eBook Gratuit

APPRENEZ Microsoft SQL Server

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#sql-server

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: microsoft-sql-server

It is an unofficial and free Microsoft SQL Server ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official Microsoft SQL Server.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec Microsoft SQL Server

Remarques

Ceci est un ensemble d'exemples soulignant l'utilisation de base de SQL Server.

Versions

Version	Date de sortie
SQL Server 2016	2016-06-01
SQL Server 2014	2014-03-18
SQL Server 2012	2011-10-11
SQL Server 2008 R2	2010-04-01
SQL Server 2008	2008-08-06
SQL Server 2005	2005-11-01
SQL Server 2000	2000-11-01

Examples

INSERT / SELECT / UPDATE / DELETE: les bases du langage de manipulation de données

D ata **M** anipulation **L** anguage (DML en abrégé) inclut des opérations telles que `INSERT` , `UPDATE` et `DELETE` :

```
-- Create a table HelloWorld

CREATE TABLE HelloWorld (
    Id INT IDENTITY,
    Description VARCHAR(1000)
)


-- DML Operation INSERT, inserting a row into the table
INSERT INTO HelloWorld (Description) VALUES ('Hello World')


-- DML Operation SELECT, displaying the table
```

```
SELECT * FROM HelloWorld


-- Select a specific column from table
SELECT Description FROM HelloWorld


-- Display number of records in the table
SELECT Count(*) FROM HelloWorld


-- DML Operation UPDATE, updating a specific row in the table
UPDATE HelloWorld SET Description = 'Hello, World!' WHERE Id = 1


-- Selecting rows from the table (see how the Description has changed after the update?)
SELECT * FROM HelloWorld


-- DML Operation - DELETE, deleting a row from the table
DELETE FROM HelloWorld WHERE Id = 1


-- Selecting the table. See table content after DELETE operation
SELECT * FROM HelloWorld
```

Dans ce script, nous **créons un tableau** pour illustrer certaines requêtes de base.

Les exemples suivants montrent comment **interroger des tables:**

```
USE Northwind;
GO
SELECT TOP 10 * FROM Customers
ORDER BY CompanyName
```

sélectionnera les 10 premiers enregistrements de la table `Customer` , classés par la colonne `CompanyName` de la base de données `Northwind` (qui est l'une des bases de données exemple de Microsoft, téléchargeable **ici**):

CustomerID	CompanyName	ContactName	ContactTitle	Address	City
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57	Berlin
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la Constitución 2222	México D.F.
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 2312	México D.F.
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover Sq.	London
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägen 8	Luleå
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57	Mannheim
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléber	Strasbourg
BOLID	Bólido Comidas preparadas	Martín Sommer	Owner	C/ Araquil, 67	Madrid
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bouchers	Marseille
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen Blvd.	Tsawassen

Notez que `Use Northwind;` modifie la base de données par défaut pour toutes les requêtes suivantes. Vous pouvez toujours référencer la base de données en utilisant la syntaxe complète sous la forme [Base de données]. [Schéma]. [Table]:

```
SELECT TOP 10 * FROM Northwind.dbo.Customers
ORDER BY CompanyName

SELECT TOP 10 * FROM Pubs.dbo.Authors
ORDER BY City
```

Ceci est utile si vous interrogez des données provenant de différentes bases de données. Notez que `dbo` , spécifié "in between" est appelé un schéma et doit être spécifié en utilisant la syntaxe complète. Vous pouvez le considérer comme un dossier dans votre base de données. `dbo` est le schéma par défaut. Le schéma par défaut peut être omis. Tous les autres schémas définis par l'utilisateur doivent être spécifiés.

Si la table de base de données contient des colonnes nommées comme des mots réservés, par exemple `Date` , vous devez inclure le nom de la colonne entre parenthèses, comme ceci:

```
-- descending order
SELECT TOP 10 [Date] FROM dbo.MyLogTable
ORDER BY [Date] DESC
```

La même chose s'applique si le nom de la colonne contient des espaces dans son nom (ce qui n'est pas recommandé). Une syntaxe alternative consiste à utiliser des guillemets au lieu de crochets, par exemple:

```
-- descending order
SELECT top 10 "Date" from dbo.MyLogTable
order by "Date" desc
```

est équivalent mais pas si couramment utilisé. Notez la différence entre guillemets et guillemets simples: les guillemets simples sont utilisés pour les chaînes, c.-à-d.

```
-- descending order
SELECT top 10 "Date" from dbo.MyLogTable
where UserId='johndoe'
order by "Date" desc
```

est une syntaxe valide. Notez que T-SQL a un préfixe `N` pour les types de données NChar et NVarchar, par exemple

```
SELECT TOP 10 * FROM Northwind.dbo.Customers
WHERE CompanyName LIKE N'AL%'
ORDER BY CompanyName
```

renvoie toutes les entreprises dont le nom de société commence par `AL` (`%` est un caractère générique, utilisez-le comme vous l'utiliseriez dans une ligne de commande DOS, par exemple `DIR AL*`). Pour `LIKE` , quelques jokers sont disponibles, regardez **ici** pour en savoir plus.

Joint

Les jointures sont utiles si vous souhaitez interroger des champs qui n'existent pas dans une

seule table, mais dans plusieurs tables. Par exemple: vous souhaitez interroger toutes les colonnes de la table `Region` dans la base de données `Northwind` . Mais vous remarquez que vous avez également besoin de `RegionDescription` , qui est stocké dans une autre table, `Region` . Cependant, il existe une clé commune, `RgionID` que vous pouvez utiliser pour combiner ces informations en une seule requête, comme suit (le `Top 5` renvoie simplement les 5 premières lignes, omet pour obtenir toutes les lignes):

```
SELECT TOP 5 Territories.*,
    Regions.RegionDescription
FROM Territories
INNER JOIN Region
    ON Territories.RegionID=Region.RegionID
ORDER BY TerritoryDescription
```

affichera toutes les colonnes des `Territories` plus la colonne `RegionDescription` de la `Region` . Le résultat est ordonné par `TerritoryDescription` .

Alias de table

Lorsque votre requête nécessite une référence à deux tables ou plus, il peut être utile d'utiliser un alias de table. Les alias de table sont des références abrégées à des tables pouvant être utilisées à la place d'un nom de table complet, ce qui peut réduire la saisie et la modification. La syntaxe pour utiliser un alias est la suivante:

```
<TableName> [as] <alias>
```

Où `as` est un mot - clé facultatif. Par exemple, la requête précédente peut être réécrite comme suit:

```
SELECT TOP 5 t.*,
    r.RegionDescription
FROM Territories t
INNER JOIN Region r
    ON t.RegionID = r.RegionID
ORDER BY TerritoryDescription
```

Les alias doivent être uniques pour toutes les tables d'une requête, même si vous utilisez la même table deux fois. Par exemple, si votre table Employee contient un champ SupervisorId, vous pouvez utiliser cette requête pour renvoyer un employé et le nom de son superviseur:

```
SELECT e.*,
    s.Name as SupervisorName -- Rename the field for output
FROM Employee e
INNER JOIN Employee s
    ON e.SupervisorId = s.EmployeeId
WHERE e.EmployeeId = 111
```

Les syndicats

Comme nous l'avons vu précédemment, une jointure ajoute des colonnes provenant de différentes sources de table. Mais que faire si vous souhaitez combiner des lignes provenant de différentes sources? Dans ce cas, vous pouvez utiliser un UNION. Supposons que vous planifiez une fête et que vous souhaitiez inviter non seulement les employés mais également les clients. Ensuite, vous pouvez exécuter cette requête pour le faire:

```
SELECT FirstName+' '+LastName as ContactName, Address, City FROM Employees
UNION
SELECT ContactName, Address, City FROM Customers
```

Il renverra les noms, adresses et villes des employés et des clients dans une seule table. Notez que les lignes en double (s'il y en a) sont automatiquement éliminées (si vous ne le souhaitez pas, utilisez plutôt `UNION ALL`). Le numéro de colonne, les noms de colonne, l'ordre et le type de données doivent correspondre à toutes les instructions select faisant partie de l'union. C'est pourquoi le premier SELECT combine `FirstName` et `LastName` d'Employé dans `ContactName` .

Variables de table

Cela peut être utile, si vous avez besoin de gérer des données temporaires (en particulier dans une procédure stockée), pour utiliser des variables de table: La différence entre une table "réelle" et une variable de table

Exemple:

```
DECLARE @Region TABLE
(
  RegionID int,
  RegionDescription NChar(50)
)
```

crée une table en mémoire. Dans ce cas, le préfixe `@` est obligatoire car il s'agit d'une variable. Vous pouvez effectuer toutes les opérations DML mentionnées ci-dessus pour insérer, supprimer et sélectionner des lignes, par exemple

```
INSERT INTO @Region values(3,'Northern')
INSERT INTO @Region values(4,'Southern')
```

Mais normalement, vous le rempliriez sur la base d'une vraie table comme

```
INSERT INTO @Region
SELECT * FROM dbo.Region WHERE RegionID>2;
```

qui lirait les valeurs filtrées de la table réelle `dbo.Region` et l'insérerait dans la table de mémoire `@Region` - où il pourrait être utilisé pour un traitement ultérieur. Par exemple, vous pouvez l'utiliser dans une jointure comme

```
SELECT * FROM Territories t
JOIN @Region r on t.RegionID=r.RegionID
```

qui dans ce cas retournerait tous `Southern` territoires `Southern Northern` et du `Southern` . Des informations plus détaillées peuvent être trouvées ici . Les tables temporaires sont discutées ici , si vous êtes intéressé pour en savoir plus sur ce sujet.

REMARQUE: Microsoft recommande uniquement l'utilisation de variables de table si le nombre de lignes de données de la variable de table est inférieur à 100. Si vous travaillez avec de plus grandes quantités de données, utilisez plutôt une **table temporaire** ou temporaire.

IMPRESSION

Afficher un message sur la console de sortie. À l'aide de SQL Server Management Studio, cela sera affiché dans l'onglet Messages plutôt que dans l'onglet Résultats:

```
PRINT 'Hello World!';
```

SÉLECTIONNEZ toutes les lignes et colonnes d'une table

Syntaxe:

```
SELECT *
FROM table_name
```

L'utilisation de l'opérateur astérisque `*` sert de raccourci pour sélectionner toutes les colonnes de la table. Toutes les lignes seront également sélectionnées car cette `SELECT` ne contient pas de clause `WHERE` , pour spécifier des critères de filtrage.

Cela fonctionnerait également de la même manière si vous ajoutiez un alias à la table, par exemple `e` dans ce cas:

```
SELECT *
FROM Employees AS e
```

Ou si vous voulez tout sélectionner dans une table spécifique, vous pouvez utiliser l'alias + ". *":

```
SELECT e.*, d.DepartmentName
FROM Employees AS e
    INNER JOIN Department AS d
        ON e.DepartmentID = d.DepartmentID
```

On peut également accéder aux objets de base de données en utilisant des noms complets:

```
SELECT * FROM [server_name].[database_name].[schema_name].[table_name]
```

Cela n'est pas nécessairement recommandé, car la modification des noms de serveur et / ou de base de données entraînerait la non-exécution des requêtes utilisant des noms qualifiés complets en raison de noms d'objet non valides.

Notez que les champs avant `table_name` peuvent être omis dans de nombreux cas si les requêtes

sont exécutées sur un seul serveur, une base de données et un schéma, respectivement. Cependant, il est fréquent qu'une base de données ait plusieurs schémas et, dans ce cas, le nom du schéma ne doit pas être omis lorsque cela est possible.

Avertissement: l' utilisation de `SELECT *` dans le code de production ou les procédures stockées peut entraîner des problèmes ultérieurs (à mesure que de nouvelles colonnes sont ajoutées à la table ou si des colonnes sont réorganisées dans la table), en particulier si votre code ou nombre de colonnes renvoyées. Il est donc plus sûr de toujours spécifier explicitement les noms de colonnes dans les instructions SELECT pour le code de production.

```
SELECT col1, col2, col3
FROM table_name
```

Sélectionnez des lignes correspondant à une condition

Généralement, la syntaxe est la suivante:

```
SELECT <column names>
FROM <table name>
WHERE <condition>
```

Par exemple:

```
SELECT FirstName, Age
FROM Users
WHERE LastName = 'Smith'
```

Les conditions peuvent être complexes:

```
SELECT FirstName, Age
FROM Users
WHERE LastName = 'Smith' AND (City = 'New York' OR City = 'Los Angeles')
```

UPDATE Ligne spécifique

```
UPDATE HelloWorlds
SET HelloWorld = 'HELLO WORLD!!!'
WHERE Id = 5
```

Le code ci-dessus met à jour la valeur du champ "HelloWorld" avec "HELLO WORLD !!!" pour l'enregistrement où "Id = 5" dans la table HelloWorlds.

Remarque: Dans une instruction de mise à jour, il est conseillé d’utiliser une clause "where" pour éviter de mettre à jour la totalité de la table, sauf si votre exigence est différente.

Mettre à jour toutes les lignes

Une simple forme de mise à jour consiste à incrémenter toutes les valeurs dans un champ donné

de la table. Pour ce faire, nous devons définir le champ et la valeur d'incrément

Voici un exemple qui incrémente le champ `Score` de 1 (dans toutes les lignes):

```
UPDATE Scores
SET score = score + 1
```

Cela peut être dangereux car vous pouvez corrompre vos données si vous faites accidentellement une mise à jour pour une **ligne spécifique** avec une ligne UPDATE pour **tous** dans la table.

Commentaires dans le code

Transact-SQL prend en charge deux formes d'écriture de commentaires. Les commentaires sont ignorés par le moteur de base de données et sont destinés aux utilisateurs.

Les commentaires sont précédés de `--` et sont ignorés jusqu'à ce qu'une nouvelle ligne soit rencontrée:

```
-- This is a comment
SELECT *
FROM MyTable -- This is another comment
WHERE Id = 1;
```

Les commentaires Slash Star commencent par `/*` et se terminent par `*/` . Tout le texte entre ces délimiteurs est considéré comme un bloc de commentaires.

```
/* This is
a multi-line
comment block. */
SELECT Id = 1, [Message] = 'First row'
UNION ALL
SELECT 2, 'Second row'
/* This is a one liner */
SELECT 'More';
```

Les commentaires en barre oblique ont l'avantage de garder le commentaire utilisable si l'instruction SQL perd de nouveaux caractères de ligne. Cela peut se produire lorsque SQL est capturé lors du dépannage.

Les commentaires Slash Star peuvent être imbriqués et un début `/*` dans un commentaire en barre oblique doit être terminé par un `*/` pour être valide. Le code suivant entraînera une erreur

```
/*
SELECT *
FROM CommentTable
WHERE Comment = '/*'
*/
```

L'étoile de barre oblique même si à l'intérieur de la citation est considérée comme le début d'un commentaire. Par conséquent, il doit être terminé avec une autre barre oblique finale. La bonne façon serait

```
/*
SELECT *
FROM CommentTable
WHERE Comment = '/*'
*/  */
```

Récupérer les informations de base du serveur

```
SELECT @@VERSION
```

Renvoie la version de MS SQL Server exécutée sur l'instance.

```
SELECT @@SERVERNAME
```

Renvoie le nom de l'instance MS SQL Server.

```
SELECT @@SERVICENAME
```

Renvoie le nom du service Windows sous lequel MS SQL Server s'exécute.

```
SELECT serverproperty('ComputerNamePhysicalNetBIOS');
```

Renvoie le nom physique de la machine sur laquelle SQL Server est exécuté. Utile pour identifier le noeud dans un cluster de basculement.

```
SELECT * FROM fn_virtualservernodes();
```

Dans un cluster de basculement, renvoie chaque noeud sur lequel SQL Server peut être exécuté. Il ne renvoie rien sinon un cluster.

Utilisation des transactions pour modifier les données en toute sécurité

Chaque fois que vous modifiez des données, dans une commande DML (Data Manipulation Language), vous pouvez envelopper vos modifications dans une transaction. DML inclut `UPDATE` , `TRUNCATE` , `INSERT` et `DELETE` . L'une des façons dont vous pouvez vous assurer que vous modifiez les bonnes données consiste à utiliser une transaction.

Les modifications DML prennent un verrou sur les lignes affectées. Lorsque vous commencez une transaction, vous devez mettre fin à la transaction ou tous les objets modifiés dans le DML resteront verrouillés par quiconque a commencé la transaction. Vous pouvez mettre fin à votre transaction avec `ROLLBACK` ou `COMMIT` . `ROLLBACK` renvoie tout ce qui se trouve dans la transaction à son état d'origine. `COMMIT` place les données dans un état final où vous ne pouvez pas annuler vos modifications sans une autre déclaration DML.

Exemple:

```
--Create a test table
```

```
USE [your database]
GO
CREATE TABLE test_transaction (column_1 varchar(10))
GO

INSERT INTO
 dbo.test_transaction
        ( column_1 )
VALUES
        ( 'a' )

BEGIN TRANSACTION --This is the beginning of your transaction

UPDATE dbo.test_transaction
SET column_1 = 'B'
OUTPUT INSERTED.*
WHERE column_1 = 'A'


ROLLBACK TRANSACTION  --Rollback will undo your changes
           --Alternatively, use COMMIT to save your results

SELECT * FROM dbo.test_transaction   --View the table after your changes have been run

DROP TABLE dbo.test_transaction
```

Remarques:

- Ceci est un **exemple simplifié** qui n'inclut pas la gestion des erreurs. Mais toute opération de base de données peut échouer et donc lancer une exception. **Voici un exemple de la** façon dont une gestion des erreurs requise peut se présenter. Vous ne devez **jamais** utiliser des transactions **sans gestionnaire d'erreurs** , sinon vous pourriez laisser la transaction dans un état inconnu.
- Selon le **niveau d'isolement** , les transactions placent des verrous sur les données interrogées ou modifiées. Vous devez vous assurer que les transactions ne sont pas exécutées pendant une longue période, car elles verrouillent les enregistrements dans une base de données et peuvent entraîner des **blocages** avec d'autres transactions parallèles. Gardez les opérations encapsulées dans les transactions aussi courtes que possible et minimisez l'impact avec la quantité de données que vous verrouillez.

SUPPRIMER toutes les lignes

```
DELETE
FROM Helloworlds
```

Cela supprimera toutes les données de la table. La table ne contiendra aucune ligne après avoir exécuté ce code. Contrairement à `DROP TABLE` , cela préserve la table et sa structure et vous pouvez continuer à insérer de nouvelles lignes dans cette table.

Une autre façon de supprimer toutes les lignes de la table est de la tronquer, comme suit:

```
TRUNCATE TABLE HelloWords
```

La différence avec l'opération DELETE est plusieurs:

1. L'opération de troncature ne stocke pas dans le fichier journal des transactions
2. S'il existe un champ `IDENTITY` , cela sera réinitialisé
3. TRUNCATE peut être appliqué sur une table entière et non sur une partie (avec la commande `DELETE` vous pouvez associer une clause `WHERE`)

Restrictions de TRUNCATE

1. Impossible de TRUNCATE une table s'il existe une référence `FOREIGN KEY`
2. Si la table participe à une vue `INDEXED VIEW`
3. Si la table est publiée en utilisant `TRANSACTIONAL REPLICATION` ou `MERGE REPLICATION`
4. Il ne déclenchera aucun TRIGGER défini dans le tableau

[sic]

TABLEAU TRUNCATE

```
TRUNCATE TABLE Helloworlds
```

Ce code supprimera toutes les données de la table Helloworlds. Le tableau tronqué est presque similaire au code `Delete from Table` . La différence est que vous ne pouvez pas utiliser les clauses where avec Truncate. La table tronquée est considérée comme meilleure que la suppression car elle utilise moins d'espaces de journal des transactions.

Notez que si une colonne d'identité existe, elle est réinitialisée à la valeur initiale (par exemple, l'ID auto-incrémenté redémarrera à partir de 1). Cela peut entraîner une incohérence si les colonnes d'identité sont utilisées comme clé étrangère dans une autre table.

Créer une nouvelle table et insérer des enregistrements de l'ancienne table

```
SELECT * INTO NewTable FROM OldTable
```

Crée une nouvelle table avec la structure de l'ancienne table et insère toutes les lignes dans la nouvelle table.

Quelques restrictions

1. Vous ne pouvez pas spécifier une variable de table ou un paramètre de valeur de table en tant que nouvelle table.
2. Vous ne pouvez pas utiliser SELECT… INTO pour créer une table partitionnée, même lorsque la table source est partitionnée. SELECT ... INTO n'utilise pas le schéma de partition de la table source; à la place, la nouvelle table est créée dans le groupe de fichiers par défaut. Pour insérer des lignes dans une table partitionnée, vous devez d'abord créer la table partitionnée, puis utiliser l'instruction INSERT INTO ... SELECT FROM.
3. Les index, les contraintes et les déclencheurs définis dans la table source ne sont pas transférés dans la nouvelle table. Ils ne peuvent pas non plus être

spécifiés dans l'instruction SELECT ... INTO. Si ces objets sont requis, vous pouvez les créer après avoir exécuté l'instruction SELECT ... INTO.

4. La spécification d'une clause ORDER BY ne garantit pas que les lignes sont insérées dans l'ordre spécifié. Lorsqu'une colonne fragmentée est incluse dans la liste de sélection, la propriété de colonne fragmentée n'est pas transférée dans la colonne de la nouvelle table. Si cette propriété est requise dans la nouvelle table, modifiez la définition de la colonne après avoir exécuté l'instruction SELECT ... INTO pour inclure cette propriété.
5. Lorsqu'une colonne calculée est incluse dans la liste de sélection, la colonne correspondante dans la nouvelle table n'est pas une colonne calculée. Les valeurs dans la nouvelle colonne sont les valeurs qui ont été calculées au moment où SELECT ... INTO a été exécuté.

[*sic*]

Obtenir le nombre de lignes de la table

L'exemple suivant peut être utilisé pour rechercher le nombre total de lignes d'une table spécifique dans une base de données si `table_name` est remplacé par la table que vous souhaitez interroger:

```
SELECT COUNT(*) AS [TotalRowCount] FROM table_name;
```

Il est également possible d'obtenir le nombre de lignes de toutes les tables en se joignant à la partition de la table basée sur l'index HEAP (index_id = 0) ou le cluster cluster (index_id = 1) des tables en utilisant le script suivant:

```
SELECT  [Tables].name                AS [TableName],
        SUM( [Partitions].[rows] )    AS [TotalRowCount]
FROM    sys.tables AS [Tables]
JOIN    sys.partitions AS [Partitions]
    ON  [Tables].[object_id]    =    [Partitions].[object_id]
    AND [Partitions].index_id IN ( 0, 1 )
--WHERE    [Tables].name = N'table name' /* uncomment to look for a specific table */
GROUP BY    [Tables].name;
```

Ceci est possible car chaque table est essentiellement une table de partition unique, à moins que des partitions supplémentaires ne lui soient ajoutées. Ce script a également l'avantage de ne pas interférer avec les opérations de lecture / écriture sur les lignes des tables.

Lire Démarrer avec Microsoft SQL Server en ligne: https://riptutorial.com/fr/sql-server/topic/236/demarrer-avec-microsoft-sql-server

Chapitre 2: Analyser une requête

Examples

Scan vs Seek

Lorsque vous visualisez un plan d'exécution, vous pouvez constater que SQL Server a décidé d'effectuer une recherche ou une analyse.

Une recherche se produit lorsque SQL Server sait où il doit aller et ne récupère que des éléments spécifiques. Cela se produit généralement lorsque de bons filtres sont placés dans une requête, par exemple `where name = 'Foo'`.

Une analyse est lorsque SQL Server ne sait pas exactement où toutes les données dont il a besoin, ou a décidé que l'analyse serait plus efficace qu'un Seek si suffisamment de données est sélectionné.

Les recherches sont généralement plus rapides puisqu'elles ne capturent qu'une sous-section des données, alors que les analyses sélectionnent la majorité des données.

Lire Analyser une requête en ligne: https://riptutorial.com/fr/sql-server/topic/7713/analyser-une-requete

Chapitre 3: Autorisations de base de données

Remarques

Syntaxe de base:

```
{GRANT| REVOKE | DENY} {PERMISSION_NAME} [ON {SECURABLE}] TO {PRINCIPAL};
```

- {GRANT | REVOKE | DENY} - Ce que vous essayez d'accomplir
  - Grant: "Donnez cette permission au principal déclaré"
  - Révoquer: "Prenez cette permission du principal déclaré"
  - Refuser: "Assurez-vous que le principal déclaré ne dispose jamais de cette permission (par exemple," `DENY SELECT` "signifie que indépendamment de toute autre autorisation, `SELECT` échouera pour ce principal)
- PERMISSION_NAME - L'opération que vous essayez d'affecter. Cela dépendra de la sécurisable. Par exemple, il n'est pas logique d'accorder `GRANT SELECT` sur une procédure stockée.
- SECURABLE - Le nom de la chose sur laquelle vous essayez d'affecter des autorisations. Ceci est *facultatif* . Dire `GRANT SELECT TO [aUser];` est parfaitement acceptable cela signifie "pour toute opération sécurisable pour laquelle l'autorisation `SELECT` un sens, `GRANT` cette autorisation".
- PRINCIPAL - Pour qui vous essayez d'affecter des autorisations. Au niveau de la base de données, cela peut être un rôle (application ou base de données) ou un utilisateur (mappé ou non sur un login) par exemple.

Examples

Modification des autorisations

```
GRANT SELECT ON [dbo].[someTable] TO [aUser];

REVOKE SELECT ON [dbo].[someTable] TO [aUser];
--REVOKE SELECT [dbo].[someTable] FROM [aUser]; is equivalent

DENY SELECT ON [dbo].[someTable] TO [aUser];
```

CRÉER UN UTILISATEUR

```
--implicitly map this user to a login of the same name as the user
CREATE USER [aUser];

--explicitly mapping what login the user should be associated with
CREATE USER [aUser] FOR LOGIN [aUser];
```

CREER UN RÔLE

```
CREATE ROLE [myRole];
```

Modification de l'appartenance à un rôle

```
-- SQL 2005+
exec sp_addrolemember @rolename = 'myRole', @membername = 'aUser';
exec sp_droprolemember @rolename = 'myRole', @membername = 'aUser';

-- SQL 2008+
ALTER ROLE [myRole] ADD MEMBER [aUser];
ALTER ROLE [myRole] DROP MEMBER [aUser];
```

Remarque: les membres de rôle peuvent être n'importe quel principal au niveau de la base de données. En d'autres termes, vous pouvez ajouter un rôle en tant que membre dans un autre rôle. De plus, l'ajout / suppression de membres de rôle est idempotent. En d'autres termes, si vous essayez d'ajouter / de supprimer, vous vous retrouverez avec / sans (respectivement) dans le rôle, quel que soit l'état actuel de l'appartenance à un rôle.

Lire Autorisations de base de données en ligne: https://riptutorial.com/fr/sql-server/topic/6788/autorisations-de-base-de-donnees

Chapitre 4: Autorisations et sécurité

Examples

Attribuer des autorisations d'objet à un utilisateur

Dans la production, il convient de sécuriser vos données et de ne permettre que des opérations sur celles-ci via des procédures stockées. Cela signifie que votre application ne peut pas exécuter directement les opérations CRUD sur vos données et peut causer des problèmes. L'attribution des autorisations est une tâche fastidieuse, fastidieuse et généralement onéreuse. Pour cette raison, il est souvent plus facile d'exploiter une partie de la puissance (considérable) contenue dans le schéma INFORMATION_SCHEMA qui est contenu dans chaque base de données SQL Server.

Au lieu d'affecter individuellement des autorisations à un utilisateur, exécutez simplement le script ci-dessous, copiez le résultat et exécutez-le dans une fenêtre de requête.

```
SELECT 'GRANT EXEC ON core.' + r.ROUTINE_NAME + ' TO ' + <MyDatabaseUsername>
FROM INFORMATION_SCHEMA.ROUTINES r
WHERE r.ROUTINE_CATALOG = '<MyDataBaseName>'
```

Lire Autorisations et sécurité en ligne: https://riptutorial.com/fr/sql-server/topic/7929/autorisations-et-securite

Chapitre 5: Avec option liens

Examples

Données de test

```
CREATE TABLE #TEST
(
Id INT,
Name VARCHAR(10)
)

Insert Into #Test
select 1,'A'
Union All
Select 1,'B'
union all
Select 1,'C'
union all
Select 2,'D'
```

Voici la sortie du tableau ci-dessus, Comme vous pouvez le voir, la colonne Id est répétée trois fois.

```
Id   Name
1    A
1    B
1    C
2    D
```

Maintenant, permet de vérifier la sortie en utilisant un ordre simple par ..

```
Select Top (1)  Id,Name From
#test
Order By Id ;
```

Sortie: (La sortie de la requête ci-dessus n'est pas garantie à chaque fois)

```
Id   Name
1    B
```

Permet d'exécuter la même requête avec l'option liens

```
Select Top (1) With Ties Id,Name
 From
#test
Order By Id
```

Sortie:

```
Id   Name
1    A
1    B
1    C
```

Comme vous pouvez le voir, SQL Server affiche toutes les lignes **liées à** Order by Column. Voyons un autre exemple pour mieux comprendre cela.

```
Select Top (1) With Ties Id,Name
 From
#test
Order By Id ,Name
```

Sortie:

```
Id   Name
1    A
```

En résumé, lorsque nous utilisons l'option liens, SQL Server génère toutes les lignes liées, quelle que soit la limite imposée.

