	Configures the authentication mode of the target instance of SQL Server.

Set-SqlAvailabilityGroup	Sets settings on an availability group.
Set-SqlAvailabilityGroupListener	Sets the port setting on an availability group listener.
Set-SqlAvailabilityReplica	Sets the settings on an availability replica.
Set-SqlAvailabilityReplicaRoleToSecondary	Sets the Availability Group replica role to secondary.
Set-SqlColumnEncryption	Encrypts, decrypts, or re-encrypts specified columns in the database.
Set-SqlCredential	Sets the properties for the SQL Credential object.
Set-SqlErrorLog	Sets or resets the maximum number of error log files before they are recycled.
Set-SqlHADREndpoint	Sets the properties of a database mirroring endpoint.
Set-SqlNetworkConfiguration	Sets the network configuration of the target instance of SQL Server.
Set-SqlSensitivityClassification	Set the information type and/or sensitivity label and information type of columns in the database.
Set-SqlSmartAdmin	Configures or modifies backup retention and storage settings.
Start-SqlInstance	Starts the specified instance of SQL Server.
Stop-SqlInstance	Stops the specified instance of SQL Server.
Suspend-SqlAvailabilityDatabase	Suspends data movement on an availability database.
Switch-SqlAvailabilityGroup	Starts a failover of an availability group to a secondary replica.
Test-SqlAvailabilityGroup	Evaluates the health of an availability group.
Test-SqlAvailabilityReplica	Evaluates the health of availability replicas.
Test-SqlDatabaseReplicaState	Evaluates the health of an availability database.
Test-SqlSmartAdmin	Tests the health of Smart Admin by evaluating SQL Server policy based management (PBM) policies.
Write-SqlTableData	Writes data to a table of a SQL database.