Lire Avec option liens en ligne: https://riptutorial.com/fr/sql-server/topic/2546/avec-option-liens

Chapitre 6: Base de données système - TempDb

Examples

Identifier l'utilisation de TempDb

La requête suivante fournira des informations sur l'utilisation de TempDb. En analysant les décomptes, vous pouvez identifier la chose qui a un impact sur TempDb

```
SELECT
 SUM (user_object_reserved_page_count)*8 as usr_obj_kb,
 SUM (internal_object_reserved_page_count)*8 as internal_obj_kb,
 SUM (version_store_reserved_page_count)*8  as version_store_kb,
 SUM (unallocated_extent_page_count)*8 as freespace_kb,
 SUM (mixed_extent_page_count)*8 as mixedextent_kb
FROM sys.dm_db_file_space_usage
```

Attribute	Meaning
Higher number of user objects	More usage of Temp tables , cursors or temp variables
Higher number of internal objects	Query plan is using a lot of database. Ex: sorting, Group by etc.
Higher number of version stores	Long running transaction or high transaction throughput

Détails de la base de données TempDB

La requête ci-dessous peut être utilisée pour obtenir les détails de la base de données TempDB:

```
USE [MASTER]
SELECT * FROM sys.databases WHERE database_id = 2
```

OU

```
USE [MASTER]
SELECT * FROM sys.master_files WHERE database_id = 2
```

Avec l'aide de DMV ci-dessous, vous pouvez vérifier la quantité d'espace TempDb utilisée par votre session. Cette requête est très utile lors du débogage des problèmes TempDb

```
SELECT * FROM sys.dm_db_session_space_usage WHERE session_id = @@SPID
```

Lire Base de données système - TempDb en ligne: https://riptutorial.com/fr/sql-server/topic/4427/base-de-donnees-systeme---tempdb

Chapitre 7: bcp (programme de copie en bloc) Utilitaire

Introduction

L'utilitaire de copie en bloc (bcp) en bloc copie les données entre une instance de Microsoft SQL Server et un fichier de données dans un format spécifié par l'utilisateur. L'utilitaire bcp peut être utilisé pour importer un grand nombre de nouvelles lignes dans des tables SQL Server ou pour exporter des données hors des tables dans des fichiers de données.

Examples

Exemple d'importation de données sans fichier de format (en utilisant le format natif)

```
REM Truncate table (for testing)
SQLCMD -Q "TRUNCATE TABLE TestDatabase.dbo.myNative;"

REM Import data
bcp TestDatabase.dbo.myNative IN D:\BCP\myNative.bcp -T -n

REM Review results
SQLCMD -Q "SELECT * FROM TestDatabase.dbo.myNative;"
```

Lire bcp (programme de copie en bloc) Utilitaire en ligne: https://riptutorial.com/fr/sql-server/topic/10942/bcp--programme-de-copie-en-bloc--utilitaire

Chapitre 8: BULK Import

Examples

INSERT EN VRAC avec options

Vous pouvez personnaliser les règles d'analyse à l'aide de différentes options dans la clause WITH:

```
BULK INSERT People
FROM 'f:\orders\people.csv'
WITH  (  CODEPAGE = '65001',
         FIELDTERMINATOR =',',
         ROWTERMINATOR ='\n'
      );
```

Dans cet exemple, CODEPAGE spécifie qu'un fichier source dans le fichier UTF-8 et les TERMINATORS sont coma et nouvelle ligne.

INSERT EN VRAC

La commande BULK INSERT peut être utilisée pour importer un fichier dans SQL Server:

```
BULK INSERT People
FROM 'f:\orders\people.csv'
```

La commande BULK INSERT mappe les colonnes dans les fichiers avec des colonnes dans la table cible.

Lecture du contenu entier du fichier en utilisant OPENROWSET (BULK)

Vous pouvez lire le contenu du fichier en utilisant la fonction OPENROWSET (BULK) et stocker le contenu dans une table:

```
INSERT INTO myTable(content)
   SELECT BulkColumn
          FROM OPENROWSET(BULK N'C:\Text1.txt', SINGLE_BLOB) AS Document;
```

L'option SINGLE_BLOB lit le contenu entier d'un fichier en tant que cellule unique.

Lire le fichier en utilisant OPENROWSET (BULK) et formater le fichier

Yu peut définir le format du fichier qui sera importé à l'aide de l'option FORMATFILE:

```
INSERT INTO mytable
SELECT a.*
FROM OPENROWSET(BULK 'c:\test\values.txt',
```

```
   FORMATFILE = 'c:\test\values.fmt') AS a;
```

Le fichier de format, format_file.fmt, décrit les colonnes de values.txt:

```
9.0
2
1  SQLCHAR  0  10 "\t"        1  ID                SQL_Latin1_General_Cp437_BIN
2  SQLCHAR  0  40 "\r\n"      2  Description       SQL_Latin1_General_Cp437_BIN
```

Lire le fichier json avec OPENROWSET (BULK)

Vous pouvez utiliser OPENROWSET pour lire le contenu du fichier et le transmettre à une autre fonction qui analysera les résultats.

L'exemple suivant montre comment lire tout le contenu du fichier JSON à l'aide d'OPENROWSET (BULK), puis fournir la fonction BulkColumn à OPENJSON pour analyser les colonnes JSON et retourner:

```
SELECT book.*
 FROM OPENROWSET (BULK 'C:\JSON\Books\books.json', SINGLE_CLOB) as j
 CROSS APPLY OPENJSON(BulkColumn)
       WITH( id nvarchar(100), name nvarchar(100), price float,
             pages int, author nvarchar(100)) AS book
```

Lire BULK Import en ligne: https://riptutorial.com/fr/sql-server/topic/7330/bulk-import

Chapitre 9: Clause OVER

Paramètres

Paramètre	Détails
PARTITION PAR	Le champ qui suit PARTITION BY est celui sur lequel le «groupement» sera basé

Remarques

La clause OVER détermine une fenêtre ou un sous-ensemble de ligne dans un ensemble de résultats de requête. Une fonction de fenêtre peut être appliquée pour définir et calculer une valeur pour chaque ligne de l'ensemble. La clause OVER peut être utilisée avec:

- Fonctions de classement
- Fonctions d'agrégat

Ainsi, une personne peut calculer des valeurs agrégées telles que des moyennes mobiles, des agrégats cumulatifs, des totaux cumulés ou les N meilleurs résultats par groupe.

De manière très abstraite, nous pouvons dire que OVER se comporte comme GROUP BY. Cependant, OVER est appliqué par champ / colonne et non à la requête dans son ensemble, comme le fait GROUP BY.

Remarque # 1: Dans SQL Server 2008 (R2), ORDER BY Clause ne peut pas être utilisé avec les fonctions de fenêtre d'agrégat (lien).

Examples

Utilisation des fonctions d'agrégation avec OVER

En utilisant le tableau des voitures , nous calculons le montant total, maximum, minimum et moyen que chaque client a dépensé et reçu plusieurs fois (NOMBRE). Elle a apporté une voiture pour la réparation.

Id CustomerId MechanicId Statut du modèle Coût total

```
SELECT CustomerId,
       SUM(TotalCost) OVER(PARTITION BY CustomerId) AS Total,
       AVG(TotalCost) OVER(PARTITION BY CustomerId) AS Avg,
       COUNT(TotalCost) OVER(PARTITION BY CustomerId) AS Count,
       MIN(TotalCost) OVER(PARTITION BY CustomerId) AS Min,
       MAX(TotalCost) OVER(PARTITION BY CustomerId) AS Max
  FROM CarsTable
 WHERE Status = 'READY'
```

Attention, l'utilisation de OVER de cette manière ne regroupera pas les lignes renvoyées. La requête ci-dessus renverra les éléments suivants:

N ° de client	Total	Avg	Compter	Min	Max
1	430	215	2	200	230
1	430	215	2	200	230

La ou les lignes dupliquées peuvent ne pas être utiles à des fins de reporting.

Si vous souhaitez simplement agréger des données, il est préférable d'utiliser la clause GROUP BY avec les fonctions d'agrégat appropriées. Par exemple:

```
SELECT CustomerId,
       SUM(TotalCost) AS Total,
       AVG(TotalCost) AS Avg,
       COUNT(TotalCost) AS Count,
       MIN(TotalCost) AS Min,
       MAX(TotalCost)  AS Max
  FROM CarsTable
 WHERE Status = 'READY'
GROUP BY CustomerId
```

Somme cumulée

À l'aide du tableau des ventes d'articles , nous essaierons de savoir comment les ventes de nos articles augmentent au fil des dates. Pour ce faire, nous calculerons la *somme cumulée* des ventes totales par commande d'articles par date de vente.

```
SELECT item_id, sale_Date
       SUM(quantity * price) OVER(PARTITION BY item_id ORDER BY sale_Date ROWS BETWEEN
UNBOUNDED PRECEDING) AS SalesTotal
  FROM SalesTable
```

Utilisation des fonctions d'agrégation pour trouver les enregistrements les plus récents

En utilisant la base de données de la bibliothèque , nous essayons de trouver le dernier livre ajouté à la base de données pour chaque auteur. Pour cet exemple simple, nous supposons un identifiant toujours incrémenté pour chaque enregistrement ajouté.

```
SELECT MostRecentBook.Name, MostRecentBook.Title
FROM ( SELECT Authors.Name,
              Books.Title,
              RANK() OVER (PARTITION BY Authors.Id ORDER BY Books.Id DESC) AS NewestRank
       FROM Authors
       JOIN Books ON Books.AuthorId = Authors.Id
     ) MostRecentBook
WHERE MostRecentBook.NewestRank = 1
```

Au lieu de RANK, deux autres fonctions peuvent être utilisées pour commander. Dans l'exemple précédent, le résultat sera le même, mais ils donnent des résultats différents lorsque le classement donne plusieurs lignes pour chaque rang.

- `RANK()` : les doublons ont le même rang, le rang suivant prend en compte le nombre de doublons du rang précédent
- `DENSE_RANK()` : les doublons ont le même rang, le rang suivant est toujours supérieur au précédent
- `ROW_NUMBER()` : donnera à chaque ligne un "rang" unique, "classant" les doublons de manière aléatoire

Par exemple, si la table comportait une colonne non unique CreationDate et que le classement a été effectué sur cette base, la requête suivante:

```
SELECT Authors.Name,
       Books.Title,
       Books.CreationDate,
       RANK() OVER (PARTITION BY Authors.Id ORDER BY Books.CreationDate DESC) AS RANK,
       DENSE_RANK() OVER (PARTITION BY Authors.Id ORDER BY Books.CreationDate DESC) AS
DENSE_RANK,
       ROW_NUMBER() OVER (PARTITION BY Authors.Id ORDER BY Books.CreationDate DESC) AS
ROW_NUMBER,
FROM Authors
JOIN Books ON Books.AuthorId = Authors.Id
```

Pourrait entraîner:

Auteur	Titre	Date de création	RANG	DENSE_RANK	ROW_NUMBER
Auteur 1	Livre 1	22/07/2016	1	1	1
Auteur 1	Livre 2	22/07/2016	1	1	2
Auteur 1	Livre 3	21/07/2016	3	2	3
Auteur 1	Livre 4	21/07/2016	3	2	4
Auteur 1	Livre 5	21/07/2016	3	2	5
Auteur 1	Livre 6	04/07/2016	6	3	6
Auteur 2	Livre 7	04/07/2016	1	1	1

Division de données en compartiments à partition égale dans NTILE

Disons que vous avez des notes d'examen pour plusieurs examens et que vous souhaitez les diviser en quartiles par examen.

```
-- Setup data:
declare @values table(Id int identity(1,1) primary key, [Value] float, ExamId int)
```

```
insert into @values ([Value], ExamId) values
(65, 1), (40, 1), (99, 1), (100, 1), (90, 1), -- Exam 1 Scores
(91, 2), (88, 2), (83, 2), (91, 2), (78, 2), (67, 2), (77, 2) -- Exam 2 Scores

-- Separate into four buckets per exam:
select ExamId,
       ntile(4) over (partition by ExamId order by [Value] desc) as Quartile,
       Value, Id
from @values
order by ExamId, Quartile
```

	ExamId	Quartile	Value	Id
1	1	1	100	4
2	1	1	99	3
3	1	2	90	5
4	1	3	65	1
5	1	4	40	2
6	2	1	91	9
7	2	1	91	6
8	2	2	88	7
9	2	2	83	8
10	2	3	78	10
11	2	3	77	12
12	2	4	67	11

`ntile` fonctionne très bien lorsque vous avez vraiment besoin d'un nombre défini de compartiments et que chacun est rempli à peu près au même niveau. Notez qu'il serait trivial de séparer ces scores en percentiles en utilisant simplement `ntile(100)` .

Lire Clause OVER en ligne: https://riptutorial.com/fr/sql-server/topic/353/clause-over

Chapitre 10: Clés étrangères

Examples

Relation / contrainte de clé étrangère

Les clés étrangères vous permettent de définir la relation entre deux tables. Une table (parent) doit avoir une clé primaire qui identifie de manière unique les lignes de la table. Une autre table (enfant) peut avoir la valeur de la clé primaire du parent dans l'une des colonnes. La contrainte FOREIGN KEY REFERENCES garantit que les valeurs de la table enfant doivent exister en tant que valeur de clé primaire dans la table parent.

Dans cet exemple, nous avons la table de la société mère avec la clé primaire CompanyId et la table de l'employé enfant avec l'ID de la société où cet employé travaille.

```
create table Company (
   CompanyId int primary key,
   Name nvarchar(200)
)
create table Employee (
    EmployeeId int,
    Name nvarchar(200),
    CompanyId int
        foreign key references Company(companyId)
)
```

Les références de clé étrangère garantissent que les valeurs insérées dans la colonne Employee.CompanyId doivent également exister dans la colonne Company.CompanyId. En outre, personne ne peut supprimer la société dans la table de la société s'il ya au moins un employé avec un ID de société correspondant dans la table enfant.

La relation FOREIGN KEY garantit que les lignes de deux tables ne peuvent pas être "dissociées".

Maintien de la relation entre les lignes parent / enfant

Supposons que nous ayons une ligne dans la table Company avec companyId 1. Nous pouvons insérer une ligne dans la table employee avec companyId 1:

```
insert into Employee values (17, 'John', 1)
```

Cependant, nous ne pouvons pas insérer d'employé dont CompanyId n'existe pas:

```
insert into Employee values (17, 'John', 111111)
```

Msg 547, niveau 16, état 0, ligne 12 L'instruction INSERT était en conflit avec la contrainte FOREIGN KEY "FK__Employee__Compan__1EE485AA". Le conflit s'est produit dans la base de

données "MyDb", table "dbo.Company", colonne "CompanyId". La déclaration a été terminée.

En outre, nous ne pouvons pas supprimer la ligne parente dans la table de l'entreprise tant qu'il y a au moins une ligne enfant dans la table des employés qui la référence.

```
delete from company where CompanyId = 1
```

Msg 547, Niveau 16, État 0, Ligne 14 L'instruction DELETE était en conflit avec la contrainte REFERENCE "FK__Employee__Compan__1EE485AA". Le conflit s'est produit dans la base de données "MyDb", table "dbo.Employee", colonne "CompanyId". La déclaration a été terminée.

La relation de clé étrangère garantit que les lignes de la société et des employés ne seront pas "dissociées".

Ajout d'une relation de clé étrangère sur une table existante

La contrainte **FOREIGN KEY** peut être ajoutée aux tables existantes qui ne sont toujours pas en relation. Imaginez que nous ayons des tables Société et Employé où la colonne Employé de la table CompanyId n'a pas de relation de clé étrangère. L'instruction ALTER TABLE vous permet d'ajouter **une** contrainte de **clé étrangère** sur une colonne existante qui référence une autre table et une clé primaire dans cette table:

```
alter table Employee
    add  foreign key (CompanyId) references Company(CompanyId)
```

Ajouter une clé étrangère sur la table existante

Les colonnes **FOREIGN KEY** avec contrainte peuvent être ajoutées aux tables existantes qui ne sont toujours pas en relation. Imaginez que nous ayons des tables Company et Employee où la table Employee n'a pas de colonne CompanyId. L'instruction ALTER TABLE vous permet d'ajouter une nouvelle colonne avec **une** contrainte de **clé étrangère** qui référence une autre table et une clé primaire dans cette table:

```
alter table Employee
    add CompanyId int foreign key references Company(CompanyId)
```

Obtenir des informations sur les contraintes de clés étrangères

La vue système sys.foreignkeys renvoie des informations sur toutes les relations de clés étrangères dans la base de données:

```
select name,
 OBJECT_NAME(referenced_object_id) as [parent table],
 OBJECT_NAME(parent_object_id) as [child table],
 delete_referential_action_desc,
 update_referential_action_desc
from sys.foreign_keys
```

Lire Clés étrangères en ligne: https://riptutorial.com/fr/sql-server/topic/5355/cles-etrangeres

Chapitre 11: Clés primaires

Remarques

Les clés primaires sont utilisées pour identifier de manière unique un enregistrement dans une table. Une table ne peut avoir qu'une seule clé primaire (bien que la clé primaire puisse être composée de plusieurs colonnes) et une clé primaire est requise pour certains types de réplication.

Les clés primaires sont souvent utilisées (mais pas obligatoirement) dans l' index clusterisé d'une table.

Examples

Créer une table avec une colonne d'identité en tant que clé primaire

```
-- Identity primary key - unique arbitrary increment number
create table person (
id int identity(1,1) primary key not null,
firstName varchar(100) not null,
lastName varchar(100) not null,
dob DateTime not null,
ssn varchar(9) not null
)
```

Créer une table avec une clé primaire GUID

```
-- GUID primary key - arbitrary unique value for table
create table person (
id uniqueIdentifier default (newId()) primary key,
firstName varchar(100) not null,
lastName varchar(100) not null,
dob DateTime not null,
ssn varchar(9) not null
)
```

Créer une table avec une clé naturelle

```
-- natural primary key - using an existing piece of data within the table that uniquely
identifies the record
create table person (
firstName varchar(100) not null,
lastName varchar(100) not null,
dob DateTime not null,
ssn varchar(9) primary key not null
)
```

Créer une table avec une clé composite

```
-- composite key - using two or more existing columns within a table to create a primary key
create table person (
firstName varchar(100) not null,
lastName varchar(100) not null,
dob DateTime not null,
ssn varchar(9) not null,
primary key (firstName, lastName, dob)
)
```

Ajouter la clé primaire à la table existante

```
ALTER TABLE person
 ADD CONSTRAINT pk_PersonSSN PRIMARY KEY (ssn)
```

Remarque: si la colonne de clé primaire (dans ce cas, `ssn`) a plus d'une ligne avec la même clé candidate, l'instruction ci-dessus échouera, car les valeurs de clé primaire doivent être uniques.

Supprimer la clé primaire

```
ALTER TABLE Person
 DROP CONSTRAINT pk_PersonSSN
```

Lire Clés primaires en ligne: https://riptutorial.com/fr/sql-server/topic/4543/cles-primaires

Chapitre 12: Colonnes calculées

Examples

Une colonne est calculée à partir d'une expression

Une colonne calculée est calculée à partir d'une expression pouvant utiliser d'autres colonnes dans la même table. L'expression peut être un nom de colonne non calculé, une constante, une fonction et toute combinaison de ces éléments connectée par un ou plusieurs opérateurs.

Créer une table avec une colonne calculée

```
Create table NetProfit
(
    SalaryToEmployee            int,
    BonusDistributed            int,
    BusinessRunningCost         int,
    BusinessMaintenanceCost     int,
    BusinessEarnings            int,
    BusinessNetIncome
                As BusinessEarnings - (SalaryToEmployee          +
                                       BonusDistributed          +
                                       BusinessRunningCost       +
                                       BusinessMaintenanceCost    )

)
```

La valeur est calculée et stockée dans la colonne calculée automatiquement lors de l'insertion d'autres valeurs.

```
Insert Into NetProfit
    (SalaryToEmployee,
     BonusDistributed,
     BusinessRunningCost,
     BusinessMaintenanceCost,
     BusinessEarnings)
Values
    (1000000,
     10000,
     1000000,
     50000,
     2500000)
```

Exemple simple que nous utilisons normalement dans les tables de log

```
CREATE TABLE [dbo].[ProcessLog](
[LogId] [int] IDENTITY(1,1) NOT NULL,
[LogType] [varchar](20) NULL,
[StartTime] [datetime] NULL,
[EndTime] [datetime] NULL,
[RunMinutes]  AS
(datediff(minute,coalesce([StartTime],getdate()),coalesce([EndTime],getdate())))
```

Cela donne une différence de temps d'exécution en minutes pour l'exécution, ce qui sera très pratique.

Lire Colonnes calculées en ligne: https://riptutorial.com/fr/sql-server/topic/5561/colonnes-calculees

Chapitre 13: COLUMNSTORE CLUSTERED

Examples

Tableau avec index CLUSTERED COLUMNSTORE

Si vous voulez avoir une table organisée au format colonne au lieu du magasin de lignes, ajoutez INDEX cci CLUSTERED COLUMNSTORE dans la définition du tableau:

```
DROP TABLE IF EXISTS Product
GO
CREATE TABLE Product (
    ProductID int,
    Name nvarchar(50) NOT NULL,
    Color nvarchar(15),
    Size nvarchar(5) NULL,
    Price money NOT NULL,
    Quantity int,
    INDEX cci CLUSTERED COLUMNSTORE
)
```

Les tables COLUMSTORE sont meilleures pour les tables où vous attendez des analyses et des rapports complets, tandis que les tables de stockage en ligne sont meilleures pour les tables où vous allez lire ou mettre à jour des ensembles de lignes plus petits.

Ajout d'un index columnstore en cluster sur une table existante

CREATE CLUSTERED COLUMNSTORE INDEX vous permet d'organiser une table au format colonne:

```
DROP TABLE IF EXISTS Product
GO
CREATE TABLE Product (
    Name nvarchar(50) NOT NULL,
    Color nvarchar(15),
    Size nvarchar(5) NULL,
    Price money NOT NULL,
    Quantity int
)
GO
CREATE CLUSTERED COLUMNSTORE INDEX cci ON Product
```

Reconstruire l'index CLUSTERED COLUMNSTORE

L'index du magasin de colonnes en cluster peut être reconstruit si vous avez beaucoup de lignes supprimées:

```
ALTER INDEX cci ON Products
REBUILD PARTITION = ALL
```

La reconstruction de CLUSTERED COLUMNSTORE "rechargera" les données de la table en cours en une nouvelle et appliquera à nouveau la compression, supprimera les lignes supprimées, etc.

Vous pouvez reconstruire une ou plusieurs partitions.

Lire COLUMNSTORE CLUSTERED en ligne: https://riptutorial.com/fr/sql-server/topic/5774/columnstore-clustered

Chapitre 14: COMMANDÉ PAR

Remarques

La clause ORDER BY a pour objet de trier les données renvoyées par une requête.

Il est important de noter que l' **ordre des lignes renvoyées par une requête n'est pas défini, sauf s'il existe une clause ORDER BY.**

Voir la documentation MSDN pour plus de détails sur la clause ORDER BY:
https://msdn.microsoft.com/en-us/library/ms188385.aspx

Examples

Clause ORDER BY simple

À l'aide de la table Employees , vous trouverez ci-dessous un exemple pour renvoyer les colonnes Id, FName et LName dans l'ordre LName (croissant):

```
SELECT Id, FName, LName FROM Employees
ORDER BY LName
```

Résultats:

Id	FName	LName
2	John	Johnson
1	James	Forgeron
4	Johnathon	Forgeron
3	Michael	Williams

Pour trier par ordre décroissant, ajoutez le mot-clé DESC après le paramètre de champ, par exemple la même requête dans l'ordre décroissant LName est:

```
SELECT Id, FName, LName FROM Employees
ORDER BY LName DESC
```

COMMANDER PAR plusieurs champs

Plusieurs champs peuvent être spécifiés pour la clause `ORDER BY` , dans l'ordre ASCending ou DESCending.

Par exemple, à l'aide de la table http://stackoverflow.com/documentation/sql/280/example-

databases/1207/item-sales-table#t=20160721131406643421 1 , nous pouvons renvoyer une requête qui trie par SaleDate dans l'ordre croissant, et Quantité en ordre décroissant.

```
SELECT ItemId, SaleDate, Quantity
FROM [Item Sales]
ORDER BY SaleDate ASC, Quantity DESC
```

Notez que le mot-clé `ASC` est facultatif et que les résultats sont triés dans l'ordre croissant d'un champ donné par défaut.

ORDRE BY avec logique complexe

Si nous voulons commander les données différemment pour chaque groupe, nous pouvons ajouter une syntaxe `CASE` à la commande `ORDER BY` . Dans cet exemple, nous voulons commander les employés du département 1 par nom de famille et les employés du département 2 par salaire.

Id	FName	LName	Numéro de téléphone	ManagerId	DépartementId	Un salaire	Date d'embauche
1	James	Forgeron	1234567890	NUL	1	1000	01-01-2002
2	John	Johnson	2468101214	1	1	400	23-03-2005
3	Michael	Williams	1357911131	1	2	600	12-05-2009
4	Johnathon	Forgeron	1212121212	2	1	500	24-07-2016
5	Sam	saxon	1372141312	2	2	400	25-03-2015

```
The following query will provide the required results:
SELECT Id, FName, LName, Salary FROM Employees
ORDER BY Case When DepartmentId = 1 then LName else Salary end
```

Commande personnalisée

Si vous souhaitez passer une commande en utilisant une colonne autre que l'ordre alphabétique / numérique, vous pouvez utiliser la `case` pour spécifier l'ordre souhaité.

`order by Group` retourne:

Groupe	Compter
Non retiré	6
Retraité	4
Total	dix

```
order by case group when 'Total' then 1 when 'Retired' then 2 else 3 end
```

retours `order by case group when 'Total' then 1 when 'Retired' then 2 else 3 end` :

Groupe	Compter
Total	dix
Retraité	4
Non retiré	6

Lire COMMANDÉ PAR en ligne: https://riptutorial.com/fr/sql-server/topic/4149/commande-par

Chapitre 15: Conseils de requête

Examples

JOIN Hints

Lorsque vous joignez deux tables, l'optimiseur de requêtes SQL (QO) peut choisir différents types de jointures à utiliser dans la requête:

- HASH rejoindre
- LOOP rejoindre
- Joindre

QO explorera les plans et choisira l'opérateur optimal pour joindre les tables. Toutefois, si vous êtes certain de connaître l'opérateur de jointure optimal, vous pouvez spécifier le type de jointure à utiliser. La jointure LOOP interne obligera QO à choisir une jointure de boucle imbriquée en joignant deux tables:

```
select top 100 *
from Sales.Orders o
    inner loop join Sales.OrderLines ol
    on o.OrderID = ol.OrderID
```

jointure de fusion interne forcera l'opérateur de jointure MERGE:

```
select top 100 *
from Sales.Orders o
    inner merge join Sales.OrderLines ol
    on o.OrderID = ol.OrderID
```

jointure de hachage interne forcera l'opérateur de jointure HASH:

```
select top 100 *
from Sales.Orders o
    inner hash join Sales.OrderLines ol
    on o.OrderID = ol.OrderID
```

Astuces GROUP BY

Lorsque vous utilisez la clause GROUP BY, l'optimiseur de requête SQL (QO) peut choisir différents types d'opérateurs de regroupement:

- HASH Aggregate qui crée une carte de hachage pour le regroupement des entrées
- Stream Aggregate qui fonctionne bien avec les entrées pré-commandées

Vous pouvez explicitement exiger que QO choisisse un opérateur ou un autre opérateur agrégé si vous savez ce qui serait optimal. Avec OPTION (ORDER GROUP), QO choisira toujours l'agrégat

de flux et ajoutera l'opérateur de tri devant l'agrégat de flux si l'entrée n'est pas triée:

```
select OrderID, AVG(Quantity)
from Sales.OrderLines
group by OrderID
OPTION (ORDER GROUP)
```

Avec OPTION (HASH GROUP), QO choisira toujours l'agrégat Hash:

```
select OrderID, AVG(Quantity)
from Sales.OrderLines
group by OrderID
OPTION (HASH GROUP)
```

Lignes FAST

Indique que la requête est optimisée pour une récupération rapide du premier nombre_rows. Ceci est un entier non négatif. Une fois que les premières Number_rows sont renvoyées, la requête continue son exécution et produit son jeu de résultats complet.

```
select OrderID, AVG(Quantity)
from Sales.OrderLines
group by OrderID
OPTION (FAST 20)
```

Conseils UNION

Lorsque vous utilisez l'opérateur UNION sur deux résultats de requête, l'optimiseur de requêtes (QO) peut utiliser les opérateurs suivants pour créer une union de deux jeux de résultats:

- Fusionner (union)
- Concat (Union)
- Match de hachage (union)

Vous pouvez explicitement spécifier quel opérateur doit être utilisé avec l'option OPTION ():

```
select OrderID, OrderDate, ExpectedDeliveryDate, Comments
from Sales.Orders
where OrderDate > DATEADD(day, -1, getdate())
UNION
select PurchaseOrderID as OrderID, OrderDate, ExpectedDeliveryDate, Comments
from Purchasing.PurchaseOrders
where OrderDate > DATEADD(day, -1, getdate())
OPTION(HASH UNION)
-- or OPTION(CONCAT UNION)
-- or OPTION(MERGE UNION)
```

Option MAXDOP

Spécifie le degré de parallélisme maximal pour la requête spécifiant cette option.

```
SELECT OrderID,
    AVG(Quantity)
FROM Sales.OrderLines
GROUP BY OrderID
OPTION (MAXDOP 2);
```

Cette option remplace l'option de configuration MAXDOP de sp_configure et du gouverneur de ressources. Si MAXDOP est défini sur zéro, le serveur choisit le degré de parallélisme maximal.

INDEX Conseils

Les indicateurs d'index sont utilisés pour forcer une requête à utiliser un index spécifique, au lieu d'autoriser Query Optimizer de SQL Server à choisir ce qu'il considère comme le meilleur index. Dans certains cas, vous pouvez obtenir des avantages en spécifiant l'index qu'une requête doit utiliser. Généralement, Query Optimizer de SQL Server choisit le meilleur index adapté à la requête, mais en raison de statistiques manquantes / obsolètes ou de besoins spécifiques, vous pouvez le forcer.

```
SELECT *
FROM mytable WITH (INDEX (ix_date))
WHERE field1 > 0
    AND CreationDate > '20170101'
```

Lire Conseils de requête en ligne: https://riptutorial.com/fr/sql-server/topic/6881/conseils-de-requete

Chapitre 16: Conversion de types de données

Examples

ESSAYEZ PARSE

SQL Server 2012

Il convertit le type de données chaîne en type de données cible (Date ou Numérique).

Par exemple, les données source sont du type chaîne et nous devons les convertir en type date. Si la tentative de conversion échoue, elle renvoie la valeur NULL.

Syntaxe: TRY_PARSE (string_value AS data_type [culture USING])

String_value - Cet argument est la valeur source qui est de type NVARCHAR (4000).
Data_type - Cet argument est le type de données cible, date ou numérique.
Culture - C'est un argument facultatif qui permet de convertir la valeur au format Culture. Supposons que vous souhaitiez afficher la date en français, vous devez alors passer le type de culture comme «Fr-FR». Si vous ne passez pas de nom de culture valide, PARSE générera une erreur.

```
DECLARE @fakeDate AS varchar(10);
DECLARE @realDate AS VARCHAR(10);
SET @fakeDate = 'iamnotadate';
SET @realDate = '13/09/2015';

SELECT TRY_PARSE(@fakeDate AS DATE); --NULL  as the parsing fails

SELECT TRY_PARSE(@realDate AS DATE); -- NULL due to type mismatch

SELECT TRY_PARSE(@realDate AS DATE USING 'Fr-FR'); -- 2015-09-13
```

ESSAYEZ CONVERTIR

SQL Server 2012

Il convertit la valeur en type de données spécifié et, en cas d'échec de la conversion, renvoie NULL. Par exemple, la valeur source au format chaîne et nous avons besoin du format date / entier. Ensuite, cela nous aidera à réaliser la même chose.

Syntaxe: TRY_CONVERT (type_données [(longueur)], expression [, style])

TRY_CONVERT () renvoie une valeur convertie au type de données spécifié si la conversion réussit; sinon, renvoie null.

Data_type - Le type de données dans lequel convertir. Ici, length est un paramètre facultatif qui

aide à obtenir un résultat dans la longueur spécifiée.
Expression - La valeur à convertir
Style - C'est un paramètre facultatif qui détermine le formatage. Supposons que vous souhaitiez un format de date tel que «18 mai 2013», alors vous avez besoin du style de passe comme 111.

```
DECLARE @sampletext AS VARCHAR(10);
SET @sampletext = '123456';
DECLARE @ realDate AS VARCHAR(10);
SET @realDate = '13/09/2015’;
SELECT TRY_CONVERT(INT, @sampletext); -- 123456
SELECT TRY_CONVERT(DATETIME, @sampletext); -- NULL
SELECT TRY_CONVERT(DATETIME, @realDate, 111); -- Sep, 13 2015
```

ESSAYEZ CAST

SQL Server 2012

Il convertit la valeur en type de données spécifié et, en cas d'échec de la conversion, renvoie NULL. Par exemple, la valeur source au format chaîne et nous en avons besoin au format double / entier. Cela nous aidera alors à y parvenir.

Syntaxe: TRY_CAST (expression AS data_type [(length)])

TRY_CAST () renvoie une valeur convertie au type de données spécifié si la conversion réussit; sinon, renvoie null.

Expression - La valeur source qui sera envoyée.
Data_type - Le type de données cible que la valeur source va générer.
Length - C'est un paramètre facultatif qui spécifie la longueur du type de données cible.

```
DECLARE @sampletext AS VARCHAR(10);
SET @sampletext = '123456';

SELECT TRY_CAST(@sampletext AS INT); -- 123456
SELECT TRY_CAST(@sampletext AS DATE); -- NULL
```

Jeter

La fonction Cast () permet de convertir une variable de type de données ou des données d'un type de données en un autre type de données.

Syntaxe

CAST ([Expression] AS Datatype)

Le type de données sur lequel vous transposez une expression est le type cible. Le type de données de l'expression à partir de laquelle vous avez transtypé est le type de source.

```
DECLARE @A varchar(2)
DECLARE @B varchar(2)
```

```
set @A='25a'
set @B='15'

Select CAST(@A as int) + CAST(@B as int)  as Result
--'25a' is casted to 25 (string to int)
--'15' is casted to 15 (string to int)

--Result
 --40

DECLARE @C varchar(2)  = 'a'

select CAST(@C as int) as Result
--Result
 --Conversion failed when converting the varchar value 'a' to data type int.
```

Lance une erreur en cas d'échec

Convertir

Lorsque vous convertissez des expressions d'un type à un autre, dans une procédure stockée ou une autre routine, il est souvent nécessaire de convertir les données d'un type datetime en un type varchar. La fonction Convert est utilisée pour de telles choses. La fonction CONVERT () peut être utilisée pour afficher les données de date / heure sous différents formats. Syntaxe

CONVERT (data_type (length), expression, style)

Style - valeurs de style pour la conversion datetime ou smalldatetime en données de caractère. Ajoutez 100 à une valeur de style pour obtenir une année à quatre places incluant le siècle (aaaa).

```
select convert(varchar(20),GETDATE(),108)

13:27:16
```

Lire Conversion de types de données en ligne: https://riptutorial.com/fr/sql-server/topic/5034/conversion-de-types-de-donnees

Chapitre 17: Courtier de service

Examples

1. Bases

Le courtier de services est une technologie basée sur une communication asynchrone entre deux entités (ou plus). Le courtier de services comprend: les types de message, les contrats, les files d'attente, les services, les itinéraires et au moins les points de terminaison d'instance.

Plus: https://msdn.microsoft.com/en-us/library/bb522893.aspx

2. Activer le courtier de services sur la base de données

```
ALTER DATABASE [MyDatabase] SET ENABLE_BROKER WITH ROLLBACK IMMEDIATE;
```

3. Créer une structure de base de courtier de services sur la base de données (communication par base de données unique)

```
USE [MyDatabase]

CREATE MESSAGE TYPE [//initiator] VALIDATION = WELL_FORMED_XML;
    GO

CREATE CONTRACT [//call/contract]
(
    [//initiator] SENT BY INITIATOR
)
GO

CREATE QUEUE  InitiatorQueue;
GO

CREATE QUEUE  TargetQueue;
GO

CREATE SERVICE InitiatorService
      ON QUEUE InitiatorQueue
(
     [//call/contract]

)

CREATE SERVICE TargetService
ON QUEUE TargetQueue
(
     [//call/contract]

)

GRANT SEND ON SERVICE::[InitiatorService] TO PUBLIC
GO
```

```
GRANT SEND ON SERVICE::[TargetService] TO PUBLIC
GO
```

Nous n'avons pas besoin de route pour une communication de base de données.

4. Comment envoyer une communication de base par l'intermédiaire d'un courtier de services

Pour cette démonstration, nous utiliserons la construction du service broker créée dans une autre partie de cette documentation. La partie mentionnée est nommée **3. Créer une construction de base de courtier de services sur la base de données (communication à base de données unique)** .

```
USE [MyDatabase]

DECLARE @ch uniqueidentifier = NEWID()
DECLARE @msg XML

BEGIN DIALOG CONVERSATION @ch
    FROM SERVICE [InitiatorService]
    TO SERVICE 'TargetService'
    ON CONTRACT [//call/contract]
    WITH ENCRYPTION = OFF; -- more possible options

        SET @msg = (
                    SELECT 'HelloThere' "elementNum1"
                    FOR XML PATH(''), ROOT('ExampleRoot'), ELEMENTS XSINIL, TYPE
                    );

SEND ON CONVERSATION @ch MESSAGE TYPE [//initiator] (@msg);
END CONVERSATION @ch;
```

Après cette conversation sera votre msg dans TargetQueue

5. Comment recevoir la conversation de TargetQueue automatiquement

Pour cette démonstration, nous utiliserons la construction du service broker créée dans une autre partie de cette documentation. La partie mentionnée est appelée **3. Créer une structure de base de courtier de services sur la base de données (communication par base de données unique)** .

Nous devons d'abord créer une procédure capable de lire et de traiter les données de la file d'attente.

```
USE [MyDatabase]
GO

SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE PROCEDURE [dbo].[p_RecieveMessageFromTargetQueue]

    AS
    BEGIN

    declare
    @message_body xml,
    @message_type_name nvarchar(256),
    @conversation_handle uniqueidentifier,
    @messagetypename nvarchar(256);



   WHILE 1=1
   BEGIN

    BEGIN TRANSACTION
        WAITFOR(
        RECEIVE TOP(1)
        @message_body = CAST(message_body as xml),
        @message_type_name = message_type_name,
        @conversation_handle = conversation_handle,
        @messagetypename = message_type_name
        FROM DwhInsertSmsQueue
        ), TIMEOUT 1000;

         IF (@@ROWCOUNT = 0)
            BEGIN
                ROLLBACK TRANSACTION
                BREAK
            END

         IF (@messagetypename = '//initiator')
             BEGIN

                IF OBJECT_ID('MyDatabase..MyExampleTableHelloThere') IS NOT NULL
                    DROP TABLE dbo.MyExampleTableHelloThere

                SELECT @message_body.value('(/ExampleRoot/"elementNum1")[1]', 'VARCHAR(50)')
AS MyExampleMessage
                INTO dbo.MyExampleTableHelloThere

             END



         IF (@messagetypename = 'http://schemas.microsoft.com/SQL/ServiceBroker/EndDialog')
            BEGIN
                END CONVERSATION @conversation_handle;
            END

    COMMIT TRANSACTION
   END


END
```

Deuxième étape: autoriser votre TargetQueue à exécuter automatiquement votre procédure:

```
USE [MyDatabase]
```

```
ALTER QUEUE [dbo].[TargetQueue] WITH STATUS = ON , RETENTION = OFF ,
ACTIVATION
 (  STATUS = ON , --activation status
    PROCEDURE_NAME = dbo.p_RecieveMessageFromTargetQueue , --procedure name
    MAX_QUEUE_READERS = 1 , --number of readers
    EXECUTE AS SELF  )
```

Lire Courtier de service en ligne: https://riptutorial.com/fr/sql-server/topic/7651/courtier-de-service

Chapitre 18: CRÉER UNE VUE

Examples

CRÉER UNE VUE

```
CREATE VIEW view_EmployeeInfo
AS
    SELECT EmployeeID,
        FirstName,
        LastName,
        HireDate
    FROM Employee
GO
```

Les lignes de vues peuvent être sélectionnées de la même manière que les tableaux:

```
SELECT FirstName
FROM view_EmployeeInfo
```

Vous pouvez également créer une vue avec une colonne calculée. Nous pouvons modifier la vue ci-dessus comme suit en ajoutant une colonne calculée:

```
CREATE VIEW view_EmployeeReport
AS
    SELECT EmployeeID,
        FirstName,
        LastName,
        Coalesce(FirstName,'') + ' ' + Coalesce(LastName,'') as FullName,
        HireDate
    FROM Employee
GO
```

Ce point de vue ajoute une colonne supplémentaire qui apparaîtra lorsque vous `SELECT` lignes de celui - ci. Les valeurs de cette colonne supplémentaire dépendent des champs `FirstName` et `LastName` dans la table `Employee` et sont automatiquement mises à jour en arrière-plan lorsque ces champs sont mis à jour.

CRÉER UNE VUE avec cryptage

```
CREATE VIEW view_EmployeeInfo
WITH ENCRYPTION
AS
SELECT EmployeeID, FirstName, LastName, HireDate
FROM Employee
GO
```

CREATE VIEW Avec INNER JOIN

```
CREATE VIEW view_PersonEmployee
AS
    SELECT P.LastName,
        P.FirstName,
        E.JobTitle
    FROM Employee AS E
    INNER JOIN Person AS P
        ON P.BusinessEntityID = E.BusinessEntityID
GO
```

Les vues peuvent utiliser des jointures pour sélectionner des données provenant de nombreuses sources telles que des tables, des fonctions de tableau ou même d'autres vues. Cet exemple utilise les colonnes FirstName et LastName de la table Person et la colonne JobTitle de la table Employee.

Cette vue peut maintenant être utilisée pour voir toutes les lignes correspondantes pour les gestionnaires de la base de données:

```
SELECT *
FROM view_PersonEmployee
WHERE JobTitle LIKE '%Manager%'
```

CREER une vue indexée

Pour créer une vue avec un index, la vue doit être créée à l'aide des mots-clés `WITH SCHEMABINDING` :

```
CREATE VIEW view_EmployeeInfo
WITH SCHEMABINDING
AS
    SELECT EmployeeID,
        FirstName,
        LastName,
        HireDate
    FROM [dbo].Employee
GO
```

Tous les index en cluster ou non peuvent être maintenant créés:

```
CREATE UNIQUE CLUSTERED INDEX IX_view_EmployeeInfo
ON view_EmployeeInfo
(
     EmployeeID ASC
)
```

Il y a des limitations aux vues indexées:

- La définition de vue peut référencer une ou plusieurs tables de la même base de données.
- Une fois l'index unique en cluster créé, des index non clusterisés supplémentaires peuvent être créés dans la vue.

- Vous pouvez mettre à jour les données dans les tables sous-jacentes, y compris insérer, mettre à jour, supprimer et même tronquer.
- Vous ne pouvez pas modifier les tables et les colonnes sous-jacentes. La vue est créée avec l'option WITH SCHEMABINDING.
- Il ne peut pas contenir COUNT, MIN, MAX, TOP, les jointures externes ou quelques autres mots-clés ou éléments.

Pour plus d'informations sur la création de vues indexées, vous pouvez lire cet article MSDN

VUES groupées

Une vue groupée est basée sur une requête avec une clause GROUP BY. Comme chacun des groupes peut avoir plus d'une ligne dans la base à partir de laquelle il a été construit, il s'agit nécessairement de vues en lecture seule. Ces vues ont généralement une ou plusieurs fonctions d'agrégat et sont utilisées à des fins de génération de rapports. Ils sont également pratiques pour contourner les faiblesses de SQL. Considérons une vue qui montre la plus grande vente dans chaque état. La requête est simple:

https://www.simple-talk.com/sql/t-sql-programming/sql-view-beyond-the-basics/

```
CREATE VIEW BigSales (state_code, sales_amt_total)
AS SELECT state_code, MAX(sales_amt)
     FROM Sales
    GROUP BY state_code;
```

VUES UNION-ed

Les vues basées sur une opération UNION ou UNION ALL sont en lecture seule car il n'existe aucun moyen unique de mapper une modification sur une seule ligne dans l'une des tables de base. L'opérateur UNION supprimera les lignes en double des résultats. Les deux opérateurs UNION et UNION ALL masquent la table dont proviennent les lignes. Ces vues doivent utiliser un, car les colonnes d'un UNION [ALL] n'ont pas de nom propre. En théorie, une UNION de deux tables disjointes, dont aucune ne contient de lignes en double, devrait pouvoir être mise à jour.

https://www.simple-talk.com/sql/t-sql-programming/sql-view-beyond-the-basics/

```
CREATE VIEW DepTally2 (emp_nbr, dependent_cnt)
AS (SELECT emp_nbr, COUNT(*)
      FROM Dependents
     GROUP BY emp_nbr)
   UNION
   (SELECT emp_nbr, 0
      FROM Personnel AS P2
     WHERE NOT EXISTS
          (SELECT *
             FROM Dependents AS D2
            WHERE D2.emp_nbr = P2.emp_nbr));
```

Lire CRÉER UNE VUE en ligne: https://riptutorial.com/fr/sql-server/topic/3815/creer-une-vue

Chapitre 19: croix appliquer

Examples

Joindre des lignes de table avec des lignes générées dynamiquement à partir d'une cellule

CROSS APPLY vous permet de "joindre" des lignes à partir d'une table avec des lignes générées dynamiquement renvoyées par une fonction de valeur de table.

Imaginez que vous ayez une table Company avec une colonne contenant un tableau de produits (colonne ProductList) et une fonction qui analyse ces valeurs et renvoie un ensemble de produits. Vous pouvez sélectionner toutes les lignes d'une table Company, appliquer cette fonction sur une colonne ProductList et "joindre" les résultats générés à la ligne mère de la société:

```
SELECT *
FROM Companies c
     CROSS APPLY dbo.GetProductList( c.ProductList ) p
```

Pour chaque ligne, la valeur de la cellule *ProductList* sera fournie à la fonction et la fonction retournera ces produits sous la forme d'un ensemble de lignes pouvant être jointes à la ligne parente.

Joindre des lignes de table avec un tableau JSON stocké dans la cellule

CROSS APPLY vous permet de "joindre" des lignes à partir d'une table avec une collection d'objets JSON stockés dans une colonne.

Imaginez que vous ayez une table Company avec une colonne contenant un tableau de produits (colonne ProductList) formaté en tant que tableau JSON. La fonction de valeur de la table OPENJSON peut analyser ces valeurs et renvoyer l'ensemble de produits. Vous pouvez sélectionner toutes les lignes d'une table Société, analyser les produits JSON avec OPENJSON et "joindre" les résultats générés avec la ligne mère de la société:

```
SELECT *
FROM Companies c
     CROSS APPLY OPENJSON( c.ProductList )
                 WITH ( Id int, Title nvarchar(30), Price money)
```

Pour chaque ligne, la valeur de la cellule *ProductList* sera fournie à la fonction OPENJSON qui transformera les objets JSON en lignes avec le schéma défini dans la clause WITH.

Filtrer les lignes par valeurs de tableau

Si vous stockez une liste de balises dans une rangée en tant que valeurs séparées par un coma, la fonction *STRING_SPLIT* vous permet de transformer une liste de balises en une table de

valeurs. **CROSS APPLY** vous permet de "joindre" les valeurs analysées par la fonction *STRING_SPLIT* avec une ligne parente.

Imaginez que vous ayez une table Product avec une colonne contenant un tableau de balises séparées par des virgules (par exemple, promo, sales, new). STRING_SPLIT et CROSS APPLY vous permettent de joindre des lignes de produits avec leurs balises pour filtrer les produits par balises:

```
SELECT *
FROM Products p
     CROSS APPLY STRING_SPLIT( p.Tags, ',' ) tags
WHERE tags.value = 'promo'
```

Pour chaque ligne, la valeur de la cellule *Tags* sera fournie à la fonction STRING_SPLIT qui renverra les valeurs des balises. Vous pouvez ensuite filtrer les lignes en fonction de ces valeurs.

Remarque: la fonction *STRING_SPLIT* n'est pas disponible avant **SQL Server 2016**

Lire croix appliquer en ligne: https://riptutorial.com/fr/sql-server/topic/5462/croix-appliquer

Chapitre 20: Cryptage

Paramètres

Paramètres facultatifs	Détails
`WITH PRIVATE KEY`	Pour CREATE CERTIFICATE, une clé privée peut être spécifiée: `(FILE='D:\Temp\CertTest\private.pvk', DECRYPTION BY PASSWORD = 'password');`

Remarques

La création d'un certificat DER fonctionnera correctement. Cependant, lorsqu'un certificat Base64 est utilisé, le serveur SQL se plaint du message crypté:

```
Msg 15468, Level 16, State 6, Line 1
An error occurred during the generation of the certificate.
```

Importez votre certificat Base64 dans le magasin de certificats de votre système d'exploitation pour pouvoir le réexporter au format binaire DER.

Une autre chose importante à faire est d'avoir une hiérarchie de chiffrement afin de protéger l'autre, jusqu'au niveau du système d'exploitation. Voir l'article sur 'Encryption of database / TDE'

Pour plus d'informations sur la création de certificats, rendez-vous sur:
https://msdn.microsoft.com/en-us/library/ms187798.aspx

Pour plus d'informations sur le chiffrement de la base de données / TDE, rendez-vous sur:
https://msdn.microsoft.com/en-us/library/bb934049.aspx

Pour plus d'informations sur le chiffrement des données, rendez-vous sur:
https://msdn.microsoft.com/en-us/library/ms188061.aspx

Examples

Chiffrement par certificat

```
CREATE CERTIFICATE My_New_Cert
FROM FILE = 'D:\Temp\CertTest\certificateDER.cer'
GO
```

Créer le certificat

```
SELECT EncryptByCert(Cert_ID('My_New_Cert'),
'This text will get encrypted') encryption_test
```

En général, vous cryptez avec une clé symétrique, cette clé serait chiffrée par la clé asymétrique (clé publique) de votre certificat.

Notez également que le chiffrement est limité à certaines longueurs en fonction de la longueur de la clé et renvoie NULL dans le cas contraire. Microsoft écrit: "Les limites sont les suivantes: une clé RSA 512 bits peut chiffrer jusqu'à 53 octets, une clé 1024 bits peut chiffrer jusqu'à 117 octets, et une clé 2048 bits peut chiffrer jusqu'à 245 octets."

EncryptByAsymKey a les mêmes limites. Pour UNICODE, cela serait divisé par 2 (16 bits par caractère), soit 58 caractères pour une clé de 1024 bits.

Cryptage de la base de données

```
USE TDE
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE My_New_Cert
GO

ALTER DATABASE TDE
SET ENCRYPTION ON
GO
```

Cela utilise le «chiffrement de données transparent» (TDE)

Chiffrement par clé symétrique

```
-- Create the key and protect it with the cert
CREATE SYMMETRIC KEY My_Sym_Key
WITH ALGORITHM = AES_256
ENCRYPTION BY CERTIFICATE My_New_Cert;
GO

-- open the key
OPEN SYMMETRIC KEY My_Sym_Key
DECRYPTION BY CERTIFICATE My_New_Cert;

-- Encrypt
SELECT EncryptByKey(Key_GUID('SSN_Key_01'), 'This text will get encrypted');
```

Chiffrement par mot de passe

```
SELECT EncryptByPassphrase('MyPassPhrase', 'This text will get encrypted')
```

Cela va également chiffrer, mais ensuite par phrase secrète au lieu de la clé asymétrique (certificat) ou par une clé symétrique explicite.

Lire Cryptage en ligne: https://riptutorial.com/fr/sql-server/topic/7096/cryptage

Chapitre 21: DBCC

Examples

Commandes de maintenance DBCC

Les commandes DBCC permettent à l'utilisateur de conserver de l'espace dans la base de données, de nettoyer les caches, de réduire les bases de données et les tables.

Les exemples sont:

```
DBCC DROPCLEANBUFFERS
```

Supprime tous les tampons propres du pool de mémoire tampon et les objets columnstore du pool d'objets columnstore.

```
DBCC FREEPROCCACHE
-- or
DBCC FREEPROCCACHE (0x060006001ECA270EC0215D05000000000000000000000000);
```

Supprime toutes les requêtes SQL dans le cache du plan. Chaque nouveau plan sera recompilé: vous pouvez spécifier un descripteur de plan, un descripteur de requête pour nettoyer les plans du plan de requête ou de l'instruction SQL spécifique.

```
DBCC FREESYSTEMCACHE ('ALL', myresourcepool);
-- or
DBCC FREESYSTEMCACHE;
```

Nettoie toutes les entrées mises en cache créées par le système. Il peut nettoyer les entrées o = dans tout ou partie du pool de ressources spécifié (**myresourcepool** dans l'exemple ci-dessus)

```
DBCC FLUSHAUTHCACHE
```

Vide le cache d'authentification de la base de données contenant des informations sur les connexions et les règles de pare-feu.

```
DBCC SHRINKDATABASE (MyDB [, 10]);
```

Réduit la base de données MyDB à 10%. Le deuxième paramètre est facultatif. Vous pouvez utiliser l'ID de base de données au lieu du nom.

```
DBCC SHRINKFILE (DataFile1, 7);
```

Réduit le fichier de données nommé DataFile1 dans la base de données en cours. La taille de la cible est de 7 Mo (ce paramètre est facultatif).

```
DBCC CLEANTABLE (AdventureWorks2012,'Production.Document', 0)
```

Récupère un espace de la table spécifiée

Déclarations de validation DBCC

Les commandes DBCC permettent à l'utilisateur de valider l'état de la base de données.

```
ALTER TABLE Table1 WITH NOCHECK ADD CONSTRAINT chkTab1 CHECK (Col1 > 100);
GO
DBCC CHECKCONSTRAINTS(Table1);
--OR
DBCC CHECKCONSTRAINTS ('Table1.chkTable1');
```

La contrainte de vérification est ajoutée avec les options de vérification, elle ne sera donc pas vérifiée sur les données existantes. DBCC déclenchera une vérification des contraintes.

Les commandes suivantes de DBCC vérifient l'intégrité de la base de données, de la table ou du catalogue:

```
DBCC CHECKTABLE tablename1 | tableid
DBCC CHECKDB databasename1 | dbid
DBCC CHECKFILEGROUP filegroup_name | filegroup_id | 0
DBCC CHECKCATALOG databasename1 | database_id1 | 0
```

Déclarations d'informations DBCC

Les commandes DBCC peuvent afficher des informations sur les objets de base de données.

```
DBCC PROCCACHE
```

Affiche les informations dans un format de table à propos du cache de procédure.

```
DBCC OUTPUTBUFFER ( session_id [ , request_id ])
```

Renvoie le tampon de sortie en cours au format hexadécimal et ASCII pour l'identifiant session_id spécifié (et l'ID de demande facultatif).

```
DBCC INPUTBUFFER ( session_id [ , request_id ])
```

Affiche la dernière instruction envoyée par un client à une instance de Microsoft SQL Server.

```
DBCC SHOW_STATISTICS ( table_or_indexed_view_name , column_statistic_or_index_name)
```

Commandes de trace DBCC

Les indicateurs de trace dans SQL Server sont utilisés pour modifier le comportement de SQL Server, activer / désactiver certaines fonctionnalités. Les commandes DBCC peuvent contrôler les

indicateurs de trace:

L'exemple suivant active l'indicateur de trace 3205 globalement et 3206 pour la session en cours:

```
DBCC TRACEON (3205, -1);
DBCC TRACEON (3206);
```

L'exemple suivant désactive l'indicateur de trace 3205 globalement et 3206 pour la session en cours:

```
DBCC TRACEON (3205, -1);
DBCC TRACEON (3206);
```

L'exemple suivant affiche l'état des indicateurs de trace 2528 et 3205:

```
DBCC TRACESTATUS (2528, 3205);
```

Déclaration DBCC

Les instructions DBCC agissent en tant que commandes de la console de base de données pour SQL Server. Pour obtenir les informations de syntaxe pour la commande DBCC spécifiée, utilisez l'instruction DBCC HELP (...).

L'exemple suivant retourne toutes les instructions DBCC pour lesquelles l'aide est disponible:

```
DBCC HELP ('?');
```

L'exemple suivant renvoie des options pour l'instruction DBCC CHECKDB:

```
DBCC HELP ('CHECKDB');
```

Lire DBCC en ligne: https://riptutorial.com/fr/sql-server/topic/7316/dbcc

Chapitre 22: DBMAIL

Syntaxe

- sp_send_dbmail [[@profile_name =] 'nom_profil'] [, [@recipients =] 'destinataires [; ... n] '] [, [@copy_recipients =]' copy_recipient [; ... n] '] [, [@blind_copy_recipients =]' blind_copy_recipient [; ... n] '] [, [@from_address =]' from_address '] [, [@reply_to =]' reply_to '] [, [@subject =]' subject '] [, [@body =]' body '] [, [@body_format =] 'body_format'] [, [@importance =] 'importance'] [, [@sensitivity =] 'sensibilité'] [, [@file_attachments =] 'pièce jointe [; ... n] '] [, [@query =]' query '] [, [@execute_query_database =]' execute_query_database '] [, [@attach_query_result_as_file =] attach_query_result_as_file] [, [@query_attachment_filename =] query_attachment_filename] [, @query_result_header =] query_result_header] [, [@query_result_width =] query_result_width] [, [@query_result_separator =] 'query_result_separator'] [, [@exclude_query_output =] exclude_query_output] [, [@append_query_error =] append_query_error] [, [@query_no_truncate =] query_no_truncate] [, [@query_result_no_padding =] @query_result_no_padding] [, [@mailitem_id =] mailitem_id] [OUTPUT]

Examples

Envoyer un simple email

Ce code envoie un simple email textuel à recipient@someaddress.com

```
EXEC msdb.dbo.sp_send_dbmail
    @profile_name = 'The Profile Name',
    @recipients = 'recipient@someaddress.com',
    @body = 'This is a simple email sent from SQL Server.',
    @subject = 'Simple email'
```

Envoyer les résultats d'une requête

Cela joint les résultats de la requête `SELECT * FROM Users` et les envoie à `recipient@someaddress.com`

```
EXEC msdb.dbo.sp_send_dbmail
    @profile_name = 'The Profile Name',
    @recipients = 'recipient@someaddress.com',
    @query = 'SELECT * FROM Users',
    @subject = 'List of users',
    @attach_query_result_as_file = 1;
```

Envoyer un email HTML

Le contenu HTML doit être transmis à `sp_send_dbmail`

SQL Server 2012

```
DECLARE @html VARCHAR(MAX);
SET @html = CONCAT
(
    '<html><body>',
    '<h1>Some Header Text</h1>',
    '<p>Some paragraph text</p>',
    '</body></html>'
)
```

SQL Server 2012

```
DECLARE @html VARCHAR(MAX);
SET @html =
    '<html><body>' +
    '<h1>Some Header Text</h1>' +
    '<p>Some paragraph text</p>' +
    '</body></html>';
```

Ensuite, utilisez la variable `@html` avec l' `@body argument` . La chaîne HTML peut également être transmise directement à `@body` , bien que cela puisse rendre le code plus difficile à lire.

```
EXEC msdb.dbo.sp_send_dbmail
    @recipients='recipient@someaddress.com',
    @subject = 'Some HTML content',
    @body = @html,
    @body_format = 'HTML';
```

Lire DBMAIL en ligne: https://riptutorial.com/fr/sql-server/topic/4908/dbmail

Chapitre 23: Déclaration CASE

Remarques

L'exemple ci-dessus est juste pour montrer la syntaxe pour utiliser des instructions de cas dans SQL Server avec l'exemple du jour de la semaine. Même si la même sortie peut être obtenue en utilisant "SELECT DATENAME (WEEKDAY, GETDATE ())" également.

Examples

Déclaration CASE simple

Dans une déclaration de cas simple, une valeur ou une variable est vérifiée par rapport à plusieurs réponses possibles. Le code ci-dessous est un exemple d'une déclaration de cas simple:

```
SELECT CASE DATEPART(WEEKDAY, GETDATE())
    WHEN 1 THEN 'Sunday'
    WHEN 2 THEN 'Monday'
    WHEN 3 THEN 'Tuesday'
    WHEN 4 THEN 'Wednesday'
    WHEN 5 THEN 'Thursday'
    WHEN 6 THEN 'Friday'
    WHEN 7 THEN 'Saturday'
END
```

Déclaration CASE recherchée

Dans une instruction Case recherchée, chaque option peut tester une ou plusieurs valeurs indépendamment. Le code ci-dessous est un exemple d'une déclaration de cas recherchée:

```
DECLARE @FirstName varchar(30) = 'John'
DECLARE @LastName varchar(30) = 'Smith'

SELECT CASE
    WHEN LEFT(@FirstName, 1) IN ('a','e','i','o','u')
        THEN 'First name starts with a vowel'
    WHEN LEFT(@LastName, 1) IN ('a','e','i','o','u')
        THEN 'Last name starts with a vowel'
    ELSE
        'Neither name starts with a vowel'
END
```

Lire Déclaration CASE en ligne: https://riptutorial.com/fr/sql-server/topic/7238/declaration-case

Chapitre 24: Déclencheur

Introduction

Un déclencheur est un type spécial de procédure stockée, qui est exécuté automatiquement après qu'un événement se soit produit. Il existe deux types de déclencheurs: les déclencheurs de langage de définition de données et les déclencheurs de langage de manipulation de données.

Il est généralement lié à une table et se déclenche automatiquement. Vous ne pouvez pas appeler explicitement un déclencheur.

Examples

Types et classifications du déclencheur

Dans SQL Server, il existe deux catégories de déclencheurs: les déclencheurs DDL et les déclencheurs DML.

Les déclencheurs DDL sont déclenchés en réponse aux événements DDL (Data Definition Language). Ces événements correspondent principalement aux instructions Transact-SQL qui commencent par les mots-clés `CREATE` , `ALTER` et `DROP` .

Les déclencheurs DML sont déclenchés en réponse aux événements DML (Data Manipulation Language). Ces événements correspondent aux instructions Transact-SQL qui commencent par les mots-clés `INSERT` , `UPDATE` et `DELETE` .

Les déclencheurs DML sont classés en deux types principaux:

1. Après les déclencheurs (pour les déclencheurs)

    - APRÈS INSÉRER Déclencheur.
    - AFTER UPDATE Trigger.
    - APRÈS SUPPRIMER Déclencheur.

2. Au lieu de déclencheurs

    - INSTEAD OF INSERT Trigger.
    - AU LIEU DE MISE À JOUR Déclencheur.
    - AU LIEU DE SUPPRIMER Déclencheur.

Déclencheurs DML

Les déclencheurs DML sont déclenchés en réponse aux instructions dml (`insert` , `update` ou `delete`).
Un déclencheur dml peut être créé pour traiter un ou plusieurs événements dml pour une seule table ou vue. Cela signifie qu'un seul déclencheur dml peut gérer l'insertion, la mise à jour et la

suppression d'enregistrements dans une table ou une vue spécifique, mais ne peut gérer que les données modifiées sur cette seule table ou vue.

Les déclencheurs DML donnent accès aux tables `inserted` et `deleted` qui contiennent des informations sur les données qui ont été / seront affectées par l'instruction d'insertion, de mise à jour ou de suppression qui a déclenché le déclencheur.

Notez que les déclencheurs DML sont basés sur des instructions et non sur des lignes. Cela signifie que si l'instruction a effectué plus d'une ligne, les tables insérées ou supprimées contiendront plus d'une ligne.

Exemples:

```
CREATE TRIGGER tblSomething_InsertOrUpdate ON tblSomething
FOR INSERT
AS

    INSERT INTO tblAudit (TableName, RecordId, Action)
    SELECT 'tblSomething', Id, 'Inserted'
    FROM Inserted

GO

CREATE TRIGGER tblSomething_InsertOrUpdate ON tblSomething
FOR UPDATE
AS

    INSERT INTO tblAudit (TableName, RecordId, Action)
    SELECT 'tblSomething', Id, 'Updated'
    FROM Inserted

GO

CREATE TRIGGER tblSomething_InsertOrUpdate ON tblSomething
FOR DELETE
AS

    INSERT INTO tblAudit (TableName, RecordId, Action)
    SELECT 'tblSomething', Id, 'Deleted'
    FROM Deleted

GO
```

Tous les exemples ci-dessus ajoutent des enregistrements à tblAudit chaque fois qu'un enregistrement est ajouté, supprimé ou mis à jour dans tblSomething.

Lire Déclencheur en ligne: https://riptutorial.com/fr/sql-server/topic/5032/declencheur

Chapitre 25: Délimiter des caractères spéciaux et des mots réservés

Remarques

En règle générale, il est préférable de ne pas utiliser les mots réservés T-SQL comme noms de tables, noms de colonnes, noms d'objets de programmation, alias, etc. Ainsi, la méthode pour échapper à ces mots-clés ne doit être appliquée .

Pour les mots réservés, l'utilisation des crochets n'est pas obligatoire. Lors de l'utilisation d'un outil tel que SQL Server Management Studio, les mots réservés seront mis en évidence pour attirer l'attention sur le fait qu'ils sont réservés.

Examples

Méthode de base

La méthode de base pour échapper à des mots réservés pour SQL Server est l'utilisation des crochets (`[` et `]`). Par exemple, *Description* et *Nom* sont des mots réservés; cependant, s'il existe un objet utilisant les deux comme noms, la syntaxe utilisée est la suivante:

```
SELECT [Description]
FROM   dbo.TableName
WHERE  [Name] = 'foo'
```

Le seul caractère spécial pour SQL Server est la seule citation `'` et il est échappé en doublant son utilisation. Par exemple, pour trouver le nom *O'Shea* dans la même table, la syntaxe suivante serait utilisée:

```
SELECT [Description]
FROM   dbo.TableName
WHERE  [Name] = 'O''Shea'
```

Lire Délimiter des caractères spéciaux et des mots réservés en ligne: https://riptutorial.com/fr/sql-server/topic/7156/delimiter-des-caracteres-speciaux-et-des-mots-reserves

Chapitre 26: Déplacer et copier des données autour des tables

Examples

Copier des données d'une table à une autre

Ce code sélectionne les données d'une table et les affiche dans l'outil de requête (généralement SSMS)

```
SELECT Column1, Column2, Column3 FROM MySourceTable;
```

Ce code insère ces données dans une table:

```
INSERT INTO MyTargetTable (Column1, Column2, Column3)
SELECT Column1, Column2, Column3 FROM MySourceTable;
```

Copier des données dans une table, en créant cette table à la volée

Ce code sélectionne des données dans une table:

```
SELECT Column1, Column2, Column3 FROM MySourceTable;
```

Ce code crée une nouvelle table appelée `MyNewTable` et y place ces données

```
SELECT Column1, Column2, Column3
INTO MyNewTable
FROM MySourceTable;
```

Déplacer des données dans une table (en supposant la méthode des clés uniques)

Pour *déplacer les* données, insérez-les d'abord dans la cible, puis supprimez tout ce que vous avez inséré dans la table source. Ce n'est pas une opération SQL normale, mais cela peut être intéressant

Qu'avez-vous inséré? Normalement, dans les bases de données, vous devez avoir une ou plusieurs colonnes que vous pouvez utiliser pour identifier de manière unique les lignes afin que nous puissions en tenir compte et les utiliser.

Cette instruction sélectionne des lignes

```
SELECT Key1, Key2, Column3, Column4 FROM MyTable;
```

Tout d'abord, nous les insérons dans notre table cible:

```
INSERT INTO TargetTable (Key1, Key2, Column3, Column4)
SELECT Key1, Key2, Column3, Column4 FROM MyTable;
```

Maintenant, en *supposant que les enregistrements des deux tables soient uniques* sur `Key1` , `Key2` , nous pouvons l'utiliser pour rechercher et supprimer des données de la table source

```
DELETE MyTable
WHERE EXISTS (
    SELECT * FROM TargetTable
    WHERE TargetTable.Key1 = SourceTable.Key1
    AND TargetTable.Key2 = SourceTable.Key2
);
```

Cela ne fonctionnera correctement que si `Key1` , `Key2` sont uniques dans les deux tables

Enfin, nous ne voulons pas que le travail soit à moitié terminé. Si nous l'enveloppons dans une transaction, alors toutes les données seront déplacées ou rien ne se produira. Cela garantit que nous n'insérons pas les données et que nous ne pouvons pas supprimer les données de la source.

```
BEGIN TRAN;

INSERT INTO TargetTable (Key1, Key2, Column3, Column4)
SELECT Key1, Key2, Column3, Column4 FROM MyTable;


DELETE MyTable
WHERE EXISTS (
    SELECT * FROM TargetTable
    WHERE TargetTable.Key1 = SourceTable.Key1
    AND TargetTable.Key2 = SourceTable.Key2
);

COMMIT TRAN;
```

Lire Déplacer et copier des données autour des tables en ligne: https://riptutorial.com/fr/sql-server/topic/1467/deplacer-et-copier-des-donnees-autour-des-tables

Chapitre 27: Dernière identité insérée

Examples

SCOPE_IDENTITY ()

```
CREATE TABLE dbo.logging_table(log_id INT IDENTITY(1,1) PRIMARY KEY,
                               log_message VARCHAR(255))

CREATE TABLE dbo.person(person_id INT IDENTITY(1,1) PRIMARY KEY,
                        person_name VARCHAR(100) NOT NULL)
GO;

CREATE TRIGGER dbo.InsertToADifferentTable ON dbo.person
AFTER INSERT
AS
    INSERT INTO dbo.logging_table(log_message)
    VALUES('Someone added something to the person table')
GO;

INSERT INTO dbo.person(person_name)
VALUES('John Doe')

SELECT SCOPE_IDENTITY();
```

Cela retournera la valeur d'identité ajoutée la plus récente produite sur la même connexion, dans l'étendue actuelle. Dans ce cas, 1, pour la première ligne de la table dbo.person.

@@IDENTITÉ

```
CREATE TABLE dbo.logging_table(log_id INT IDENTITY(1,1) PRIMARY KEY,
                               log_message VARCHAR(255))

CREATE TABLE dbo.person(person_id INT IDENTITY(1,1) PRIMARY KEY,
                        person_name VARCHAR(100) NOT NULL)
GO;

CREATE TRIGGER dbo.InsertToADifferentTable ON dbo.person
AFTER INSERT
AS
    INSERT INTO dbo.logging_table(log_message)
    VALUES('Someone added something to the person table')
GO;

INSERT INTO dbo.person(person_name)
VALUES('John Doe')

SELECT @@IDENTITY;
```

Cela retournera l'identité la plus récemment ajoutée sur la même connexion, indépendamment de l'étendue. Dans ce cas, quelle que soit la valeur actuelle de la colonne d'identité de logging_table, en supposant qu'aucune autre activité ne se produit sur l'instance de SQL Server et qu'aucun

autre déclencheur ne soit déclenché à partir de cette insertion.

IDENT_CURRENT ('nom_table')

```
SELECT IDENT_CURRENT('dbo.person');
```

Cela sélectionnera la valeur d'identité la plus récemment ajoutée sur la table sélectionnée, indépendamment de la connexion ou de la portée.

@@ IDENTITY et MAX (ID)

```
SELECT MAX(Id) FROM Employees -- Display the value of Id in the last row in Employees table.
GO
INSERT INTO Employees (FName, LName, PhoneNumber) -- Insert a new row
VALUES ('John', 'Smith', '25558696525')
GO
SELECT @@IDENTITY
GO
SELECT MAX(Id) FROM Employees -- Display the value of Id of the newly inserted row.
GO
```

Les deux dernières valeurs des instructions SELECT sont les mêmes.

Lire Dernière identité insérée en ligne: https://riptutorial.com/fr/sql-server/topic/5674/derniere-identite-inseree

Chapitre 28: Des curseurs

Syntaxe

- DECLARE nom_curseur CURSOR [LOCAL | GLOBAL]
  - [FORWARD_ONLY | FAIRE DÉFILER]
    [STATIQUE | KEYSET | DYNAMIQUE | AVANCE RAPIDE]
    [READ_ONLY | SCROLL_LOCKS | OPTIMISTIQUE]
    [TYPE_WARNING]
  - FOR select_statement
  - [FOR UPDATE [OF nom_colonne [, ... n]]]

Remarques

Normalement, vous voudriez éviter d'utiliser des curseurs car ils peuvent avoir un impact négatif sur les performances. Cependant, dans certains cas particuliers, vous devrez peut-être parcourir votre enregistrement de données par enregistrement et effectuer certaines actions.

Examples

Curseur de base seulement

Normalement, vous voudriez éviter d'utiliser des curseurs car ils peuvent avoir un impact négatif sur les performances. Cependant, dans certains cas particuliers, vous devrez peut-être parcourir votre enregistrement de données par enregistrement et effectuer certaines actions.

```
DECLARE @orderId AS INT

-- here we are creating our cursor, as a local cursor and only allowing
-- forward operations
DECLARE rowCursor CURSOR LOCAL FAST_FORWARD FOR
    -- this is the query that we want to loop through record by record
    SELECT [OrderId]
    FROM [dbo].[Orders]

-- first we need to open the cursor
OPEN rowCursor

-- now we will initialize the cursor by pulling the first row of data, in this example the
[OrderId] column,
-- and storing the value into a variable called @orderId
FETCH NEXT FROM rowCursor INTO @orderId

-- start our loop and keep going until we have no more records to loop through
WHILE @@FETCH_STATUS = 0
BEGIN

    PRINT @orderId
```

```
    -- this is important, as it tells SQL Server to get the next record and store the
[OrderId] column value into the @orderId variable
    FETCH NEXT FROM rowCursor INTO @orderId

END

-- this will release any memory used by the cursor
CLOSE rowCursor
DEALLOCATE rowCursor
```

Syntaxe du curseur rudimentaire

Une syntaxe de curseur simple, fonctionnant sur quelques exemples de lignes de test:

```
/* Prepare test data */
DECLARE @test_table TABLE
(
    Id INT,
    Val VARCHAR(100)
);
INSERT INTO @test_table(Id, Val)
VALUES
    (1, 'Foo'),
    (2, 'Bar'),
    (3, 'Baz');
/* Test data prepared */

/* Iterator variable @myId, for example sake */
DECLARE @myId INT;

/* Cursor to iterate rows and assign values to variables */
DECLARE myCursor CURSOR FOR
    SELECT Id
    FROM @test_table;

/* Start iterating rows */
OPEN myCursor;
FETCH NEXT FROM myCursor INTO @myId;

/* @@FETCH_STATUS global variable will be 1 / true until there are no more rows to fetch */
WHILE @@FETCH_STATUS = 0
BEGIN

    /* Write operations to perform in a loop here. Simple SELECT used for example */
    SELECT Id, Val
    FROM @test_table
    WHERE Id = @myId;

    /* Set variable(s) to the next value returned from iterator; this is needed otherwise the
cursor will loop infinitely. */
    FETCH NEXT FROM myCursor INTO @myId;
END
/* After all is done, clean up */
CLOSE myCursor;
DEALLOCATE myCursor;
```

Résultats du SSMS. Notez que ce sont toutes des requêtes séparées, elles ne sont en aucun cas unifiées. Notez que le moteur de requête traite chaque itération une par une au lieu d'un

ensemble.

Id	Val
1	Foo
(1 rang (s) affecté (s))	

Id	Val
2	Bar
(1 rang (s) affecté (s))	

Id	Val
3	Baz
(1 rang (s) affecté (s))	

Lire Des curseurs en ligne: https://riptutorial.com/fr/sql-server/topic/870/des-curseurs

Chapitre 29: Des vues

Remarques

Les vues sont des requêtes stockées qui peuvent être interrogées comme des tables régulières. Les vues ne font pas partie du modèle physique de la base de données. Toute modification appliquée à la source de données d'une vue, telle qu'une table, sera également reflétée dans la vue.

Examples

Créer une vue

```
CREATE VIEW dbo.PersonsView
AS
SELECT
    name,
    address
FROM persons;
```

Créer ou remplacer une vue

Cette requête supprime la vue - si elle existe déjà - et en crée une nouvelle.

```
IF OBJECT_ID('dbo.PersonsView', 'V') IS NOT NULL
    DROP VIEW dbo.PersonsView
GO

CREATE VIEW dbo.PersonsView
AS
SELECT
    name,
    address
FROM persons;
```

Créer une vue avec liaison de schéma

Si une vue est créée AVEC SCHEMABINDING, la ou les tables sous-jacentes ne peuvent pas être supprimées ou modifiées de telle manière qu'elles casseraient la vue. Par exemple, une colonne de table référencée dans une vue ne peut pas être supprimée.

```
CREATE VIEW dbo.PersonsView
WITH SCHEMABINDING
AS
SELECT
    name,
    address
FROM dbo.PERSONS  -- database schema must be specified when WITH SCHEMABINDING is present
```

Les vues *sans* liaison de schéma peuvent se casser si leur ou leurs tables sous-jacentes changent ou sont supprimées. L'interrogation d'une vue interrompue génère un message d'erreur. sp_refreshview peut être utilisé pour garantir que les vues existantes sans liaison de schéma ne sont pas endommagées.

Lire Des vues en ligne: https://riptutorial.com/fr/sql-server/topic/5327/des-vues

Chapitre 30: Données spatiales

Introduction

Il y a 2 types de données spatiales

Système de coordonnées **géométriques** X / Y pour une surface plane

Géographie Système de coordonnées de latitude / longitude pour une surface courbe (la terre). Il existe plusieurs projections de surfaces courbes, de sorte que chaque espace géographique doit permettre à SQL Server de savoir quelle projection utiliser. L'ID de référence spatiale habituel (SRID) est le 4326, qui mesure les distances en kilomètres. Ceci est le SRID par défaut utilisé dans la plupart des cartes Web

Examples

POINT

Crée un seul point. Ce sera un point de géométrie ou de géographie en fonction de la classe utilisée.

Paramètre	Détail
Lat ou X	Est une expression flottante représentant la coordonnée x du point généré
Long ou y	Est une expression flottante représentant la coordonnée y du point généré
Chaîne	Texte bien connu (WKB) d'une forme géométrique / géographique
Binaire	Binaire bien connu (WKB) d'une forme géométrique / géographique
SRID	Est une expression int représentant l'ID de référence spatiale (SRID) de l'instance de géométrie / géographie que vous souhaitez renvoyer

```
--Explicit constructor
DECLARE @gm1 GEOMETRY = GEOMETRY::Point(10,5,0)

DECLARE @gg1 GEOGRAPHY = GEOGRAPHY::Point(51.511601,-0.096600,4326)

--Implicit constructor (using WKT - Well Known Text)
DECLARE @gm1 GEOMETRY = GEOMETRY::STGeomFromText('POINT(5 10)', 0)

DECLARE @gg1 GEOGRAPHY= GEOGRAPHY::STGeomFromText('POINT(-0.096600 51.511601)', 4326)

--Implicit constructor (using WKB - Well Known Binary)
DECLARE @gm1 GEOMETRY = GEOMETRY::STGeomFromWKB(0x0101000000000000000000144000000000000002440,
0)

DECLARE @gg1 GEOGRAPHY= GEOGRAPHY::STGeomFromWKB(0x01010000005F29CB10C7BAB8BFEACC3D247CC14940,
```

```
4326)
```

Lire Données spatiales en ligne: https://riptutorial.com/fr/sql-server/topic/6816/donnees-spatiales

Chapitre 31: Drop mot-clé

Introduction

Le mot-clé Drop peut être utilisé avec divers objets SQL. Cette rubrique fournit des exemples rapides d'utilisation différente avec des objets de base de données.

Remarques

Liens vers MSDN.

- DROP TABLE (Transact-SQL)
- DROP PROCEDURE (Transact-SQL)
- DROP DATABASE (Transact-SQL)

Examples

Tables de chute

La commande **DROP TABLE** supprime les définitions de table et toutes les données, index, déclencheurs, contraintes et autorisations associées.

Avant de déposer une table, vous devez vérifier si des objets (vues, procédures stockées, autres tables) font référence à la table.

Vous ne pouvez pas supprimer une table référencée par une autre table par FOREIGN KEY. Vous devez d'abord supprimer la clé étrangère en la référençant.

Vous pouvez supprimer une table référencée par une vue ou une procédure stockée, mais après la suppression de la table, la vue ou la procédure stockée n'est plus utilisable.

La syntaxe

```
DROP TABLE [ IF EXISTS ] [ database_name . [ schema_name ] . | schema_name . ]
table_name [ ,...n ] [ ; ]
```

- `IF EXISTS` - Supprimez la table uniquement si elle existe
- `database_name` - Indiquez le nom de la base de données contenant la table
- `schema_name` - Indiquez le nom du schéma sous lequel se trouve la table
- `table_name` - Indiquez le nom de la table à `table_name`

Exemples

Supprimez la table avec le nom **TABLE_1** de la base de données actuelle et le schéma par défaut

dbo

```
DROP TABLE Table_1;
```

Supprimer la table avec **TABLE_1** de la base de données **HR** et le schéma par défaut dbo

```
DROP TABLE HR.Table_1;
```

Supprimez la table avec **TABLE_1** de la base de données **HR** et schéma **externe**

```
DROP TABLE HR.external.TABLE_1;
```

Déposer des bases de données

La commande **DROP DATABASE** supprime un catalogue de base de données, quel que soit son état (hors ligne, lecture seule, suspect, etc.), de l'instance SQL Server actuelle.

Une base de données ne peut pas être supprimée si des instantanés de base de données lui sont associés, car les instantanés de base de données doivent être supprimés en premier.

Une chute de base de données supprime tous les fichiers de disque physique (sauf si celui-ci est hors ligne) utilisés par la base de données, sauf si vous utilisez la procédure stockée 'sp_detach_db'.

Une chute d'instantané de base de données supprime l'instantané de l'instance SQL Server et supprime les fichiers physiques également utilisés.

Une base de données supprimée ne peut être recréée qu'en restaurant une sauvegarde (et non à partir d'un instantané de base de données).

La syntaxe

```
DROP DATABASE [ IF EXISTS ] { database_name | database_snapshot_name } [ ,...n ] [;]
```

- `IF EXISTS` - Supprimez la table uniquement si elle existe
- `database_name` - Spécifie le nom de la base de données à supprimer.
- `database_snapshot_name` - Spécifie l'instantané de base de données à supprimer
-

Exemples

Supprimer une base de données unique

```
DROP DATABASE Database1;
```

Suppression de plusieurs bases de données

```
DROP DATABASE Database1, Database2;
```

Supprimer un instantané

```
DROP DATABASE Database1_snapshot17;
```

Supprimer si la base de données existe

```
DROP DATABASE IF EXISTS Database1;
```

Déposer des tables temporaires

Dans SQL Server, nous avons 2 types de tables temporaires:

1. `##GlobalTempTable` est un type de table temporaire entre toutes les sessions de l'utilisateur.
2. `#LocalTempTable` temp tab - c'est un type de table temporaire qui n'existe que dans la portée actuelle (uniquement dans le processus actuel - vous pouvez obtenir l'ID de votre processus actuel par `SELECT @@SPID`)

Le processus de dépôt des tables temporaires est le même que pour la table normale:

```
DROP TABLE [ database_name . [ schema_name ] . | schema_name . ] table_name
```

AVANT SQL Server 2016:

```
IF(OBJECT_ID('tempdb..#TempTable') is not null)
    DROP TABLE #TempTable;
```

SQL Server 2016:

```
DROP TABLE IF EXISTS #TempTable
```

Lire Drop mot-clé en ligne: https://riptutorial.com/fr/sql-server/topic/9532/drop-mot-cle

Chapitre 32: Dynamic SQL Pivot

Introduction

Cette rubrique explique comment effectuer un pivot dynamique dans SQL Server.

Examples

Pivot SQL dynamique de base

```
if object_id('tempdb.dbo.#temp') is not null drop table #temp
create table #temp
(
    dateValue datetime,
    category varchar(3),
    amount decimal(36,2)
)

insert into #temp values ('1/1/2012', 'ABC', 1000.00)
insert into #temp values ('2/1/2012', 'DEF', 500.00)
insert into #temp values ('2/1/2012', 'GHI', 800.00)
insert into #temp values ('2/10/2012', 'DEF', 700.00)
insert into #temp values ('3/1/2012', 'ABC', 1100.00)


DECLARE
    @cols AS NVARCHAR(MAX),
    @query  AS NVARCHAR(MAX);

SET @cols = STUFF((SELECT distinct ',' + QUOTENAME(c.category)
            FROM #temp c
            FOR XML PATH(''), TYPE
            ).value('.', 'NVARCHAR(MAX)')
        ,1,1,'')

set @query = '
            SELECT
                dateValue,
                ' + @cols + '
            from
            (
                select
                     dateValue,
                     amount,
                     category
                from #temp
           ) x
            pivot
            (
                 sum(amount)
                for category in (' + @cols + ')
            ) p '


exec sp_executeSql @query
```

Lire Dynamic SQL Pivot en ligne: https://riptutorial.com/fr/sql-server/topic/10751/dynamic-sql-pivot

Chapitre 33: Ensemble de résultats de limite

Introduction

À mesure que les tables de base de données se développent, il est souvent utile de limiter les résultats des requêtes à un nombre ou à un pourcentage fixe. Cela peut être réalisé en utilisant le mot-clé `TOP` SQL Server ou la clause `OFFSET FETCH` .

Paramètres

Paramètre	Détails
`TOP`	Mot clé limitant Utiliser avec un numéro.
`PERCENT`	Mot clé en pourcentage. Vient après `TOP` et nombre limite.

Remarques

Si la clause `ORDER BY` est utilisée, la limitation s'applique au jeu de résultats ordonné.

Examples

Limiter avec TOP

Cet exemple limite `SELECT` result à 100 lignes.

```
SELECT TOP 100 *
FROM table_name;
```

Il est également possible d'utiliser une variable pour spécifier le nombre de lignes:

```
DECLARE @CountDesiredRows int = 100;
SELECT TOP (@CountDesiredRows) *
FROM table_name;
```

Limiter avec POURCENT

Cet exemple limite le résultat de `SELECT` à 15% du nombre total de lignes.

```
SELECT TOP 15 PERCENT *
FROM table_name
```

Limiter avec FETCH

SQL Server 2012

`FETCH` est généralement plus utile pour la pagination, mais peut être utilisé comme alternative à `TOP` :

```
SELECT *
FROM table_name
ORDER BY 1
OFFSET 0 ROWS
FETCH NEXT 50 ROWS ONLY
```

Lire Ensemble de résultats de limite en ligne: https://riptutorial.com/fr/sql-server/topic/1555/ensemble-de-resultats-de-limite

Chapitre 34: ESSAYEZ / CATCH

Remarques

TRY / CATCH est une construction de langage spécifique à T-SQL de MS SQL Server.

Il permet la gestion des erreurs dans T-SQL, similaire à celle observée dans le code .NET.

Examples

Transaction dans un TRY / CATCH

Cela annulera les deux insertions en raison d'un datetime non valide:

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, 'not a date', 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION -- First Rollback and then throw.
    THROW
END CATCH
```

Cela va valider les deux insertions:

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    THROW
    ROLLBACK TRANSACTION
END CATCH
```

Augmenter les erreurs dans le bloc try-catch

La fonction RAISERROR générera une erreur dans le bloc TRY CATCH:

```
DECLARE @msg nvarchar(50) = 'Here is a problem!'
BEGIN TRY
    print 'First statement';
    RAISERROR(@msg, 11, 1);
    print 'Second statement';
```

```
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
END CATCH
```

RAISERROR avec un deuxième paramètre supérieur à 10 (11 dans cet exemple) arrêtera l'exécution dans TRY BLOCK et générera une erreur qui sera traitée dans le bloc CATCH. Vous pouvez accéder à un message d'erreur à l'aide de la fonction ERROR_MESSAGE (). La sortie de cet échantillon est la suivante:

```
First statement
Error: Here is a problem!
```

Envoyer des messages d'information dans try catch block

RAISERROR avec sévérité (deuxième paramètre) inférieure ou égale à 10 ne lancera pas d'exception.

```
BEGIN TRY
    print 'First statement';
    RAISERROR( 'Here is a problem!', 10, 15);
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
END CATCH
```

Après l'instruction RAISERROR, la troisième instruction sera exécutée et le bloc CATCH ne sera pas appelé. Le résultat de l'exécution est le suivant:

```
First statement
Here is a problem!
Second statement
```

Exception de relance générée par RAISERROR

Vous pouvez relancer l'erreur que vous rencontrez dans le bloc CATCH en utilisant l'instruction TRHOW:

```
DECLARE @msg nvarchar(50) = 'Here is a problem! Area: ''%s'' Line:''%i'''
BEGIN TRY
    print 'First statement';
    RAISERROR(@msg, 11, 1, 'TRY BLOCK', 2);
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
    THROW;
END CATCH
```

Notez que dans ce cas, nous soulevons une erreur avec des arguments formatés (quatrième et

cinquième paramètre). Cela peut être utile si vous souhaitez ajouter plus d'informations dans le message. Le résultat de l'exécution est le suivant:

```
First statement
Error: Here is a problem! Area: 'TRY BLOCK' Line:'2'
Msg 50000, Level 11, State 1, Line 26
Here is a problem! Area: 'TRY BLOCK' Line:'2'
```

Lancer une exception dans les blocs TRY / CATCH

Vous pouvez lancer une exception dans try catch block:

```
DECLARE @msg nvarchar(50) = 'Here is a problem!'
BEGIN TRY
    print 'First statement';
    THROW 51000, @msg, 15;
    print 'Second statement';
END TRY
BEGIN CATCH
    print 'Error: ' + ERROR_MESSAGE();
    THROW;
END CATCH
```

Exception avec être traité dans le bloc CATCH, puis renvoyé en utilisant THROW sans paramètres.

```
First statement
Error: Here is a problem!
Msg 51000, Level 16, State 15, Line 39
Here is a problem!
```

THROW est similaire à RAISERROR avec les différences suivantes:

- Il est recommandé que les nouvelles applications utilisent THROW au lieu de RASIERROR.
- THROW peut utiliser n'importe quel nombre comme premier argument (numéro d'erreur), RAISERROR ne peut utiliser que des identifiants dans la vue sys.messages
- THROW a la gravité 16 (ne peut pas être modifié)
- THROW ne peut pas formater des arguments comme RAISERROR. Utilisez la fonction FORMATMESSAGE comme argument de RAISERROR si vous avez besoin de cette fonctionnalité.

Lire ESSAYEZ / CATCH en ligne: https://riptutorial.com/fr/sql-server/topic/5189/essayez---catch

Chapitre 35: Exporter des données dans un fichier txt à l'aide de SQLCMD

Syntaxe

- sqlcmd -S SHERAZM-E7450 \ SQL2008R2 -d Baseline_DB_Aug_2016 -o c: \ employee.txt -Q "select * from employee"

Examples

En utilisant SQLCMD sur l'invite de commandes

La structure de commande est

sqlcmd -S votre nom_serveur \ nom_instance -d nom_base_de_données -o nom_fichier_profil_withpath -Q "votre requête select"

Les commutateurs sont les suivants

-S pour le nom de serveur et le nom de l'instance

-d pour la base de données source

-o pour le fichier de sortie cible (il va créer un fichier de sortie)

-Q pour la requête pour récupérer des données

Lire Exporter des données dans un fichier txt à l'aide de SQLCMD en ligne: https://riptutorial.com/fr/sql-server/topic/7076/exporter-des-donnees-dans-un-fichier-txt-a-l-aide-de-sqlcmd

Chapitre 36: Expressions de table communes

Syntaxe

- WITH *cte_name* [(*nom_colonne_1* , *nom_colonne_2* , ...)] AS (*expression_cte*)

Remarques

Il est nécessaire de séparer un CTE de l'instruction précédente par un caractère point-virgule (`;`).

ie `;WITH CommonTableName (...) SELECT ... FROM CommonTableName ...`

La portée d'un CTE est un lot unique, et seulement en aval de sa définition. Un lot peut contenir plusieurs CTE et un CTE peut faire référence à un autre CTE défini précédemment dans le lot, mais un CTE peut ne pas faire référence à un autre CTE défini ultérieurement dans le lot.

Examples

Hiérarchie des employés

Configuration de la table

```
CREATE TABLE dbo.Employees
(
    EmployeeID INT NOT NULL PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    ManagerID INT NULL
)

GO

INSERT INTO Employees VALUES (101, 'Ken', 'Sánchez', NULL)
INSERT INTO Employees VALUES (102, 'Keith', 'Hall', 101)
INSERT INTO Employees VALUES (103, 'Fred', 'Bloggs', 101)
INSERT INTO Employees VALUES (104, 'Joseph', 'Walker', 102)
INSERT INTO Employees VALUES (105, 'Žydrė', 'Klybė', 101)
INSERT INTO Employees VALUES (106, 'Sam', 'Jackson', 105)
INSERT INTO Employees VALUES (107, 'Peter', 'Miller', 103)
INSERT INTO Employees VALUES (108, 'Chloe', 'Samuels', 105)
INSERT INTO Employees VALUES (109, 'George', 'Weasley', 105)
INSERT INTO Employees VALUES (110, 'Michael', 'Kensington', 106)
```

Expression de table commune

```
;WITH cteReports (EmpID, FirstName, LastName, SupervisorID, EmpLevel) AS
```

```
(
    SELECT EmployeeID, FirstName, LastName, ManagerID, 1
    FROM Employees
    WHERE ManagerID IS NULL

    UNION ALL

    SELECT e.EmployeeID, e.FirstName, e.LastName, e.ManagerID, r.EmpLevel + 1
    FROM Employees        AS e
    INNER JOIN cteReports AS r ON e.ManagerID = r.EmpID
)

SELECT
    FirstName + ' ' + LastName AS FullName,
    EmpLevel,
    (SELECT FirstName + ' ' + LastName FROM Employees WHERE EmployeeID =
cteReports.SupervisorID) AS ManagerName
FROM cteReports
ORDER BY EmpLevel, SupervisorID
```

Sortie:

Nom complet	EmpLevel	ManagerName
Ken Sánchez	1	*nul*
Keith Hall	2	Ken Sánchez
Fred Bloggs	2	Ken Sánchez
Žydre Klybe	2	Ken Sánchez
Joseph Walker	3	Keith Hall
Peter Miller	3	Fred Bloggs
Sam Jackson	3	Žydre Klybe
Chloé Samuels	3	Žydre Klybe
George Weasley	3	Žydre Klybe
Michael Kensington	4	Sam Jackson

Trouver le neuvième salaire le plus élevé en utilisant CTE

Table d'employés:

```
|  ID  | FirstName | LastName | Gender | Salary |
+------+-----------+----------+--------+--------+
| 1    | Jahangir  | Alam     | Male   | 70000  |
```

```
| 2    | Arifur    | Rahman   | Male   | 60000  |
| 3    | Oli       | Ahammed  | Male   | 45000  |
| 4    | Sima      | Sultana  | Female | 70000  |
| 5    | Sudeepta  | Roy      | Male   | 80000  |
+------+-----------+----------+--------+--------+
```

CTE (expression de table commune):

```
 WITH RESULT AS
(
    SELECT SALARY,
           DENSE_RANK() OVER (ORDER BY SALARY DESC) AS DENSERANK
    FROM EMPLOYEES
)
SELECT TOP 1 SALARY
FROM RESULT
WHERE DENSERANK = 1
```

Pour trouver le deuxième salaire le plus élevé, remplacez simplement N par 2. De même, pour trouver le troisième salaire le plus élevé, remplacez simplement N par 3.

Supprimer les lignes en double à l'aide de CTE

Table d'employés:

```
|  ID  | FirstName | LastName | Gender | Salary |
+------+-----------+----------+--------+--------+
|  1   | Mark      | Hastings | Male   | 60000  |
|  1   | Mark      | Hastings | Male   | 60000  |
|  2   | Mary      | Lambeth  | Female | 30000  |
|  2   | Mary      | Lambeth  | Female | 30000  |
|  3   | Ben       | Hoskins  | Male   | 70000  |
|  3   | Ben       | Hoskins  | Male   | 70000  |
|  3   | Ben       | Hoskins  | Male   | 70000  |
+------+-----------+----------+--------+--------+
```

CTE (expression de table commune):

```
WITH EmployeesCTE AS
(
   SELECT *, ROW_NUMBER()OVER(PARTITION BY ID ORDER BY ID) AS RowNumber
   FROM Employees
)
DELETE FROM EmployeesCTE WHERE RowNumber > 1
```

Résultat de l'exécution:

```
|  ID  | FirstName | LastName | Gender | Salary |
+------+-----------+----------+--------+--------+
|  1   | Mark      | Hastings | Male   | 60000  |
|  2   | Mary      | Lambeth  | Female | 30000  |
|  3   | Ben       | Hoskins  | Male   | 70000  |
+------+-----------+----------+--------+--------+
```

Générer une table de dates en utilisant CTE

```
DECLARE @startdate CHAR(8), @numberDays TINYINT

SET @startdate = '20160101'
SET @numberDays = 10;

WITH CTE_DatesTable
AS
(
  SELECT CAST(@startdate as date) AS [date]
  UNION ALL
  SELECT DATEADD(dd, 1, [date])
  FROM CTE_DatesTable
  WHERE DATEADD(dd, 1, [date]) <= DateAdd(DAY, @numberDays-1, @startdate)
)

SELECT [date] FROM CTE_DatesTable

OPTION (MAXRECURSION 0)
```

Cet exemple renvoie une table de dates à une seule colonne, en commençant par la date spécifiée dans la variable @startdate et en renvoyant la valeur de dates @numberDays suivante.

CTE récursif

Cet exemple montre comment obtenir chaque année de cette année à 2011 (2012 - 1).

```
WITH yearsAgo
(
    myYear
)
AS
(
     -- Base Case: This is where the recursion starts
     SELECT DATEPART(year, GETDATE()) AS myYear

     UNION ALL  -- This MUST be UNION ALL (cannot be UNION)

     -- Recursive Section: This is what we're doing with the recursive call
     SELECT yearsAgo.myYear - 1
     FROM yearsAgo
     WHERE yearsAgo.myYear >= 2012
)
     SELECT myYear FROM yearsAgo;  -- A single SELECT, INSERT, UPDATE, or DELETE
```

mon annee
2016
2015
2014
2013

mon annee
2012
2011

Vous pouvez contrôler la récursivité (débordement de la pile de réflexion dans le code) avec MAXRECURSION en tant qu'option de requête qui limitera le nombre d'appels récursifs.

```
WITH yearsAgo
(
    myYear
)
AS
(
     -- Base Case
     SELECT DATEPART(year , GETDATE()) AS myYear
     UNION ALL
     -- Recursive Section
     SELECT yearsAgo.myYear - 1
     FROM yearsAgo
     WHERE yearsAgo.myYear >= 2002
)
     SELECT * FROM yearsAgo
     OPTION (MAXRECURSION 10);
```

> Msg 530, niveau 16, état 1, ligne 2L'instruction s'est terminée. La récursivité maximale 10 a été épuisée avant la fin de l'instruction.

CTE avec plusieurs déclarations AS

```
;WITH cte_query_1
AS
(
    SELECT *
    FROM database.table1
),
cte_query_2
AS
(
    SELECT *
    FROM database.table2
)
SELECT *
FROM cte_query_1
WHERE cte_query_one.fk IN
(
    SELECT PK
    FROM cte_query_2
)
```

Avec les expressions de table communes, il est possible de créer plusieurs requêtes à l'aide d'instructions AS séparées par des virgules. Une requête peut alors référencer une ou toutes ces requêtes de différentes manières, même en les rejoignant.

Lire Expressions de table communes en ligne: https://riptutorial.com/fr/sql-server/topic/1343/expressions-de-table-communes

Chapitre 37: Filestream

Introduction

FILESTREAM intègre le moteur de base de données SQL Server à un système de fichiers NTFS en stockant les données BLOB (binary large object) varbinary (max) sous forme de fichiers sur le système de fichiers. Les instructions Transact-SQL peuvent insérer, mettre à jour, interroger, rechercher et sauvegarder des données FILESTREAM. Les interfaces du système de fichiers Win32 fournissent un accès en continu aux données.

Examples

Exemple

Source: MSDN https://technet.microsoft.com/en-us/library/bb933993(v=sql.105).aspx

Lire Filestream en ligne: https://riptutorial.com/fr/sql-server/topic/9509/filestream

Chapitre 38: Fonction Split String dans Sql Server

Examples

Diviser une chaîne dans Sql Server 2016

Dans **SQL Server 2016,** enfin, ils ont introduit la fonction de chaîne de fractionnement: `STRING_SPLIT`

Paramètres: accepte deux paramètres

Chaîne :

> Est une expression de n'importe quel type de caractère (par exemple, nvarchar, varchar, nchar ou char).

séparateur :

> Expression à caractère unique de n'importe quel type de caractère (par exemple, nvarchar (1), varchar (1), nchar (1) ou char (1)) utilisé comme séparateur pour les chaînes concaténées.

Remarque: Vous devez toujours vérifier si l'expression est une chaîne non vide.

Exemple:

```
Select Value
From STRING_SPLIT('a|b|c','|')
```

Dans l'exemple ci-dessus

```
String    : 'a|b|c'
separator : '|'
```

Résultat :

```
+-----+
|Value|
+-----+
|a    |
+-----+
|b    |
+-----+
|c    |
+-----+
```

Si c'est une chaîne vide:

```
SELECT value
FROM STRING_SPLIT('',',')
```

Résultat :

```
  +-----+
  |Value|
  +-----+
1 |     |
  +-----+
```

Vous pouvez éviter la situation ci-dessus en ajoutant une clause `WHERE`

```
SELECT value
FROM STRING_SPLIT('',',')
WHERE LTRIM(RTRIM(value))<>''
```

Séparer la chaîne dans Sql Server 2008/2012/2014 en utilisant XML

Comme il n'y a pas de fonction `STRING_SPLIT` , nous devons utiliser le hack XML pour diviser la chaîne en lignes:

Exemple:

```
SELECT split.a.value('.', 'VARCHAR(100)') AS Value
FROM   (SELECT Cast ('<M>' + Replace('A|B|C', '|', '</M><M>')+ '</M>' AS XML) AS Data) AS A
       CROSS apply data.nodes ('/M') AS Split(a);
```

Résultat:

```
+-----+
|Value|
+-----+
|A    |
+-----+
|B    |
+-----+
|C    |
+-----+
```

T-SQL Table variable et XML

```
Declare @userList Table(UserKey VARCHAR(60))
Insert into @userList values ('bill'),('jcom'),('others')
--Declared a table variable and insert 3 records

Declare @text XML
Select  @text = (
        select UserKey  from @userList for XML Path('user'), root('group')
)
--Set the XML value from Table
```

```
Select @text

--View the variable value
XML:
\<group>\<user>\<UserKey>bill\</UserKey>\</user>\<user>\<UserKey>jcom\</UserKey>\</user>\<user>\<UserKe
```

Lire Fonction Split String dans Sql Server en ligne: https://riptutorial.com/fr/sql-server/topic/3713/fonction-split-string-dans-sql-server

Chapitre 39: Fonctions d'agrégat

Introduction

Les fonctions d'agrégation dans SQL Server exécutent des calculs sur des ensembles de valeurs, renvoyant une valeur unique.

Syntaxe

- AVG (*expression* [ALL | DISTINCT])
- COUNT ([ALL | DISTINCT] *expression*)
- MAX (*expression* [ALL | DISTINCT])
- MIN (*expression* [ALL | DISTINCT])
- SUM ([ALL | DISTINCT] *expression*)

Examples

SOMME()

Renvoie la somme des valeurs numériques dans une colonne donnée.

Nous avons une table comme indiqué dans la figure qui sera utilisée pour effectuer différentes fonctions d'agrégat. Le nom de la table est *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select SUM(MarksObtained) From Marksheet
```

La fonction `sum` ne considère pas les lignes avec une valeur NULL dans le champ utilisé comme paramètre

Dans l'exemple ci-dessus, si nous avons une autre ligne comme celle-ci:

```
106    Italian    NULL
```

Cette ligne ne sera pas prise en compte dans le calcul de la somme

AVG ()

Renvoie la moyenne des valeurs numériques dans une colonne donnée.

Nous avons une table comme indiqué dans la figure qui sera utilisée pour effectuer différentes fonctions d'agrégat. Le nom de la table est *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select AVG(MarksObtained) From Marksheet
```

La fonction `average` ne prend pas en compte les lignes avec une valeur NULL dans le champ utilisé comme paramètre

Dans l'exemple ci-dessus, si nous avons une autre ligne comme celle-ci:

```
106    Italian    NULL
```

Cette ligne ne sera pas prise en compte dans le calcul moyen

MAX ()

Renvoie la plus grande valeur dans une colonne donnée.

Nous avons une table comme indiqué dans la figure qui sera utilisée pour effectuer différentes fonctions d'agrégat. Le nom de la table est *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select MAX(MarksObtained) From Marksheet
```

MIN ()

Renvoie la plus petite valeur dans une colonne donnée.

Nous avons une table comme indiqué dans la figure qui sera utilisée pour effectuer différentes fonctions d'agrégat. Le nom de la table est *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select MIN(MarksObtained) From Marksheet
```

COMPTER()

Renvoie le nombre total de valeurs dans une colonne donnée.

Nous avons une table comme indiqué dans la figure qui sera utilisée pour effectuer différentes fonctions d'agrégat. Le nom de la table est *Marksheet* .

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

```
Select COUNT(MarksObtained) From Marksheet
```

La fonction `count` ne prend pas en compte les lignes avec une valeur NULL dans le champ utilisé comme paramètre. Généralement, le paramètre count est * (tous les champs), donc seulement si tous les champs de la ligne sont NULL, cette ligne ne sera pas prise en compte.

Dans l'exemple ci-dessus, si nous avons une autre ligne comme celle-ci:

```
106    Italian    NULL
```

Cette ligne ne sera pas prise en compte dans le calcul du compte

REMARQUE

La fonction `COUNT(*)` renvoie le nombre de lignes d'une table. Cette valeur peut également être obtenue en utilisant une expression constante non nulle qui ne contient aucune référence de colonne, telle que `COUNT(1)` .

Exemple

```
Select COUNT(1) From Marksheet
```

COUNT (Nom_Colonne) avec GROUP BY Nom_Colonne

La plupart du temps, nous aimons obtenir le nombre total d'occurrences d'une valeur de colonne

dans une table, par exemple:

NOM DE LA TABLE: RAPPORTS

ReportName	RapportPrix
Tester	10.00 $
Tester	10.00 $
Tester	10.00 $
Test 2	11.00 $
Tester	10.00 $
Test 3	14.00 $
Test 3	14.00 $
Test 4	100.00 $

```
SELECT
    ReportName AS REPORT NAME,
    COUNT(ReportName) AS COUNT
FROM
    REPORTS
GROUP BY
    ReportName
```

NOM DU RAPPORT	COMPTER
Tester	4
Test 2	1
Test 3	2
Test 4	1

Lire Fonctions d'agrégat en ligne: https://riptutorial.com/fr/sql-server/topic/5802/fonctions-d-agregat

Chapitre 40: Fonctions d'agrégation de chaînes dans SQL Server

Examples

Utilisation de STUFF pour l'agrégation de chaînes

Nous avons une table Student avec SubjectId. Ici, il faut concaténer en fonction de subjectId.

Toutes les versions de SQL Server

```
create table #yourstudent (subjectid int, studentname varchar(10))

insert into #yourstudent (subjectid, studentname) values
 ( 1       ,'Mary'    )
,( 1       ,'John'    )
,( 1       ,'Sam'    )
,( 2       ,'Alaina')
,( 2       ,'Edward')

select subjectid,  stuff(( select concat( ',', studentname) from #yourstudent y where
y.subjectid = u.subjectid for xml path('')),1,1, '')
    from #yourstudent u
    group by subjectid
```

String_Agg pour l'agrégation de chaînes

Dans le cas de SQL Server 2017 ou de vnext, nous pouvons utiliser STRING_AGG intégré pour cette agrégation. Pour la même table d'étudiant,

```
create table #yourstudent (subjectid int, studentname varchar(10))

insert into #yourstudent (subjectid, studentname) values
 ( 1       ,'Mary'    )
,( 1       ,'John'    )
,( 1       ,'Sam'    )
,( 2       ,'Alaina')
,( 2       ,'Edward')

select subjectid, string_agg(studentname, ',') from #yourstudent
    group by subjectid
```

Lire Fonctions d'agrégation de chaînes dans SQL Server en ligne: https://riptutorial.com/fr/sql-server/topic/9892/fonctions-d-agregation-de-chaines-dans-sql-server

Chapitre 41: Fonctions de chaîne

Remarques

Liste des fonctions de chaîne (triées par ordre alphabétique):

- Ascii
- Carboniser
- Charindex
- Concat
- Différence
- Format
- La gauche
- Len
- Inférieur
- Ltrim
- Nchar
- Patindex
- Quotename
- Remplacer
- Reproduire
- Sens inverse
- Droite
- Rtrim
- Soundex
- Espace
- Str
- String_escape

- String_split
- Des trucs
- Substring
- Unicode
- Plus haut

Examples

La gauche

Renvoie une sous-chaîne commençant par le caractère le plus à gauche d'une chaîne et jusqu'à la longueur maximale spécifiée.

Paramètres:

1. expression de caractère. L'expression de caractère peut être de tout type de données pouvant être implicitement converti en `varchar` ou `nvarchar` , sauf pour `text` ou `ntext`
2. longueur maximale. Un nombre entier compris entre 0 et `bigint` valeur max (9,223,372,036,854,775,807).
   Si le paramètre de longueur maximale est négatif, une erreur sera générée.

```
SELECT LEFT('This is my string', 4) -- result: 'This'
```

Si la longueur maximale est supérieure au nombre de caractères de la chaîne, la chaîne entière est renvoyée.

```
SELECT LEFT('This is my string', 50) -- result: 'This is my string'
```

Droite

Renvoie une sous-chaîne qui est la partie la plus à droite de la chaîne, avec la longueur maximale spécifiée.

Paramètres:

1. expression de caractère. L'expression de caractère peut être de tout type de données pouvant être implicitement converti en `varchar` ou `nvarchar` , sauf pour `text` ou `ntext`
2. longueur maximale. Un nombre entier compris entre 0 et `bigint` valeur max (9,223,372,036,854,775,807). Si le paramètre de longueur maximale est négatif, une erreur sera générée.

```
SELECT RIGHT('This is my string', 6) -- returns 'string'
```

Si la longueur maximale est supérieure au nombre de caractères de la chaîne, la chaîne entière

est renvoyée.

```
SELECT RIGHT('This is my string', 50) -- returns 'This is my string'
```

Substring

Renvoie une sous-chaîne qui commence par le caractère qui se trouve dans l'index de démarrage spécifié et la longueur maximale spécifiée.

Paramètres:

1. Expression de caractère. L'expression de caractère peut être de n'importe quel type de données pouvant être implicitement converti en `varchar` ou `nvarchar` , sauf pour `text` ou `ntext` .
2. Index de départ. Un nombre (`int` ou `bigint`) qui spécifie l'index de démarrage de la sous-chaîne demandée. (**Remarque: les** chaînes du serveur SQL sont des index de base 1, ce qui signifie que le premier caractère de la chaîne est l'index 1). Ce nombre peut être inférieur à 1. Dans ce cas, si la somme de l'index de départ et de la longueur maximale est supérieure à 0, la chaîne de retour serait une chaîne commençant par le premier caractère de l'expression + longueur max - 1). Si c'est moins que 0, une chaîne vide sera renvoyée.
3. Longueur maximale. Un nombre entier compris entre 0 et `bigint` valeur max (9,223,372,036,854,775,807). Si le paramètre de longueur maximale est négatif, une erreur sera générée.

```
SELECT SUBSTRING('This is my string', 6, 5) -- returns 'is my'
```

Si la longueur maximale + l'index de début est supérieur au nombre de caractères de la chaîne, la chaîne entière est renvoyée.

```
SELECT SUBSTRING('Hello World',1,100) -- returns 'Hello World'
```

Si l'index de démarrage est plus grand que le nombre de caractères de la chaîne, une chaîne vide est renvoyée.

```
SELECT SUBSTRING('Hello World',15,10) -- returns ''
```

ASCII

Renvoie une valeur int représentant le code ASCII du caractère le plus à gauche d'une chaîne.

```
SELECT ASCII('t') -- Returns 116
SELECT ASCII('T') -- Returns 84
SELECT ASCII('This') -- Returns 84
```

Si la chaîne est Unicode et que le caractère le plus à gauche n'est pas ASCII mais peut être représenté dans le classement actuel, une valeur supérieure à 127 peut être renvoyée:

```
SELECT ASCII(N'ï') -- returns 239 when `SERVERPROPERTY('COLLATION') =
'SQL_Latin1_General_CP1_CI_AS'`
```

Si la chaîne est Unicode et que le caractère le plus à gauche ne peut pas être représenté dans le classement actuel, la valeur int de 63 est renvoyée: (qui représente le point d'interrogation en ASCII):

```
SELECT ASCII(N'□') -- returns 63
SELECT ASCII(nchar(2039)) -- returns 63
```

CharIndex

Renvoie l'index de début de la première occurrence d'une expression de chaîne dans une autre expression de chaîne.

Liste de paramètres:

1. Chaîne à trouver (jusqu'à 8000 caractères)
2. Chaîne à rechercher (tout type et longueur de données de caractère valide, y compris binaire)
3. (Facultatif) index pour démarrer. Un nombre de type int ou big int. Si omis ou moins de 1, la recherche commence au début de la chaîne.

Si la chaîne à rechercher est `varchar(max)` , `nvarchar(max)` ou `varbinary(max)` , la fonction `CHARINDEX` renverra une valeur `bigint` . Sinon, il retournera un `int` .

```
SELECT CHARINDEX('is', 'this is my string') -- returns 3
SELECT CHARINDEX('is', 'this is my string', 4) -- returns 6
SELECT CHARINDEX(' is', 'this is my string') -- returns 5
```

Carboniser

Retourne un caractère représenté par un code int ASCII.

```
SELECT CHAR(116) -- Returns 't'
SELECT CHAR(84)  -- Returns 'T'
```

Cela peut être utilisé pour introduire de nouvelles lignes / lignes `CHAR(10)` , le chariot retourne `CHAR(13)` , etc. Voir AsciiTable.com pour référence.

Si la valeur de l'argument n'est pas comprise entre 0 et 255, la fonction CHAR renvoie `NULL` .
Le type de données renvoyé par la fonction `CHAR` est `char(1)`

Len

Renvoie le nombre de caractères d'une chaîne.
Remarque: la fonction `LEN` ignore les espaces de fin:

```
SELECT LEN('My string'), -- returns 9
       LEN('My string   '), -- returns 9
       LEN('   My string') -- returns 12
```

Si la longueur, y compris les espaces de fuite, est souhaitée, il existe plusieurs techniques pour y parvenir, bien que chacune ait ses inconvénients. Une technique consiste à ajouter un seul caractère à la chaîne, puis à utiliser le `LEN` moins un:

```
DECLARE @str varchar(100) = 'My string   '
SELECT LEN(@str + 'x') - 1 -- returns 12
```

L'inconvénient est que si le type de la variable de chaîne ou de la colonne est de la longueur maximale, l'ajout du caractère supplémentaire est ignoré et la longueur résultante ne compte toujours pas les espaces de fin. Pour remédier à cela, la version modifiée suivante résout le problème et donne les résultats corrects dans tous les cas, au prix d'une petite durée d'exécution supplémentaire (résultats corrects, y compris avec des paires de substitution et une vitesse d'exécution raisonnable). semble être la meilleure technique à utiliser:

```
SELECT LEN(CONVERT(NVARCHAR(MAX), @str) + 'x') - 1
```

Une autre technique consiste à utiliser la fonction `DATALENGTH` .

```
DECLARE @str varchar(100) = 'My string   '
SELECT DATALENGTH(@str) -- returns 12
```

Il est important de noter que `DATALENGTH` renvoie la longueur en octets de la chaîne en mémoire. Ce sera différent pour `varchar` vs `nvarchar` .

```
DECLARE @str nvarchar(100) = 'My string   '
SELECT DATALENGTH(@str) -- returns 24
```

Vous pouvez ajuster cela en divisant la longueur de données de la chaîne par la longueur de données d'un seul caractère (qui doit être du même type). L'exemple ci-dessous le fait, et gère également le cas où la chaîne cible est vide, évitant ainsi une division par zéro.

```
DECLARE @str nvarchar(100) = 'My string   '
SELECT DATALENGTH(@str) / DATALENGTH(LEFT(LEFT(@str, 1) + 'x', 1)) -- returns 12
```

Même cela, cependant, a un problème dans SQL Server 2012 et supérieur. Il produira des résultats incorrects lorsque la chaîne contient des paires de substitution (certains caractères peuvent occuper plus d'octets que d'autres caractères dans la même chaîne).

Une autre technique consiste à utiliser `REPLACE` pour convertir des espaces en un caractère non-espace et prendre la `LEN` d' `LEN` du résultat. Cela donne des résultats corrects dans tous les cas, mais a une très faible vitesse d'exécution avec de longues chaînes.

Concat

SQL Server 2012

Retourne une chaîne qui est le résultat de deux chaînes ou plus jointes. `CONCAT` accepte deux arguments ou plus.

```
SELECT CONCAT('This', ' is', ' my', ' string') -- returns 'This is my string'
```

Remarque: Contrairement à la concaténation de chaînes à l'aide de l'opérateur de concaténation de chaînes (`+`), lors du passage d'une valeur NULL à la fonction `concat` , elle le convertit implicitement en une chaîne vide:

```
SELECT CONCAT('This', NULL, ' is', ' my', ' string'), -- returns 'This is my string'
       'This' + NULL + ' is' + ' my' + ' string' -- returns NULL.
```

Les arguments d'un type non-chaîne seront également convertis implicitement en chaîne:

```
SELECT CONCAT('This', ' is my ', 3, 'rd string') -- returns 'This is my 3rd string'
```

Les variables de type non-chaîne seront également converties en format chaîne, inutile de les convertir manuellement en chaîne:

```
DECLARE @Age INT=23;
SELECT CONCAT('Ram is ', @Age,' years old');  -- returns 'Ram is 23 years old'
```

SQL Server 2012

Les anciennes versions ne prennent pas en charge la fonction `CONCAT` et doivent utiliser l'opérateur de concaténation de chaînes (`+`) à la place. Les types non-chaîne doivent être convertis ou convertis en types de chaînes afin de les concaténer de cette manière.

```
SELECT 'This is the number ' + CAST(42 AS VARCHAR(5)) --returns 'This is the number 42'
```

Inférieur

Renvoie une expression de caractère (`varchar` ou `nvarchar`) après avoir converti tous les caractères majuscules en minuscules.

Paramètres:

1. Expression de caractère. Toute expression de caractère ou de données binaires pouvant être implicitement convertie en `varchar` .

```
SELECT LOWER('This IS my STRING') -- Returns 'this is my string'

DECLARE @String nchar(17) = N'This IS my STRING';
SELECT LOWER(@String) -- Returns 'this is my string'
```

Plus haut

Renvoie une expression de caractère (`varchar` ou `nvarchar`) après avoir converti tous les

caractères minuscules en majuscules.

Paramètres:

1. Expression de caractère. Toute expression de caractère ou de données binaires pouvant être implicitement convertie en `varchar` .

```
SELECT UPPER('This IS my STRING') -- Returns 'THIS IS MY STRING'

DECLARE @String nchar(17) = N'This IS my STRING';
SELECT UPPER(@String) -- Returns 'THIS IS MY STRING'
```

LTrim

Renvoie une expression de caractère (`varchar` ou `nvarchar`) après avoir supprimé tous les espaces blancs en tête, c.-à-d. Les espaces blancs de gauche à droite du premier caractère non blanc.

Paramètres:

1. expression de caractère. Toute expression de données de caractère ou binaire pouvant être implicitement convertie en `varcher` , à l'exception de `text` , `ntext` et `image` .

```
SELECT LTRIM('    This is my string') -- Returns 'This is my string'
```

RTrim

Retourne une expression de caractère (`varchar` ou `nvarchar`) après avoir supprimé tous les espaces blancs à la fin, c'est-à-dire les espaces de l'extrémité droite de la chaîne jusqu'au premier caractère d'espacement non blanc à gauche.

Paramètres:

1. expression de caractère. Toute expression de données de caractère ou binaire pouvant être implicitement convertie en `varcher` , à l'exception de `text` , `ntext` et `image` .

```
SELECT RTRIM('This is my string     ') -- Returns 'This is my string'
```

Unicode

Renvoie la valeur entière représentant la valeur Unicode du premier caractère de l'expression d'entrée.

Paramètres:

1. Expression de caractères Unicode. Toute expression `nchar` ou `nvarchar` valide.

```
SELECT UNICODE(N'Ɛ') -- Returns 400
```

```
DECLARE @Unicode nvarchar(11) = N'Ɛ is a char'
SELECT UNICODE(@Unicode) -- Returns 400
```

NChar

Renvoie le ou les caractères Unicode (`nchar(1)` ou `nvarchar(2)`) correspondant à l'argument entier qu'il reçoit, tel que défini par la norme Unicode.

Paramètres:

1. expression entière. Toute expression entière qui est un nombre positif entre 0 et 65535, ou si le classement de la base de données prend en charge le caractère supplémentaire (CS) drapeau, la plage prise en charge est comprise entre 0 et 1114111. Si l'expression entière ne tombe pas dans cette plage, `null` est revenu.

```
SELECT NCHAR(257) -- Returns 'ā'
SELECT NCHAR(400) -- Returns 'Ɛ'
```

Sens inverse

Renvoie une valeur de chaîne dans l'ordre inverse.

Paramètres:

1. expression de chaîne. Toute chaîne ou donnée binaire pouvant être implicitement convertie en `varchar` .

```
Select REVERSE('Sql Server') -- Returns 'revreS lqS'
```

PatIndex

Renvoie la position de départ de la première occurrence du modèle spécifié dans l'expression spécifiée.

Paramètres:

1. modèle. Une expression de caractère contient la séquence à rechercher. Limité à une longueur maximale de 8000 caractères. Les caractères génériques (`%` , _) peuvent être utilisés dans le motif. Si le motif ne commence pas par un caractère générique, il ne peut correspondre qu'au début de l'expression. S'il ne se termine pas par un caractère générique, il ne peut correspondre qu'à ce qui se trouve à la fin de l'expression.

2. expression. Tout type de données de chaîne.

```
SELECT PATINDEX('%ter%', 'interesting') -- Returns 3.

SELECT PATINDEX('%t_r%t%', 'interesting') -- Returns 3.

SELECT PATINDEX('ter%', 'interesting') -- Returns 0, since 'ter' is not at the start.
```

```
SELECT PATINDEX('inter%', 'interesting') -- Returns 1.

SELECT PATINDEX('%ing', 'interesting') -- Returns 9.
```

Espace

Retourne une chaîne (`varchar`) d'espaces répétés.

Paramètres:

1. expression entière. Toute expression entière, jusqu'à 8000. Si la `null` est négative, la valeur `null` est renvoyée. si 0, une chaîne vide est renvoyée. (Pour retourner une chaîne plus longue que 8000 espaces, utilisez Replicate.

```
SELECT SPACE(-1) -- Returns NULL
SELECT SPACE(0)  -- Returns an empty string
SELECT SPACE(3)  -- Returns '   ' (a string containing 3 spaces)
```

Reproduire

Répète une valeur de chaîne un nombre de fois spécifié.

Paramètres:

1. expression de chaîne. L'expression de chaîne peut être une chaîne de caractères ou des données binaires.
2. expression entière. Tout type entier, y compris `bigint` . Si négatif, `null` est renvoyé. Si 0, une chaîne vide est renvoyée.

```
SELECT REPLICATE('a', -1)  -- Returns NULL

SELECT REPLICATE('a', 0)  -- Returns ''

SELECT REPLICATE('a', 5)  -- Returns 'aaaaa'

SELECT REPLICATE('Abc', 3) -- Returns 'AbcAbcAbc'
```

Remarque: Si l'expression de chaîne n'est pas de type `varchar(max)` ou `nvarchar(max)` , la valeur de retour ne dépassera pas 8 000 caractères. Replicate s'arrêtera avant d'ajouter la chaîne qui entraînera un dépassement de la valeur de retour:

```
SELECT LEN(REPLICATE('a b c d e f g h i j k l', 350)) -- Returns 7981

SELECT LEN(REPLICATE(cast('a b c d e f g h i j k l' as varchar(max)), 350)) -- Returns 8050
```

Remplacer

Renvoie une chaîne (`varchar` ou `nvarchar`) où toutes les occurrences d'une sous-chaîne spécifiée sont remplacées par une autre sous-chaîne.

Paramètres:

1. expression de chaîne. C'est la chaîne qui serait recherchée. Il peut s'agir d'un type de données caractère ou binaire.
2. modèle. C'est la sous-chaîne qui serait remplacée. Il peut s'agir d'un type de données caractère ou binaire. L'argument pattern ne peut pas être une chaîne vide.
3. remplacement. C'est la sous-chaîne qui remplacerait la sous-chaîne de motif. Cela peut être un caractère ou des données binaires.

```
SELECT REPLACE('This is my string', 'is', 'XX') -- Returns 'ThXX XX my string'.
```

Remarques:

- Si l'expression de chaîne n'est pas de type `varchar(max)` ou `nvarchar(max)` , la fonction `replace` tronque la valeur de retour à 8 000 caractères.
- Le type de données renvoyé dépend des types de données d'entrée - renvoie `nvarchar` si l'une des valeurs d'entrée est `nvarchar` ou `varchar` sinon.
- Renvoie `NULL` si l'un des paramètres d'entrée est `NULL`

String_Split

SQL Server 2016

Divise une expression de chaîne à l'aide d'un séparateur de caractères. Notez que `STRING_SPLIT()` est une fonction table et doit donc être utilisée dans la clause `FROM` .

Paramètres:

1. chaîne. Toute expression de type caractère (`char` , `nchar` , `varchar` ou `nvarchar`)
2. séparateur. Une expression de caractère unique de n'importe quel type (`char(1)` , `nchar(1)` , `varchar(1)` ou `nvarchar(1)`).

Retourne une table de colonne unique où chaque ligne contient un fragment de la chaîne. Le nom des colonnes est `value` et le type de données est `nvarchar` si l'un des paramètres est `nchar` ou `nvarchar` , sinon `varchar` .

L'exemple suivant sépare une chaîne en utilisant l'espace comme séparateur:

```
SELECT value FROM STRING_SPLIT('Lorem ipsum dolor sit amet.', ' ');
```

Résultat:

```
value
-----
Lorem
ipsum
dolor
sit
amet.
```

Remarques:

> La fonction `STRING_SPLIT` est disponible uniquement sous le niveau de compatibilité **130** . Si le niveau de compatibilité de votre base de données est inférieur à 130, SQL Server ne pourra pas trouver et exécuter la fonction `STRING_SPLIT` . Vous pouvez modifier le niveau de compatibilité d'une base de données à l'aide de la commande suivante:

```
ALTER DATABASE [database_name] SET COMPATIBILITY_LEVEL = 130
```

SQL Server 2016

Les anciennes versions du serveur SQL n'ont pas de fonction de chaîne de caractères intégrée. Il existe de nombreuses fonctions définies par l'utilisateur qui traitent le problème de la division d'une chaîne. Vous pouvez lire l'article d'Aaron Bertrand « Split strings» dans le bon sens - ou la meilleure façon de comparer certaines d'entre elles.

Str

Renvoie les données de caractères (`varchar`) converties à partir de données numériques.

Paramètres:

1. expression flottante. Un type de données numérique approximatif avec un point décimal.
2. longueur. **optionnel.** La longueur totale de l'expression de chaîne qui renverrait, y compris les chiffres, le point décimal et les espaces de début (si nécessaire). La valeur par défaut est 10.
3. décimal. **optionnel.** Le nombre de chiffres à droite du séparateur décimal. Si elle est supérieure à 16, le résultat sera tronqué à seize places à droite du point décimal.

```
SELECT STR(1.2) -- Returns '         1'

SELECT STR(1.2, 3) -- Returns '  1'

SELECT STR(1.2, 3, 2) -- Returns '1.2'

SELECT STR(1.2, 5, 2) -- Returns ' 1.20'

SELECT STR(1.2, 5, 5) -- Returns '1.200'

SELECT STR(1, 5, 2) -- Returns ' 1.00'

SELECT STR(1) -- Returns '         1'
```

Quotename

Renvoie une chaîne Unicode entourée de délimiteurs pour en faire un identificateur SQL Server délimité valide.

Paramètres:

1. chaîne de caractères. Chaîne de données Unicode, 128 caractères maximum (`sysname`). Si une chaîne de saisie dépasse 128 caractères, la fonction renvoie `null` .
2. personnage de citation. **Facultatif** Un seul caractère à utiliser comme délimiteur. Peut - être un guillemet simple (`'` ou ``), un crochet gauche ou à droite (`{` , `[` , `(` , `<` ou `>` , `)` , `]` , `}`) ou une double guillemet (`"`). Toute autre valeur renvoie null La valeur par défaut est entre crochets.

```
SELECT QUOTENAME('what''s my name?')      -- Returns [what's my name?]

SELECT QUOTENAME('what''s my name?', '[') -- Returns [what's my name?]
SELECT QUOTENAME('what''s my name?', ']') -- Returns [what's my name?]

SELECT QUOTENAME('what''s my name?', '''') -- Returns 'what''s my name?'

SELECT QUOTENAME('what''s my name?', '"') -- Returns "what's my name?"

SELECT QUOTENAME('what''s my name?', ')') -- Returns (what's my name?)
SELECT QUOTENAME('what''s my name?', '(') -- Returns (what's my name?)

SELECT QUOTENAME('what''s my name?', '<') -- Returns <what's my name?>
SELECT QUOTENAME('what''s my name?', '>') -- Returns <what's my name?>

SELECT QUOTENAME('what''s my name?', '{') -- Returns {what's my name?}
SELECT QUOTENAME('what''s my name?', '}') -- Returns {what's my name?}

SELECT QUOTENAME('what''s my name?', '`') -- Returns `what's my name?`
```

Soundex

Renvoie un code à quatre caractères (`varchar`) pour évaluer la similarité phonétique de deux chaînes.

Paramètres:

1. expression de caractère. Une expression alphanumérique de données de caractères.

La fonction soundex crée un code à quatre caractères basé sur la manière dont l'expression des caractères retentirait au moment de son utilisation. le premier caractère est la version majuscule du premier caractère du paramètre, les 3 autres sont des nombres représentant les lettres de l'expression (sauf a, e, i, o, u, h, w et y sont ignorés) .

```
SELECT SOUNDEX ('Smith') -- Returns 'S530'

SELECT SOUNDEX ('Smythe') -- Returns 'S530'
```

Différence

Renvoie une valeur entière (`int`) qui indique la différence entre les valeurs soundex de deux expressions de caractère.

Paramètres:

1. expression de caractère 1.
2. expression de caractère 2.

Les deux paramètres sont des expressions alphanumériques des données de caractère.

L'entier retourné est le nombre de caractères dans les valeurs soundex des paramètres identiques, donc 4 signifie que les expressions sont très similaires et 0 signifie qu'elles sont très différentes.

```
SELECT  SOUNDEX('Green'),  -- G650
        SOUNDEX('Greene'),  -- G650
        DIFFERENCE('Green','Greene') -- Returns 4

SELECT  SOUNDEX('Blotchet-Halls'),  -- B432
        SOUNDEX('Greene'),  -- G650
        DIFFERENCE('Blotchet-Halls', 'Greene') -- Returns 0
```

Format

SQL Server 2012

Retourne une valeur `NVARCHAR` formatée avec le format et la culture spécifiés (si spécifié). Ceci est principalement utilisé pour convertir les types date-heure en chaînes.

Paramètres:

1. `value` . Une expression d'un type de données pris en charge à formater. les types valides sont listés ci-dessous.
2. `format` Un `NVARCHAR` format `NVARCHAR` . Voir la documentation officielle Microsoft pour les chaînes au format standard et personnalisé .
3. `culture` **Facultatif** argument `nvarchar` spécifiant une culture. La valeur par défaut est la culture de la session en cours.

RENDEZ-VOUS AMOUREUX

En utilisant des chaînes de format standard:

```
DECLARE @d DATETIME = '2016-07-31';

SELECT
    FORMAT ( @d, 'd', 'en-US' ) AS 'US English Result' -- Returns '7/31/2016'
   ,FORMAT ( @d, 'd', 'en-gb' ) AS 'Great Britain English Result' -- Returns '31/07/2016'
   ,FORMAT ( @d, 'd', 'de-de' ) AS 'German Result' -- Returns '31.07.2016'
   ,FORMAT ( @d, 'd', 'zh-cn' ) AS 'Simplified Chinese (PRC) Result' -- Returns '2016/7/31'
   ,FORMAT ( @d, 'D', 'en-US' ) AS 'US English Result' -- Returns 'Sunday, July 31, 2016'
   ,FORMAT ( @d, 'D', 'en-gb' ) AS 'Great Britain English Result' -- Returns '31 July 2016'
   ,FORMAT ( @d, 'D', 'de-de' ) AS 'German Result' -- Returns 'Sonntag, 31. Juli 2016'
```

Utiliser des chaînes de format personnalisées:

```
SELECT FORMAT( @d, 'dd/MM/yyyy', 'en-US' ) AS 'DateTime Result' -- Returns '31/07/2016'
      ,FORMAT(123456789,'###-##-####') AS 'Custom Number Result' -- Returns '123-45-6789',
```

```
      ,FORMAT( @d,'dddd, MMMM dd, yyyy hh:mm:ss tt','en-US') AS 'US' -- Returns 'Sunday, July
31, 2016 12:00:00 AM'
      ,FORMAT( @d,'dddd, MMMM dd, yyyy hh:mm:ss tt','hi-IN') AS 'Hindi' -- Returns रविवार, जुलाई 31,
2016 12:00:00 पूर्वाह्न
   ,FORMAT ( @d, 'dddd', 'en-US' )  AS 'US' -- Returns 'Sunday'
      ,FORMAT ( @d, 'dddd', 'hi-IN' )  AS 'Hindi' -- Returns 'रविवार'
```

`FORMAT` peut également être utilisé pour formater `CURRENCY` , `PERCENTAGE` et `NUMBERS` .

DEVISE

```
DECLARE @Price1 INT = 40
SELECT FORMAT(@Price1,'c','en-US') AS 'CURRENCY IN US Culture' -- Returns '$40.00'
       ,FORMAT(@Price1,'c','de-DE') AS 'CURRENCY IN GERMAN Culture' -- Returns '40,00 €'
```

Nous pouvons spécifier le nombre de chiffres après la décimale.

```
DECLARE @Price DECIMAL(5,3) = 40.356
SELECT FORMAT( @Price, 'C') AS 'Default', -- Returns '$40.36'
       FORMAT( @Price, 'C0') AS 'With 0 Decimal', -- Returns '$40'
       FORMAT( @Price, 'C1') AS 'With 1 Decimal', -- Returns '$40.4'
       FORMAT( @Price, 'C2') AS 'With 2 Decimal', -- Returns '$40.36'
```

POURCENTAGE

```
   DECLARE @Percentage float = 0.35674
   SELECT FORMAT( @Percentage, 'P') AS '% Default', -- Returns '35.67 %'
   FORMAT( @Percentage, 'P0') AS '% With 0 Decimal', -- Returns '36 %'
   FORMAT( @Percentage, 'P1') AS '% with 1 Decimal'  -- Returns '35.7 %'
```

NOMBRE

```
DECLARE @Number AS DECIMAL(10,2) = 454545.389
SELECT FORMAT( @Number, 'N','en-US') AS 'Number Format in US', -- Returns '454,545.39'
FORMAT( @Number, 'N','en-IN')  AS 'Number Format in INDIA', -- Returns '4,54,545.39'
FORMAT( @Number, '#.0')     AS 'With 1 Decimal', -- Returns '454545.4'
FORMAT( @Number, '#.00')    AS 'With 2 Decimal', -- Returns '454545.39'
FORMAT( @Number, '#,##.00') AS 'With Comma and 2 Decimal', -- Returns '454,545.39'
FORMAT( @Number, '##.00')   AS 'Without Comma and 2 Decimal', -- Returns '454545.39'
FORMAT( @Number, '000000000') AS 'Left-padded to nine digits' -- Returns '000454545'
```

Liste de types de valeurs valides: (source)

```
Category         Type             .Net type
-----------------------------------------
Numeric          bigint           Int64
Numeric          int              Int32
Numeric          smallint         Int16
Numeric          tinyint          Byte
Numeric          decimal          SqlDecimal
Numeric          numeric          SqlDecimal
Numeric          float            Double
Numeric          real             Single
Numeric          smallmoney       Decimal
```

```
Numeric          money            Decimal
Date and Time    date             DateTime
Date and Time    time             TimeSpan
Date and Time    datetime         DateTime
Date and Time    smalldatetime    DateTime
Date and Time    datetime2        DateTime
Date and Time    datetimeoffset   DateTimeOffset
```

Notes IMPORTANTES:

- `FORMAT` renvoie `NULL` pour les erreurs autres qu'une culture non valide. Par exemple, `NULL` est renvoyé si la valeur spécifiée dans le format n'est pas valide.
- `FORMAT` repose sur la présence du CLR (Common Language Runtime) .NET Framework.
- `FORMAT` s'appuie sur les règles de formatage CLR qui imposent que les deux-points et les points doivent être échappés. Par conséquent, lorsque la chaîne de format (deuxième paramètre) contient deux points ou un point, le signe deux-points ou le point doivent être précédés d'une barre oblique inverse lorsqu'une valeur d'entrée (premier paramètre) est du type de données temporel.

Voir aussi Formatage de la date et de l'heure à l'aide de l' exemple de documentation FORMAT .

String_escape

SQL Server 2016

Échappe les caractères spéciaux dans les textes et renvoie du texte (`nvarchar(max)`) avec des caractères échappés.

Paramètres:

1. texte. est une expression `nvarchar` représentant la chaîne à échapper.
2. type. Règles d'évasion qui seront appliquées. Actuellement, la seule valeur prise en charge est `'json'` .

```
SELECT STRING_ESCAPE('\   /
\\    "     ', 'json') -- returns '\\\t\/\n\\\\\t\"\t'
```

Liste des caractères qui seront échappés:

```
Special character    Encoded sequence
--------------------------------------
Quotation mark (")   \"
Reverse solidus (\)  \\
Solidus (/)          \/
Backspace            \b
Form feed            \f
New line             \n
Carriage return      \r
Horizontal tab       \t
```

```
Control character    Encoded sequence
-----------------------------------
CHAR(0)              \u0000
CHAR(1)              \u0001
...                  ...
CHAR(31)             \u001f
```

Lire Fonctions de chaîne en ligne: https://riptutorial.com/fr/sql-server/topic/4113/fonctions-de-chaine

Chapitre 42: Fonctions de classement

Syntaxe

- DENSE_RANK () OVER ([<partition_by_clause>] <order_by_clause>)
- RANK () OVER ([partition_by_clause] order_by_clause)

Paramètres

Arguments	Détails
`<partition_by_clause>`	Divise le jeu de résultats produit par la clause FROM en partitions auxquelles la fonction `DENSE_RANK` est appliquée. Pour la syntaxe `PARTITION BY` , voir Clause OVER (Transact-SQL) .
`<order_by_clause>`	Détermine l'ordre dans lequel la fonction `DENSE_RANK` est appliquée aux lignes d'une partition.
`OVER ( [ partition_by_clause ] order_by_clause)`	`partition_by_clause` divise le jeu de résultats produit par la clause `FROM` en partitions auxquelles la fonction est appliquée. Si elle n'est pas spécifiée, la fonction traite toutes les lignes du jeu de résultats de la requête en tant que groupe unique. order_by_clause détermine l'ordre des données avant l'application de la fonction. Le order_by_clause est requis. La `<rows or range clause>` de la clause `OVER` ne peut pas être spécifiée pour la fonction `RANK` . Pour plus d'informations, voir Clause OVER (Transact-SQL) .

Remarques

Si deux ou plusieurs lignes sont liées à un rang dans la même partition, chaque ligne liée reçoit le même rang. Par exemple, si les deux principaux vendeurs ont la même valeur SalesYTD, ils sont tous deux classés un. Le vendeur avec le prochain SalesYTD le plus élevé est classé numéro deux. C'est un de plus que le nombre de lignes distinctes avant cette ligne. Par conséquent, les nombres renvoyés par la fonction `DENSE_RANK` n'ont pas de lacunes et ont toujours des rangs consécutifs.

L'ordre de tri utilisé pour l'ensemble de la requête détermine l'ordre dans lequel les lignes apparaissent dans un résultat. Cela implique qu'une ligne classée numéro un ne doit pas nécessairement être la première ligne de la partition.

`DENSE_RANK` n'est pas déterministe. Pour plus d'informations, voir Fonctions déterministes et non déterministes .

Examples

RANG()

A RANK () Retourne le rang de chaque ligne dans le jeu de résultats de la colonne partitionnée.

Par exemple :

```
Select Studentid,Name,Subject,Marks,
RANK() over(partition by name order by Marks desc)Rank
From Exam
order by name,subject

   Studentid    Name    Subject    Marks    Rank

    101         Ivan    Maths       70       2
    101         Ivan    Science     80       1
    101         Ivan    Social      60       3
    102         Ryan    Maths       60       2
    102         Ryan    Science     50       3
    102         Ryan    Social      70       1
    103         Tanvi   Maths       90       1
    103         Tanvi   Science     90       1
    103         Tanvi   Social      80       3
```

DENSE_RANK ()

Identique à celle de RANK (). Il renvoie le rang sans aucune lacune:

```
Select  Studentid, Name,Subject,Marks,
DENSE_RANK() over(partition by name order by Marks desc)Rank
From Exam
order by name

Studentid    Name    Subject    Marks    Rank
101          Ivan    Science    80       1
101          Ivan    Maths      70       2
101          Ivan    Social     60       3
102          Ryan    Social     70       1
102          Ryan    Maths      60       2
102          Ryan    Science    50       3
103          Tanvi   Maths      90       1
103          Tanvi   Science    90       1
103          Tanvi   Social     80       2
```

Lire Fonctions de classement en ligne: https://riptutorial.com/fr/sql-server/topic/5031/fonctions-de-classement

Chapitre 43: Fonctions de fenêtre

Examples

Moyenne mobile centrée

Calculer une moyenne mobile centrée sur un prix sur 6 mois (126 jours ouvrables):

```
SELECT TradeDate, AVG(Px) OVER (ORDER BY TradeDate ROWS BETWEEN 63 PRECEDING AND 63 FOLLOWING)
AS PxMovingAverage
FROM HistoricalPrices
```

Notez que, parce qu'il faudra *jusqu'à* 63 lignes avant et après chaque ligne renvoyée, au début et à la fin de la plage TradeDate, elle ne sera pas centrée: quand elle atteindra la valeur TradeDate la plus élevée, inclure dans la moyenne.

Recherchez l'élément le plus récent dans une liste d'événements horodatés

Dans les tables enregistrant des événements, il existe souvent un champ datetime qui enregistre l'heure à laquelle un événement s'est produit. Trouver l'événement le plus récent peut être difficile car il est toujours possible que deux événements aient été enregistrés avec des horodatages exactement identiques. Vous pouvez utiliser row_number () (ordre par ...) pour vous assurer que tous les enregistrements sont classés de manière unique, et sélectionnez le premier (où my_ranking = 1)

```
select *
from (
    select
        *,
        row_number() over (order by crdate desc) as my_ranking
    from sys.sysobjects
) g
where my_ranking=1
```

Cette même technique peut être utilisée pour renvoyer une seule ligne de n'importe quel jeu de données avec des valeurs potentiellement dupliquées.

Moyenne mobile des 30 derniers articles

Moyenne mobile des 30 derniers articles vendus

```
SELECT
    value_column1,
    (   SELECT
            AVG(value_column1) AS moving_average
        FROM Table1 T2
        WHERE ( SELECT
                    COUNT(*)
                FROM Table1 T3
```

```
                WHERE date_column1 BETWEEN T2.date_column1 AND T1.date_column1
                ) BETWEEN 1 AND 30
    ) as MovingAvg
FROM Table1 T1
```

Lire Fonctions de fenêtre en ligne: https://riptutorial.com/fr/sql-server/topic/3209/fonctions-de-fenetre

Chapitre 44: Fonctions logiques

Examples

CHOISIR

SQL Server 2012

Renvoie l'élément à l'index spécifié dans une liste de valeurs. Si `index` dépasse les limites des `values` `NULL` est renvoyé.

Paramètres:

1. `index` : integer, index à l'élément dans les `values` . 1 base
2. `values` : tout type, liste séparée par des virgules

```
SELECT CHOOSE (1, 'apples', 'pears', 'oranges', 'bananas') AS chosen_result

chosen_result
-------------
apples
```

IIF

SQL Server 2012

Retourne l'une des deux valeurs, selon qu'une expression booléenne donnée est vraie ou fausse.

Paramètres:

1. `boolean_expression` évaluée à dtermine quelle valeur à retourner
2. `true_value` est retourné si `boolean_expression` évalué à true
3. `false_value` retourné si `boolean_expression` évalué à false

```
SELECT IIF (42 > 23, 'I knew that!', 'That is not true.') AS iif_result

iif_result
------------
I knew that!
```

SQL Server 2012

`IIF` peut être remplacé par une déclaration `CASE` . L'exemple ci-dessus peut être écrit comme

```
SELECT CASE WHEN 42 > 23 THEN 'I knew that!' ELSE 'That is not true.' END AS iif_result

iif_result
------------
I knew that!
```

Lire Fonctions logiques en ligne: https://riptutorial.com/fr/sql-server/topic/10647/fonctions-logiques

Chapitre 45: FUSIONNER

Introduction

À partir de SQL Server 2008, il est possible d'effectuer des opérations d'insertion, de mise à jour ou de suppression dans une seule instruction à l'aide de l'instruction MERGE.

L'instruction MERGE vous permet de joindre une source de données avec une table ou une vue cible, puis d'effectuer plusieurs actions sur la cible en fonction des résultats de cette jointure.

Syntaxe

- Selon MSDN - https://msdn.microsoft.com/en-us/library/bb510625.aspx [AVEC <common_table_expression> [, ... n]] MERGE [TOP (expression) [POURCENT]] [INTO] < target_table> [WITH (<merge_hint>)] [[AS] table_alias] UTILISATION <table_source> ON <merge_search_condition> [QUAND CORRIGÉ [ET <clause_search_condition>] THEN <merge_matched>] [... n] CIBLE] [ET <clause_search_condition>] THEN <merge_not_matched>] [QUAND N'EST PAS CORRESPONDANT PAR SOURCE [ET <clause_search_condition>] ALORS <merge_matched>] [... n] [<output_clause>] [OPTION (<query_hint> [,. ..n])]; <target_table> :: = {[nom_bdd. nom_schéma | nom_schéma] target_table} <merge_hint> :: = {{[<table_hint_limited> [, ... n]] [[,] INDEX (index_val [, ... n])]}} <table_source> :: = {table_ou_nom_view [[AS] table_alias] [<tableample_clause>] [WITH (table_hint [[,] ... n])] | rowset_function [[AS] table_alias] [(bulk_column_alias [, ... n])] | user_defined_function [[AS] table_alias] | OPENXML <openxml_clause> | dérivée_table [AS] alias_table [(alias_colonne [, ... n])] | <rejoindre_table> | <pivoted_table> | <tablepivoted_table>} <merge_search_condition> :: = <search_condition> <merge_matched> :: = {UPDATE SET <set_clause> | DELETE} <set_clause> :: = SET {nom_colonne = {expression | DÉFAUT | NULL} | {udt_column_name. {{property_name = expression | field_name = expression} | method_name (argument [, ... n])}} | nom_colonne {.WRITE (expression, @Offset, @Length)} | @variable = expression | @variable = colonne = expression | nom_colonne {+ = | - = | * = | / = | % = | & = | ^ = | | =} expression | @ variable {+ = | - = | * = | / = | % = | & = | ^ = | | =} expression | @variable = colonne {+ = | - = | * = | / = | % = | & = | ^ = | | =} expression} [, ... n] <merge_not_matched> :: = {INSERT [(column_list)] {VALUES (values_list) | VALEURS PAR DÉFAUT}} <clause_search_condition> :: = <search_condition> :: = {[NOT] | (<condition_recherche>)} [{ET | OU} [NOT] {| (<search_condition>)}] [, ... n] :: = {expression {= | <> | ! = |

  | > = | ! > | <| <= | ! <} expression | expression_chaîne [NOT] LIKE expression_chaîne [ESCAPE 'escape_character'] | expression [NOT] BETWEEN expression AND expression | expression IS [NOT] NULL | CONTAINS ({column | *}, '<contains_search_condition>') | FREETEXT ({colonne | *}, 'freetext_string') | expression [NOT] IN (sous-requête | expression [, ... n]) | expression {= | <> | ! = | | > = | ! > | <| <= | ! <} {TOUS | QUEL | ANY} (sous-requête) | EXISTS (sous-requête)} <output_clause> :: = {[OUTPUT <dml_select_list> INTO {@table_variable | output_table} [(column_list)]] [OUTPUT <dml_select_list>]}

<dml_select_list> :: = {<nom_colonne> | expression_ scalaire} [[AS] identifiant_alias_colonne] [, ... n] <nom_colonne> :: = {SUPPRIMÉ | INSÉRÉ | from_table_name}. {* | nom_colonne} | $ action

Remarques

Effectue des opérations d'insertion, de mise à jour ou de suppression sur une table cible en fonction des résultats d'une jointure avec une table source. Par exemple, vous pouvez synchroniser deux tables en insérant, en mettant à jour ou en supprimant des lignes dans une table en fonction des différences trouvées dans l'autre table.

Examples

MERGE to Insert / Update / Delete

```
MERGE INTO targetTable

USING sourceTable
ON (targetTable.PKID = sourceTable.PKID)

WHEN MATCHED AND (targetTable.PKID > 100) THEN
    DELETE

WHEN MATCHED AND (targetTable.PKID <= 100) THEN
    UPDATE SET
        targetTable.ColumnA = sourceTable.ColumnA,
        targetTable.ColumnB = sourceTable.ColumnB

WHEN NOT MATCHED THEN
    INSERT (ColumnA, ColumnB) VALUES (sourceTable.ColumnA, sourceTable.ColumnB);

WHEN NOT MATCHED BY SOURCE THEN
    DELETE
; --< Required
```

La description:

- `MERGE INTO targetTable` - table à modifier
- `USING sourceTable` - source de données (peut être une table, une vue ou une fonction de table)
- `ON ...` - joint la condition entre `targetTable` et `sourceTable` .
- `WHEN MATCHED` - actions à entreprendre lorsqu'une correspondance est trouvée
- - `AND (targetTable.PKID > 100)` - condition (s) supplémentaire (s) à satisfaire pour que l'action soit effectuée
- `THEN DELETE` - supprime l'enregistrement correspondant de la `targetTable`
- `THEN UPDATE` - met à jour les colonnes de l'enregistrement correspondant spécifié par `SET ...`.
- `WHEN NOT MATCHED` - actions à entreprendre lorsque la correspondance est introuvable dans **`targetTable`**
- `WHEN NOT MATCHED BY SOURCE` - actions à entreprendre lorsque la correspondance est introuvable dans **`sourceTable`**

Commentaires:

Si une action spécifique n'est pas nécessaire, omettez la condition, par exemple en supprimant `WHEN NOT MATCHED THEN INSERT` empêchera l'insertion des enregistrements.

L'instruction de fusion nécessite un point-virgule de terminaison.

Restrictions:

- `WHEN MATCHED` n'autorise pas l'action `INSERT`
- `UPDATE` action `UPDATE` ne peut mettre à jour une ligne qu'une seule fois. Cela implique que la condition de jointure doit produire des correspondances uniques.

Fusion à l'aide de la source CTE

```
WITH SourceTableCTE AS
(
    SELECT * FROM SourceTable
)
MERGE
 TargetTable AS target
USING SourceTableCTE AS source
ON (target.PKID = source.PKID)
WHEN MATCHED THEN
    UPDATE SET target.ColumnA = source.ColumnA
WHEN NOT MATCHED THEN
    INSERT (ColumnA) VALUES (Source.ColumnA);
```

FUSIONNER en utilisant la table source dérivée

```
MERGE INTO TargetTable  AS Target
USING (VALUES (1,'Value1'), (2, 'Value2'), (3,'Value3'))
       AS Source (PKID, ColumnA)
ON Target.PKID = Source.PKID
WHEN MATCHED THEN
    UPDATE SET target.ColumnA= source.ColumnA
WHEN NOT MATCHED THEN
    INSERT (PKID, ColumnA) VALUES (Source.PKID, Source.ColumnA);
```

Exemple de fusion - Synchroniser la table source et cible

Pour illustrer l'instruction MERGE, considérez les deux tableaux suivants:

1. **dbo.Product** : Ce tableau contient des informations sur le produit vendu par cette société
2. **dbo.ProductNew** : Ce tableau contient des informations sur le produit que la société vendra ultérieurement.

Le T-SQL suivant créera et remplira ces deux tables

```
IF OBJECT_id(N'dbo.Product',N'U') IS NOT NULL
DROP TABLE dbo.Product
```

```
GO

CREATE TABLE dbo.Product (
ProductID INT PRIMARY KEY,
ProductName NVARCHAR(64),
PRICE MONEY
)

IF OBJECT_id(N'dbo.ProductNew',N'U') IS NOT NULL
DROP TABLE dbo.ProductNew
GO

CREATE TABLE dbo.ProductNew (
ProductID INT PRIMARY KEY,
ProductName NVARCHAR(64),
PRICE MONEY
)

INSERT INTO dbo.Product VALUES(1,'IPod',300)
,(2,'IPhone',400)
,(3,'ChromeCast',100)
,(4,'raspberry pi',50)

INSERT INTO dbo.ProductNew VALUES(1,'Asus Notebook',300)
,(2,'Hp Notebook',400)
,(3,'Dell Notebook',100)
,(4,'raspberry pi',50)
```

Maintenant, supposons que nous voulions synchroniser la table cible dbo.Product avec la table dbo.ProductNew. Voici le critère pour cette tâche:

1. Les produits existant dans la table source dbo.ProductNew et la table cible dbo.Product sont mis à jour dans la table cible dbo.Product avec les nouveaux nouveaux produits.

2. Tout produit de la table source dbo.ProductNew qui n'existe pas dans la table cible dob.Product est inséré dans la table cible dbo.Product.

3. Tout produit de la table cible dbo.Product qui n'existe pas dans la table source dbo.ProductNew doit être supprimé de la table cible dbo.Product. Voici l'instruction MERGE pour effectuer cette tâche.

```
MERGE dbo.Product AS SourceTbl
USING dbo.ProductNew AS TargetTbl ON (SourceTbl.ProductID = TargetTbl.ProductID)
WHEN MATCHED
            AND SourceTbl.ProductName <> TargetTbl.ProductName
            OR SourceTbl.Price <> TargetTbl.Price
    THEN UPDATE SET SourceTbl.ProductName = TargetTbl.ProductName,
                SourceTbl.Price = TargetTbl.Price
WHEN NOT MATCHED
    THEN INSERT (ProductID, ProductName, Price)
         VALUES (TargetTbl.ProductID, TargetTbl.ProductName, TargetTbl.Price)
WHEN NOT MATCHED BY SOURCE
    THEN DELETE
OUTPUT $action, INSERTED.*, DELETED.*;
```

Remarque: Le point-virgule doit être présent à la fin de l'instruction MERGE.

	$action	ProductID	ProductName	PRICE	ProductID	ProductName	PRICE
1	UPDATE	1	Asus Notebook	300.00	1	IPod	300.00
2	UPDATE	2	Hp Notebook	400.00	2	IPhone	400.00
3	UPDATE	3	Dell Notebook	100.00	3	ChromeCast	100.00

Fusionner en utilisant sauf

Utilisez EXCEPT pour empêcher les mises à jour des enregistrements inchangés

```
MERGE TargetTable targ
USING SourceTable AS src
    ON src.id = targ.id
WHEN MATCHED
    AND EXISTS (
        SELECT src.field
        EXCEPT
        SELECT targ.field
        )
    THEN
        UPDATE
        SET field = src.field
WHEN NOT MATCHED BY TARGET
    THEN
        INSERT (
            id
            ,field
            )
        VALUES (
            src.id
            ,src.field
            )
WHEN NOT MATCHED BY SOURCE
    THEN
        DELETE;
```

Lire FUSIONNER en ligne: https://riptutorial.com/fr/sql-server/topic/4550/fusionner

Chapitre 46: Générer une plage de dates

Paramètres

Paramètre	Détails
@Partir de la date	La limite inférieure inclusive de la plage de dates générée.
@À ce jour	La limite supérieure inclusive de la plage de dates générée.

Remarques

La plupart des experts semblent recommander de créer une table Dates au lieu de générer une séquence à la volée. Voir http://dba.stackexchange.com/questions/86435/filling-in-date-holes-in-grouped-by-date-sql-data

Examples

Génération de la plage de dates avec le CTE récursif

En utilisant un CTE récursif, vous pouvez générer une plage de dates incluse:

```
Declare  @FromDate    Date = '2014-04-21',
         @ToDate      Date = '2014-05-02'

;With DateCte (Date) As
(
    Select  @FromDate Union All
    Select  DateAdd(Day, 1, Date)
    From    DateCte
    Where   Date < @ToDate
)
Select  Date
From    DateCte
Option  (MaxRecursion 0)
```

Le paramètre `MaxRecursion` par défaut est 100. La génération de plus de 100 dates à l'aide de cette méthode nécessite le segment `Option (MaxRecursion N)` de la requête, où `N` correspond au paramètre `MaxRecursion` souhaité. `MaxRecursion` cette valeur sur `0` , la limitation `MaxRecursion` sera entièrement `MaxRecursion` .

Génération d'une plage de dates avec une table de contrôle

Une autre façon de générer une plage de dates consiste à utiliser une table de correspondance pour créer les dates entre les plages:

```
Declare   @FromDate   Date = '2014-04-21',
          @ToDate     Date = '2014-05-02'

;With
   E1(N) As (Select 1 From (Values (1), (1), (1), (1), (1), (1), (1), (1), (1), (1)) DT(N)),
   E2(N) As (Select 1 From E1 A Cross Join E1 B),
   E4(N) As (Select 1 From E2 A Cross Join E2 B),
   E6(N) As (Select 1 From E4 A Cross Join E2 B),
   Tally(N) As
   (
        Select    Row_Number() Over (Order By (Select Null))
        From    E6
   )
Select   DateAdd(Day, N - 1, @FromDate) Date
From     Tally
Where    N <= DateDiff(Day, @FromDate, @ToDate) + 1
```

Lire Générer une plage de dates en ligne: https://riptutorial.com/fr/sql-server/topic/3232/generer-une-plage-de-dates

Chapitre 47: Gestion de la base de données SQL Azure

Examples

Rechercher des informations de niveau de service pour la base de données SQL Azure

Azure SQL Database a différentes éditions et niveaux de performance.

Vous pouvez trouver la version, l'édition (de base, standard ou premium) et l'objectif de service (S0, S1, P4, P11, etc.) de la base de données SQL exécutée en tant que service dans Azure à l'aide des instructions suivantes:

```
select @@version
SELECT DATABASEPROPERTYEX('Wwi', 'EDITION')
SELECT DATABASEPROPERTYEX('Wwi', 'ServiceObjective')
```

Niveau de service de modification de la base de données SQL Azure

Vous pouvez augmenter ou réduire la base de données SQL Azure à l'aide de l'instruction ALTER DATABASE:

```
ALTER DATABASE WWI
MODIFY (SERVICE_OBJECTIVE = 'P6')
-- or
ALTER DATABASE CURRENT
MODIFY (SERVICE_OBJECTIVE = 'P2')
```

Si vous essayez de modifier le niveau de service pendant que vous modifiez le niveau de service de la base de données en cours, vous obtenez l'erreur suivante:

> Msg 40802, niveau 16, état 1, ligne 1 Une affectation d'objectif de service sur le serveur '.....' et la base de données '.....' est déjà en cours. Veuillez patienter jusqu'à ce que l'état d'affectation de l'objectif de service pour la base de données soit marqué comme "Terminé".

Réexécutez votre instruction ALTER DATABASE lorsque la période de transition se termine.

Réplication de la base de données SQL Azure

Vous pouvez créer un réplica secondaire de base de données portant le même nom sur un autre serveur Azure SQL Server, en rendant la base de données locale principale et en répliquant de manière asynchrone les données primaires vers les nouvelles secondaires.

```
ALTER DATABASE <<mydb>>
ADD SECONDARY ON SERVER <<secondaryserver>>
WITH ( ALLOW_CONNECTIONS = ALL )
```

Le serveur cible peut se trouver dans un autre centre de données (utilisable pour la géoréplication). Si une base de données portant le même nom existe déjà sur le serveur cible, la commande échouera. La commande est exécutée sur la base de données principale du serveur hébergeant la base de données locale qui deviendra la base de données principale. Lorsque ALLOW_CONNECTIONS est défini sur ALL (défini sur NO par défaut), le réplica secondaire sera une base de données en lecture seule qui autorisera toutes les connexions avec les autorisations appropriées pour la connexion.

La réplique de base de données secondaire peut être promue vers le serveur principal à l'aide de la commande suivante:

```
ALTER DATABASE mydb FAILOVER
```

Vous pouvez supprimer la base de données secondaire sur le serveur secondaire:

```
ALTER DATABASE <<mydb>>
REMOVE SECONDARY ON SERVER <<testsecondaryserver>>
```

Créer une base de données SQL Azure dans un pool élastique

Vous pouvez mettre votre base de données SQL Azure dans un pool élastique SQL:

```
CREATE DATABASE wwi
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

Vous pouvez créer une copie d'une base de données existante et la placer dans un pool élastique:

```
CREATE DATABASE wwi
AS COPY OF myserver.WideWorldImporters
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

Lire Gestion de la base de données SQL Azure en ligne: https://riptutorial.com/fr/sql-server/topic/7113/gestion-de-la-base-de-donnees-sql-azure

Chapitre 48: Gouverneur des ressources

Remarques

Le gouverneur de ressources dans SQL Server est une fonctionnalité qui vous permet de gérer l'utilisation des ressources par différentes applications et utilisateurs. Il démarre en temps réel en définissant les limites de CPU et de mémoire. Cela évitera qu'un processus lourd consomme toutes les ressources du système alors que, par exemple, des tâches plus petites les attendent.

Disponible uniquement dans les éditions Enterprise

Examples

Lecture des statistiques

```
select *
from sys.dm_resource_governor_workload_groups

select *
from sys.dm_resource_governor_resource_pools
```

Créer un pool pour les requêtes ad hoc

Créez d'abord un pool de ressources en plus de celui par défaut

```
CREATE RESOURCE POOL [PoolAdhoc] WITH(min_cpu_percent=0,
        max_cpu_percent=50,
        min_memory_percent=0,
        max_memory_percent=50)
GO
```

Créez le groupe de charge pour le pool

```
CREATE WORKLOAD GROUP [AdhocMedium] WITH(importance=Medium) USING [PoolAdhoc]
```

Créez la fonction qui contient la logique du gouverneur de ressources et attachez-la

```
create function [dbo].[ufn_ResourceGovernorClassifier]()
  returns sysname with schemabinding
as
begin
    return CASE
                WHEN APP_NAME() LIKE 'Microsoft Office%'                        THEN
'AdhocMedium'         -- Excel
                WHEN APP_NAME() LIKE 'Microsoft SQL Server Management Studio%'    THEN
'AdhocMedium'         -- Adhoc SQL
                WHEN SUSER_NAME() LIKE 'DOMAIN\username'                    THEN 'AdhocMedium'
-- Ssis
                ELSE 'default'
```

```
            END
end

GO

alter resource governor
with (classifier_function = dbo.ufn_ResourceGovernorClassifier)

GO

alter resource governor reconfigure

GO
```

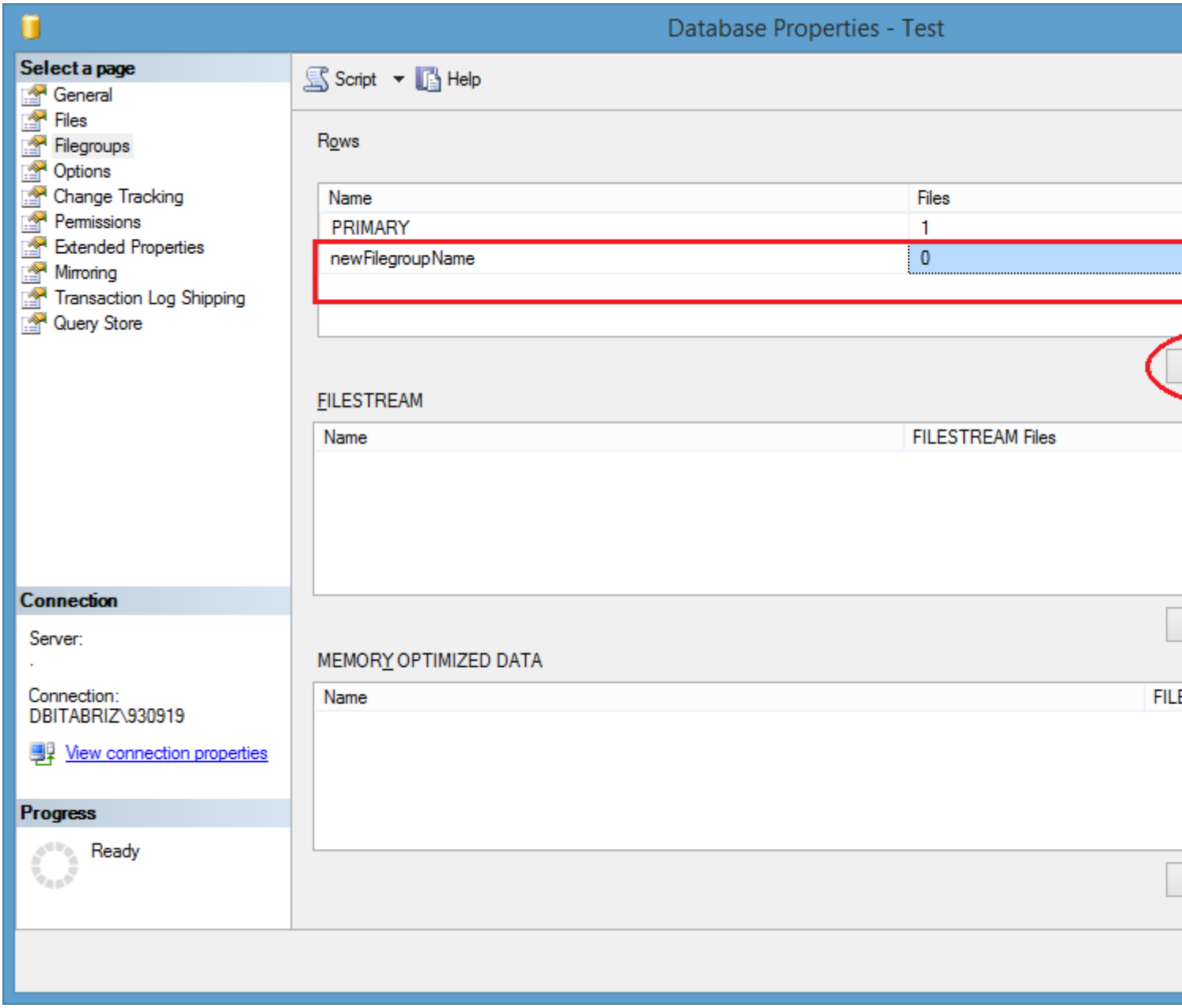
Lire Gouverneur des ressources en ligne: https://riptutorial.com/fr/sql-server/topic/4146/gouverneur-des-ressources

Chapitre 49: Groupe de fichiers

Examples

Créer un groupe de fichiers dans la base de données

Nous pouvons le créer par deux voies. Tout d'abord à partir du mode de concepteur de propriétés de base de données:



Et par scripts SQL:

```
USE master;
GO
-- Create the database with the default data
-- filegroup and a log file. Specify the
```

```
-- growth increment and the max size for the
-- primary data file.

CREATE DATABASE TestDB ON PRIMARY
(
    NAME = 'TestDB_Primary',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_Prm.mdf',
    SIZE = 1 GB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
), FILEGROUP TestDB_FG1
(
    NAME = 'TestDB_FG1_1',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_FG1_1.ndf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
),
(
    NAME = 'TestDB_FG1_2',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_FG1_2.ndf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
) LOG ON
(
    NAME = 'TestDB_log',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB.ldf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
);

go
ALTER DATABASE TestDB MODIFY FILEGROUP TestDB_FG1 DEFAULT;
go

-- Create a table in the user-defined filegroup.
USE TestDB;
Go

CREATE TABLE MyTable
(
    col1 INT PRIMARY KEY,
    col2 CHAR(8)
)
ON TestDB_FG1;
GO
```

Lire Groupe de fichiers en ligne: https://riptutorial.com/fr/sql-server/topic/5461/groupe-de-fichiers

Chapitre 50: Indexation de texte intégral

Examples

A. Création d'un index unique, d'un catalogue de texte intégral et d'un index de texte intégral

L'exemple suivant crée un index unique sur la colonne JobCandidateID de la table HumanResources.JobCandidate de la base de données exemple AdventureWorks2012. L'exemple crée ensuite un catalogue de texte intégral par défaut, ft. Enfin, l'exemple crée un index de texte intégral dans la colonne Resume, à l'aide du catalogue ft et de la liste d'arrêt du système.

```
USE AdventureWorks2012;
GO
CREATE UNIQUE INDEX ui_ukJobCand ON HumanResources.JobCandidate(JobCandidateID);
CREATE FULLTEXT CATALOG ft AS DEFAULT;
CREATE FULLTEXT INDEX ON HumanResources.JobCandidate(Resume)
   KEY INDEX ui_ukJobCand
   WITH STOPLIST = SYSTEM;
GO
```

https://www.simple-talk.com/sql/learn-sql-server/understanding-full-text-indexing-in-sql-server/

https://msdn.microsoft.com/en-us/library/cc879306.aspx

https://msdn.microsoft.com/en-us/library/ms142571.aspx

Création d'un index de texte intégral sur plusieurs colonnes de la table

```
USE AdventureWorks2012;
GO
CREATE FULLTEXT CATALOG production_catalog;
GO
CREATE FULLTEXT INDEX ON Production.ProductReview
 (
  ReviewerName
     Language 1033,
  EmailAddress
     Language 1033,
  Comments
     Language 1033
 )
  KEY INDEX PK_ProductReview_ProductReviewID
      ON production_catalog;
GO
```

Créer un index de texte intégral avec une liste de propriétés de recherche sans le remplir

```
USE AdventureWorks2012;
GO
CREATE FULLTEXT INDEX ON Production.Document
  (
  Title
      Language 1033,
  DocumentSummary
      Language 1033,
  Document
      TYPE COLUMN FileExtension
      Language 1033
  )
  KEY INDEX PK_Document_DocumentID
         WITH STOPLIST = SYSTEM, SEARCH PROPERTY LIST = DocumentPropertyList, CHANGE_TRACKING
OFF, NO POPULATION;
   GO
```

Et le peupler plus tard avec

```
ALTER FULLTEXT INDEX ON Production.Document SET CHANGE_TRACKING AUTO;
GO
```

Recherche en texte intégral

```
SELECT product_id
FROM products
WHERE CONTAINS(product_description, "Snap Happy 100EZ" OR FORMSOF(THESAURUS,'Snap Happy') OR
'100EZ')
AND product_cost < 200 ;


SELECT candidate_name,SSN
FROM candidates
WHERE CONTAINS(candidate_resume,"SQL Server") AND candidate_division =DBA;
```

Pour plus d'informations détaillées et https://msdn.microsoft.com/en-us/library/ms142571.aspx

Lire Indexation de texte intégral en ligne: https://riptutorial.com/fr/sql-server/topic/4557/indexation-de-texte-integral

Chapitre 51: Indice

Examples

Créer un index clusterisé

Avec un index clusterisé, les pages feuille contiennent les lignes de table réelles. Par conséquent, il ne peut y avoir qu'un seul index clusterisé.

```
CREATE TABLE Employees
(
    ID CHAR(900),
    FirstName NVARCHAR(3000),
    LastName NVARCHAR(3000),
    StartYear CHAR(900)
)
GO

CREATE CLUSTERED INDEX IX_Clustered
ON Employees(ID)
GO
```

Créer un index non clusterisé

Les index non clusterisés ont une structure distincte des lignes de données. Un index non clusterisé contient les valeurs de clé d'index non mises en cluster et chaque entrée de valeur de clé comporte un pointeur sur la ligne de données contenant la valeur de clé. Il peut y avoir un maximum de 999 index non clusterisés sur SQL Server 2008/2012.

Lien pour référence: https://msdn.microsoft.com/en-us/library/ms143432.aspx

```
CREATE TABLE Employees
(
    ID CHAR(900),
    FirstName NVARCHAR(3000),
    LastName NVARCHAR(3000),
    StartYear CHAR(900)
)
GO

CREATE NONCLUSTERED INDEX IX_NonClustered
ON Employees(StartYear)
GO
```

Afficher les infos d'index

```
SP_HELPINDEX tableName
```

Index sur vue

```
CREATE VIEW  View_Index02
WITH SCHEMABINDING
AS
SELECT c.CompanyName, o.OrderDate, o.OrderID, od.ProductID
     FROM dbo.Customers C
        INNER JOIN dbo.orders O ON c.CustomerID=o.CustomerID
            INNER JOIN dbo.[Order Details] od ON o.OrderID=od.OrderID
GO

CREATE UNIQUE CLUSTERED INDEX IX1 ON
    View_Index02(OrderID, ProductID)
```

Index de baisse

```
DROP INDEX IX_NonClustered ON Employees
```

Retourne la taille et les index de fragmentation

```
sys.dm_db_index_physical_stats (
    { database_id | NULL | 0 | DEFAULT }
  , { object_id | NULL | 0 | DEFAULT }
  , { index_id | NULL | 0 | -1 | DEFAULT }
  , { partition_number | NULL | 0 | DEFAULT }
  , { mode | NULL | DEFAULT }
)


Sample :

SELECT * FROM sys.dm_db_index_physical_stats
    (DB_ID(N'DBName'), OBJECT_ID(N'IX_NonClustered '), NULL, NULL , 'DETAILED');
```

Réorganiser et reconstruire l'index

valeur avg_fragmentation_in_percent	Déclaration corrective
> 5% et <= 30%	RÉORGANISER
> 30%	RECONSTRUIRE

```
ALTER INDEX IX_NonClustered ON tableName REORGANIZE;


ALTER INDEX ALL ON Production.Product
 REBUILD WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON,
          STATISTICS_NORECOMPUTE = ON);
```

Reconstruire ou réorganiser tous les index sur une table

La reconstruction des index se fait à l'aide de l'instruction suivante

```
ALTER INDEX All ON tableName REBUILD;
```

Cela supprime l'index et le recrée, en supprimant les fractions, récupère l'espace disque et réorganise les pages d'index.

On peut aussi réorganiser un index en utilisant

```
ALTER INDEX All ON tableName REORGANIZE;
```

qui utilisera des ressources système minimales et défragmentera le niveau feuille des index clusterisés et non clusterisés sur les tables et les vues en réorganisant physiquement les pages de niveau feuille afin qu'elles correspondent à l'ordre logique de gauche à droite des nœuds feuilles

Reconstruire toute la base de données d'index

```
EXEC sp_MSForEachTable 'ALTER INDEX ALL ON ? REBUILD'
```

Enquêtes d'index

Vous pouvez utiliser "SP_HELPINDEX Table_Name", mais Kimberly Tripp a une procédure stockée (qui peut être trouvée ici), qui est un meilleur exemple, car elle montre plus sur les index, y compris les colonnes et la définition de filtre, par exemple:
Usage:

```
USE Adventureworks
EXEC sp_SQLskills_SQL2012_helpindex 'dbo.Product'
```

Tibor Karaszi a aussi une procédure stockée (trouvée ici). Le dernier affichera également des informations sur l'utilisation des index et fournira éventuellement une liste de suggestions d'index.
Usage:

```
USE Adventureworks
EXEC sp_indexinfo 'dbo.Product'
```

Lire Indice en ligne: https://riptutorial.com/fr/sql-server/topic/4998/indice

Chapitre 52: Insérer

Examples

Ajouter une ligne à une table nommée factures

```
INSERT INTO Invoices [ /* column names may go here */ ]
VALUES (123, '1234abc', '2016-08-05 20:18:25.770', 321, 5, '2016-08-04');
```

- Les noms de colonne sont obligatoires si la table dans laquelle vous insérez contient une colonne avec l'attribut IDENTITY.

```
INSERT INTO Invoices ([ID], [Num], [DateTime], [Total], [Term], [DueDate])
VALUES (123, '1234abc', '2016-08-05 20:18:25.770', 321, 5, '2016-08-25');
```

Lire Insérer en ligne: https://riptutorial.com/fr/sql-server/topic/5323/inserer

Chapitre 53: INSÉRER DANS

Introduction

L'instruction INSERT INTO est utilisée pour insérer de nouveaux enregistrements dans une table.

Examples

INSERT Hello World INTO table

```
CREATE TABLE MyTableName
(
    Id INT,
    MyColumnName NVARCHAR(1000)
)
GO

INSERT INTO MyTableName (Id, MyColumnName)
VALUES (1, N'Hello World!')
GO
```

INSERT sur des colonnes spécifiques

Pour effectuer une insertion sur des colonnes spécifiques (par opposition à toutes), vous devez spécifier les colonnes que vous souhaitez mettre à jour.

```
INSERT INTO USERS (FIRST_NAME, LAST_NAME)
VALUES ('Stephen', 'Jiang');
```

Cela ne fonctionnera que si les colonnes que vous n'avez pas listées sont nullables, identité, type de données d'horodatage ou colonnes calculées; ou des colonnes qui ont une contrainte de valeur par défaut. Par conséquent, si l'une de ces colonnes est non nullable, sans identité, sans horodatage, non calculée, sans valeur par défaut ... alors la tentative de ce type d'insert déclenche un message d'erreur indiquant que vous devez fournir un valeur pour le ou les champs applicables.

INSÉRER plusieurs lignes de données

Pour insérer plusieurs lignes de données dans SQL Server 2008 ou version ultérieure:

```
INSERT INTO USERS VALUES
(2, 'Michael', 'Blythe'),
(3, 'Linda', 'Mitchell'),
(4, 'Jillian', 'Carson'),
(5, 'Garrett', 'Vargas');
```

Pour insérer plusieurs lignes de données dans les versions antérieures de SQL Server, utilisez

"UNION ALL" comme suit:

```
INSERT INTO USERS (FIRST_NAME, LAST_NAME)
SELECT 'James', 'Bond' UNION ALL
SELECT 'Miss', 'Moneypenny' UNION ALL
SELECT 'Raoul', 'Silva'
```

Notez que le mot clé "INTO" est facultatif dans les requêtes INSERT. Un autre avertissement est que le serveur SQL ne prend en charge que 1000 lignes dans un INSERT, vous devez donc les diviser en lots.

INSÉRER une seule ligne de données

Une seule ligne de données peut être insérée de deux manières:

```
INSERT INTO USERS(Id, FirstName, LastName)
VALUES (1, 'Mike', 'Jones');
```

Ou

```
INSERT INTO USERS
VALUES (1, 'Mike', 'Jones');
```

Notez que la deuxième instruction d'insertion n'autorise les valeurs que dans le même ordre que les colonnes de la table, alors que dans la première insertion, l'ordre des valeurs peut être modifié comme suit:

```
INSERT INTO USERS(FirstName, LastName, Id)
VALUES ('Mike', 'Jones', 1);
```

Utilisez OUTPUT pour obtenir le nouvel identifiant

Lorsque INSERTing, vous pouvez utiliser `OUTPUT INSERTED.ColumnName` pour obtenir des valeurs à partir de la nouvelle ligne insérée, par exemple le nouvel ID généré, utile si vous avez une colonne `IDENTITY` ou toute sorte de valeur par défaut ou calculée.

Lorsque vous appelez ceci par programmation (par exemple, depuis ADO.net), vous le traitez comme une requête normale et lisez les valeurs comme si vous aviez fait une `SELECT` .

```
-- CREATE TABLE OutputTest ([Id] INT NOT NULL PRIMARY KEY IDENTITY, [Name] NVARCHAR(50))

INSERT INTO OutputTest ([Name])
OUTPUT INSERTED.[Id]
VALUES ('Testing')
```

Si l'ID de la ligne récemment ajoutée est requis dans le même ensemble de requêtes ou de procédures stockées.

```
-- CREATE a table variable having column with the same datatype of the ID
```

```
DECLARE @LastId TABLE ( id int);

INSERT INTO OutputTest ([Name])
OUTPUT INSERTED.[Id] INTO @LastId
VALUES ('Testing')

SELECT id FROM @LastId

-- We can set the value in a variable and use later in procedure

DECLARE @LatestId int = (SELECT id FROM @LastId)
```

INSERT from SELECT Résultats de la requête

Pour insérer des données extraites d'une requête SQL (une ou plusieurs lignes)

```
INSERT INTO Table_name (FirstName, LastName, Position)
SELECT FirstName, LastName, 'student' FROM Another_table_name
```

Notez que «étudiant» dans SELECT est une constante de chaîne qui sera insérée dans chaque ligne.

Si nécessaire, vous pouvez sélectionner et insérer des données depuis / dans la même table

Lire INSÉRER DANS en ligne: https://riptutorial.com/fr/sql-server/topic/3814/inserer-dans

Chapitre 54: Installation de SQL Server sur Windows

Examples

introduction

Voici les éditions disponibles de SQL Server, comme indiqué par Editions Matrix :

- Express: base de données gratuite d'entrée de gamme. Inclut la fonctionnalité de base du RDBMS. Limité à 10 G de la taille du disque. Idéal pour le développement et les tests.
- Standard Edition: édition standard sous licence. Comprend des fonctionnalités de base et des fonctionnalités de Business Intelligence.
- Enterprise Edition: édition complète de SQL Server. Comprend des fonctionnalités avancées de sécurité et d'entreposage de données.
- Developer Edition: comprend toutes les fonctionnalités de Enterprise Edition et aucune limitation, et il est gratuit à télécharger et à utiliser à des fins de développement uniquement.

Après avoir téléchargé / acquis SQL Server, l'installation est exécutée avec SQLSetup.exe, disponible sous forme d'interface graphique ou de programme de ligne de commande.

L'installation via l'un de ces éléments nécessite que vous spécifiiez une clé de produit et que vous exécutiez une configuration initiale qui inclut l'activation des fonctionnalités, la séparation des services et la définition des paramètres initiaux pour chacun d'eux. Des services et fonctionnalités supplémentaires peuvent être activés à tout moment en exécutant le programme SQLSetup.exe dans la ligne de commande ou la version GUI.

Lire Installation de SQL Server sur Windows en ligne: https://riptutorial.com/fr/sql-server/topic/5801/installation-de-sql-server-sur-windows

Chapitre 55: Instantanés de base de données

Remarques

Un instantané de base de données est une vue statique en lecture seule d'une base de données SQL Server compatible avec la base de données source au moment de la création de l'instantané.

Un instantané de base de données réside toujours sur la même instance de serveur que sa base de données source. Lorsque la base de données source est mise à jour, la capture instantanée de la base de données est mise à jour.

Un instantané diffère d'une sauvegarde car le processus de création des instantanés est instantané et que l'instantané n'occupe de l'espace que lorsque des modifications de la base de données source sont appliquées. En revanche, une sauvegarde stocke une copie complète des données au moment de la création de la sauvegarde.

En outre, un instantané fournit une copie instantanée de la base de données en lecture seule, tandis qu'une sauvegarde doit être restaurée sur un serveur pour être lisible (et une fois restaurée, vous pouvez également y écrire)

Les instantanés de base de données ne sont disponibles que dans les éditions Enterprise et Developer.

Examples

Créer un instantané de base de données

Un instantané de base de données est une vue statique en lecture seule d'une base de données SQL Server (la base de données source). Il est similaire à la sauvegarde, mais il est disponible comme toute autre base de données afin que le client puisse interroger la base de données des instantanés.

```
CREATE DATABASE MyDatabase_morning -- name of the snapshot
ON (
     NAME=MyDatabase_data, -- logical name of the data file of the source database
     FILENAME='C:\SnapShots\MySnapshot_Data.ss' -- snapshot file;
)
AS SNAPSHOT OF MyDatabase; -- name of source database
```

Vous pouvez également créer un instantané de la base de données avec plusieurs fichiers:

```
CREATE DATABASE MyMultiFileDBSnapshot ON
    (NAME=MyMultiFileDb_ft, FILENAME='C:\SnapShots\MyMultiFileDb_ft.ss'),
    (NAME=MyMultiFileDb_sys, FILENAME='C:\SnapShots\MyMultiFileDb_sys.ss'),
    (NAME=MyMultiFileDb_data, FILENAME='C:\SnapShots\MyMultiFileDb_data.ss'),
    (NAME=MyMultiFileDb_indx, FILENAME='C:\SnapShots\MyMultiFileDb_indx.ss')
AS SNAPSHOT OF MultiFileDb;
```

Restaurer un instantané de base de données

Si les données d'une base de données source sont endommagées ou si des données incorrectes sont écrites dans la base de données, dans certains cas, la restauration de la base de données à partir d'une sauvegarde constitue une alternative à la base de données.

```
RESTORE DATABASE MYDATABASE FROM DATABASE_SNAPSHOT='MyDatabase_morning';
```

> **Avertissement:** Ceci *supprimera toutes les modifications* apportées à la base de données source depuis la prise de l'instantané!

SUPPRIMER Instantané

Vous pouvez supprimer les instantanés existants de la base de données à l'aide de l'instruction DELETE DATABASE:

```
DROP DATABASE Mydatabase_morning
```

Dans cette instruction, vous devez faire référence au nom de l'instantané de la base de données.

Lire Instantanés de base de données en ligne: https://riptutorial.com/fr/sql-server/topic/677/instantanes-de-base-de-donnees

Chapitre 56: Instruction SELECT

Introduction

Dans SQL, les `SELECT` renvoient des ensembles de résultats provenant de collections de données telles que des tables ou des vues. `SELECT` peuvent être utilisées avec diverses autres clauses comme `WHERE` , `GROUP BY` ou `ORDER BY` pour affiner davantage les résultats souhaités.

Examples

SELECT de base de la table

Sélectionnez toutes les colonnes d'une table (table système dans ce cas):

```
SELECT *
FROM sys.objects
```

Ou sélectionnez simplement certaines colonnes spécifiques:

```
SELECT object_id, name, type, create_date
FROM sys.objects
```

Filtrer les lignes à l'aide de la clause WHERE

La clause WHERE filtre uniquement les lignes qui satisfont certaines conditions:

```
SELECT *
FROM sys.objects
WHERE type = 'IT'
```

Trier les résultats en utilisant ORDER BY

La clause ORDER BY trie les lignes du jeu de résultats renvoyé par une colonne ou une expression:

```
SELECT *
FROM sys.objects
ORDER BY create_date
```

Résultat du groupe avec GROUP BY

La clause GROUP BY groupe les lignes par une valeur quelconque:

```
SELECT type, count(*) as c
FROM sys.objects
```

```
GROUP BY type
```

Vous pouvez appliquer une fonction sur chaque groupe (fonction d'agrégat) pour calculer la somme ou le décompte des enregistrements du groupe.

type	c
SQ	3
S	72
IL	16
PK	1
U	5

Filtrer les groupes à l'aide de la clause HAVING

La clause HAVING supprime les groupes qui ne satisfont pas à la condition:

```
SELECT type, count(*) as c
FROM sys.objects
GROUP BY type
HAVING count(*) < 10
```

type	c
SQ	3
PK	1
U	5

Retourner uniquement les N premières lignes

La clause TOP ne renvoie que les N premières lignes du résultat:

```
SELECT TOP 10 *
FROM sys.objects
```

Pagination en utilisant OFFSET FETCH

La clause OFFSET FETCH est une version plus avancée de TOP. Il vous permet d'ignorer les lignes N1 et de prendre les lignes N2 suivantes:

```
SELECT *
FROM sys.objects
```

```
ORDER BY object_id
OFFSET 50 ROWS FETCH NEXT 10 ROWS ONLY
```

Vous pouvez utiliser OFFSET sans chercher pour ignorer les 50 premières lignes:

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 50 ROWS
```

SELECT sans FROM (pas de source de données)

L'instruction SELECT peut être exécutée sans clause FROM:

```
declare @var int = 17;

SELECT @var as c1, @var + 2 as c2, 'third' as c3
```

Dans ce cas, une ligne contenant des valeurs / résultats d’expressions est renvoyée.

Lire Instruction SELECT en ligne: https://riptutorial.com/fr/sql-server/topic/4662/instruction-select

Chapitre 57: Intégration du Common Language Runtime

Examples

Activer le CLR sur la base de données

Les procédures CLR ne sont pas activées par défaut. Vous devez exécuter les requêtes suivantes pour activer CLR:

```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'clr enabled', 1;
GO
RECONFIGURE;
GO
```

De plus, si un module CLR nécessite un accès externe, vous devez définir la propriété TRUSTWORTHY sur ON dans votre base de données:

```
ALTER DATABASE MyDbWithClr SET TRUSTWORTHY ON
```

Ajout de .dll contenant des modules Sql CLR

Les procédures, fonctions, déclencheurs et types écrits en langues .Net sont stockés dans des fichiers .dll. Une fois que vous créez un fichier .dll contenant des procédures CLR, vous devez l'importer dans SQL Server:

```
CREATE ASSEMBLY MyLibrary
FROM 'C:\lib\MyStoredProcedures.dll'
    WITH PERMISSION_SET = EXTERNAL_ACCESS
```

PERMISSION_SET est Safe par défaut, ce qui signifie que le code dans .dll n'a pas besoin d'une autorisation pour accéder aux ressources externes (par exemple, fichiers, sites Web, autres serveurs) et qu'il n'utilisera pas de code natif pouvant accéder à la mémoire.

PERMISSION_SET = EXTERNAL_ACCESS est utilisé pour marquer les assemblys contenant du code qui accéderont aux ressources externes.

Vous pouvez trouver des informations sur les fichiers d'assemblage CLR actuels dans la vue sys.assemblies:

```
SELECT *
FROM sys.assemblies asms
```

```
WHERE is_user_defined = 1
```

Créer une fonction CLR dans SQL Server

Si vous avez créé la fonction .Net, l'avez compilée dans .dll et importée dans SQL Server en tant qu'assemblage, vous pouvez créer une fonction définie par l'utilisateur qui référence la fonction dans cet assemblage:

```
CREATE FUNCTION dbo.TextCompress(@input nvarchar(max))
RETURNS varbinary(max)
AS EXTERNAL NAME MyLibrary.[Name.Space.ClassName].TextCompress
```

Vous devez spécifier le nom de la fonction et de la signature avec des paramètres d'entrée et renvoyer des valeurs correspondant à la fonction .Net. Dans la clause AS EXTERNAL NAME, vous devez spécifier le nom de l'assembly, le nom de l'espace de nom / classe où cette fonction est placée et le nom de la méthode dans la classe qui contient le code qui sera exposé en tant que fonction.

Vous pouvez trouver des informations sur les fonctions CLR à l'aide de la requête suivante:

```
SELECT * FROM dbo.sysobjects WHERE TYPE ='FS'
```

Créer un type CLR défini par l'utilisateur dans SQL Server

Si vous avez créé une classe .Net qui représente un type défini par l'utilisateur, qui l'a compilé dans .dll et l'a importé dans un serveur SQL en tant qu'assemblage, vous pouvez créer une fonction définie par l'utilisateur qui référence cette classe:

```
CREATE TYPE dbo.Point
EXTERNAL NAME MyLibrary.[Name.Space.Point]
```

Vous devez spécifier le nom du type qui sera utilisé dans les requêtes T-SQL. Dans la clause EXTERNAL NAME, vous devez spécifier le nom de l'assembly, l'espace de noms et le nom de la classe.

Créer une procédure CLR dans SQL Server

Si vous avez créé une méthode .Net dans une classe, l'avez compilée dans .dll et importée dans SQL Server en tant qu'assemblage, vous pouvez créer une procédure stockée définie par l'utilisateur qui fait référence à la méthode dans cet assembly:

```
CREATE PROCEDURE dbo.DoSomethng(@input nvarchar(max))
AS EXTERNAL NAME MyLibrary.[Name.Space.ClassName].DoSomething
```

Vous devez spécifier le nom de la procédure et la signature avec les paramètres d'entrée correspondant à la méthode .Net. Dans la clause AS EXTERNAL NAME, vous devez spécifier le nom de l'assembly, le nom de l'espace de nom / classe où cette procédure est placée et le nom

de la méthode dans la classe contenant le code qui sera exposé en tant que procédure.

Lire Intégration du Common Language Runtime en ligne: https://riptutorial.com/fr/sql-server/topic/7116/integration-du-common-language-runtime

Chapitre 58: Interrogation des résultats par page

Examples

Row_Number ()

```
SELECT Row_Number() OVER(ORDER BY UserName) As RowID, UserFirstName, UserLastName
FROM Users
```

A partir de laquelle il générera un jeu de résultats avec un champ RowID que vous pourrez utiliser pour naviguer entre les deux.

```
SELECT *
FROM
    ( SELECT Row_Number() OVER(ORDER BY UserName) As RowID, UserFirstName, UserLastName
      FROM Users
    ) As RowResults
WHERE RowID Between 5 AND 10
```

Lire Interrogation des résultats par page en ligne: https://riptutorial.com/fr/sql-server/topic/5803/interrogation-des-resultats-par-page

Chapitre 59: Joindre

Introduction

En langage SQL (Structured Query Language), une méthode JOIN permet de lier deux tables de données dans une même requête, permettant à la base de données de renvoyer un ensemble contenant des données provenant des deux tables ou d'utiliser les données d'une table pour les utiliser. Filtrer sur le deuxième tableau. Il existe plusieurs types de JOIN définis dans le standard SQL ANSI.

Examples

Jointure interne

`Inner join` renvoie uniquement les enregistrements / lignes qui correspondent / existent dans les deux tables en fonction d'une ou de plusieurs conditions (spécifiées à l'aide du mot clé `ON`). C'est le type de jointure le plus courant. La syntaxe générale pour `inner join` est la suivante:

```
SELECT *
FROM table_1
INNER JOIN table_2
  ON table_1.column_name = table_2.column_name
```

Il peut également être simplifié comme simplement `JOIN` :

```
SELECT *
FROM table_1
JOIN table_2
  ON table_1.column_name = table_2.column_name
```

Exemple

```
/* Sample data. */
DECLARE @Animal table (
    AnimalId Int IDENTITY,
    Animal Varchar(20)
);

DECLARE @AnimalSound table (
    AnimalSoundId Int IDENTITY,
    AnimalId Int,
    Sound Varchar(20)
);

INSERT INTO @Animal (Animal) VALUES ('Dog');
INSERT INTO @Animal (Animal) VALUES ('Cat');
INSERT INTO @Animal (Animal) VALUES ('Elephant');

INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (1, 'Barks');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (2, 'Meows');
```

```
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (3, 'Trumpets');
/* Sample data prepared. */

SELECT
    *
FROM
    @Animal
    JOIN @AnimalSound
        ON @Animal.AnimalId = @AnimalSound.AnimalId;
```

```
AnimalId    Animal               AnimalSoundId AnimalId    Sound
----------- -------------------- ------------- ----------- --------------------
1           Dog                  1             1           Barks
2           Cat                  2             2           Meows
3           Elephant             3             3           Trumpets
```

Utiliser la jointure interne avec la jointure externe gauche (Substitute for Not existe)

Cette requête renverra des données de la table 1 où les champs correspondant à table2 avec une clé et des données ne figurant pas dans le tableau 1 lors de la comparaison avec Table2 avec une condition et une clé

```
select *
  from Table1 t1
    inner join Table2 t2 on t1.ID_Column = t2.ID_Column
    left  join Table3 t3 on t1.ID_Column = t3.ID_Column
  where t2.column_name = column_value
    and t3.ID_Column is null
  order by t1.column_name;
```

Cross Join

`A cross join` est une jointure cartésienne, signifiant un produit cartésien des deux tables. Cette jointure ne nécessite aucune condition pour joindre deux tables. Chaque ligne de la table de gauche se joindra à chaque ligne de la table de droite. Syntaxe pour une jointure croisée:

```
SELECT * FROM table_1
CROSS JOIN table_2
```

Exemple:

```
/* Sample data. */
DECLARE @Animal table (
    AnimalId Int IDENTITY,
    Animal Varchar(20)
);

DECLARE @AnimalSound table (
    AnimalSoundId Int IDENTITY,
    AnimalId Int,
    Sound Varchar(20)
);

INSERT INTO @Animal (Animal) VALUES ('Dog');
```

```
INSERT INTO @Animal (Animal) VALUES ('Cat');
INSERT INTO @Animal (Animal) VALUES ('Elephant');

INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (1, 'Barks');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (2, 'Meows');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (3, 'Trumpet');
/* Sample data prepared. */

SELECT
    *
FROM
    @Animal
    CROSS JOIN @AnimalSound;
```

Résultats:

```
AnimalId    Animal              AnimalSoundId AnimalId    Sound
----------- ------------------- ------------- ----------- --------------------
1           Dog                 1             1           Barks
2           Cat                 1             1           Barks
3           Elephant            1             1           Barks
1           Dog                 2             2           Meows
2           Cat                 2             2           Meows
3           Elephant            2             2           Meows
1           Dog                 3             3           Trumpet
2           Cat                 3             3           Trumpet
3           Elephant            3             3           Trumpet
```

Notez qu'il existe d'autres manières d'appliquer une CROSS JOIN. Il s'agit d'une jointure *"ancienne"* (obsolète depuis ANSI SQL-92) sans condition, ce qui entraîne une jointure croisée / cartésienne:

```
SELECT *
FROM @Animal, @AnimalSound;
```

Cette syntaxe fonctionne également grâce à une condition de jointure "toujours vraie", mais elle n'est pas recommandée et doit être évitée, au profit de la syntaxe explicite de `CROSS JOIN` , dans un souci de lisibilité.

```
SELECT *
FROM
    @Animal
    JOIN @AnimalSound
        ON 1=1
```

Jointure externe

Jointure externe gauche

`LEFT JOIN` renvoie toutes les lignes de la table de gauche, correspondant aux lignes de la table de droite où les conditions de la clause `ON` sont remplies. Les lignes dans lesquelles la clause `ON` n'est pas remplie ont `NULL` dans toutes les colonnes de la table de droite. La syntaxe d'une `LEFT JOIN` est la suivante:

```
SELECT * FROM table_1 AS t1
LEFT JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Right Outer Join

`RIGHT JOIN` renvoie toutes les lignes de la table de droite, correspondant aux lignes de la table de gauche où les conditions de la clause `ON` sont remplies. Les lignes dans lesquelles la clause `ON` n'est pas remplie ont la `NULL` dans toutes les colonnes de la table de gauche. La syntaxe de `RIGHT JOIN` est la suivante:

```
SELECT * FROM table_1 AS t1
RIGHT JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Full Outer Join

`FULL JOIN` combine `LEFT JOIN` et `RIGHT JOIN` . Toutes les lignes sont renvoyées des deux tables, que les conditions de la clause `ON` soient remplies ou non. Les lignes qui ne satisfont pas à la clause `ON` sont renvoyées avec `NULL` dans toutes les colonnes de la table opposée (c'est-à-dire que pour une ligne de la table de gauche, toutes les colonnes de la table de droite contiendront `NULL` et inversement). La syntaxe d'un `FULL JOIN` est la suivante:

```
SELECT * FROM table_1 AS t1
FULL JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Exemples

```
/* Sample test data. */
DECLARE @Animal table (
    AnimalId Int IDENTITY,
    Animal Varchar(20)
);

DECLARE @AnimalSound table (
    AnimalSoundId Int IDENTITY,
    AnimalId Int,
    Sound Varchar(20)
);

INSERT INTO @Animal (Animal) VALUES ('Dog');
INSERT INTO @Animal (Animal) VALUES ('Cat');
INSERT INTO @Animal (Animal) VALUES ('Elephant');
INSERT INTO @Animal (Animal) VALUES ('Frog');

INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (1, 'Barks');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (2, 'Meows');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (3, 'Trumpet');
INSERT INTO @AnimalSound (AnimalId, Sound) VALUES (5, 'Roars');
/* Sample data prepared. */
```

JOINTURE EXTERNE GAUCHE

```
SELECT *
FROM @Animal As t1
```

```
LEFT JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Résultats pour `LEFT JOIN`

```
AnimalId    Animal               AnimalSoundId AnimalId    Sound
----------- -------------------- ------------- ----------- --------------------
1           Dog                  1             1           Barks
2           Cat                  2             2           Meows
3           Elephant             3             3           Trumpet
4           Frog                 NULL          NULL        NULL
```

DROIT EXTERIEUR

```
SELECT *
FROM @Animal As t1
RIGHT JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Résultats pour `RIGHT JOIN`

```
AnimalId    Animal               AnimalSoundId AnimalId    Sound
----------- -------------------- ------------- ----------- --------------------
1           Dog                  1             1           Barks
2           Cat                  2             2           Meows
3           Elephant             3             3           Trumpet
NULL        NULL                 4             5           Roars
```

FULL OUTER JOIN

```
SELECT *
FROM @Animal As t1
FULL JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Résultats pour `FULL JOIN`

```
AnimalId    Animal               AnimalSoundId AnimalId    Sound
----------- -------------------- ------------- ----------- --------------------
1           Dog                  1             1           Barks
2           Cat                  2             2           Meows
3           Elephant             3             3           Trumpet
4           Frog                 NULL          NULL        NULL
NULL        NULL                 4             5           Roars
```

Utiliser la jointure dans une mise à jour

Les jointures peuvent également être utilisées dans une instruction `UPDATE` :

```
CREATE TABLE Users (
    UserId int NOT NULL,
    AccountId int NOT NULL,
    RealName nvarchar(200) NOT NULL
)
```

```
CREATE TABLE Preferences (
    UserId int NOT NULL,
    SomeSetting bit NOT NULL
)
```

Mettez à jour la colonne `SomeSetting` du filtrage des tables de `Preferences` par un prédicat de la table `Users` comme suit:

```
UPDATE p
SET p.SomeSetting = 1
FROM Users u
JOIN Preferences p ON u.UserId = p.UserId
WHERE u.AccountId = 1234
```

`p` est un alias de `Preferences` défini dans la clause `FROM` de l'instruction. Seules les lignes avec un `AccountId` correspondant de la table `Users` seront mises à jour.

Mise à jour avec les instructions de jointure externes à gauche

```
Update t
SET  t.Column1=100
FROM Table1 t LEFT JOIN Table12 t2
ON t2.ID=t.ID
```

Mise à jour des tables avec jointure interne et fonction d'agrégation

```
UPDATE t1
SET t1.field1 = t2.field2Sum
FROM table1 t1
INNER JOIN (select field3, sum(field2) as field2Sum
from table2
group by field3) as t2
on t2.field3 = t1.field3
```

Rejoindre une sous-requête

La jointure sur une sous-requête est souvent utilisée lorsque vous souhaitez obtenir des données agrégées (telles que Count, Avg, Max ou Min) à partir d'une table enfant / détails et l'afficher avec les enregistrements de la table parent / header. Par exemple, vous souhaiterez peut-être récupérer la ligne supérieure / première enfant en fonction de la date ou de l'ID ou peut-être souhaitez-vous un nombre total de lignes enfants ou une moyenne.

Cet exemple utilise des alias qui facilitent la lecture des requêtes lorsque plusieurs tables sont impliquées. Dans ce cas, nous récupérons toutes les lignes des commandes d'achat de la table parente et récupérons uniquement la dernière ligne enfant (ou la plus récente) de la table enfant PurchaseOrderLineItems. Cet exemple suppose que la table enfant utilise des identifiants numériques incrémentiels.

```
SELECT po.Id, po.PODate, po.VendorName, po.Status, item.ItemNo,
  item.Description, item.Cost, item.Price
FROM PurchaseOrders po
```

```
LEFT JOIN
     (
       SELECT l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price, Max(l.id) as Id
       FROM PurchaseOrderLineItems l
       GROUP BY l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price
     ) AS item ON item.PurchaseOrderId = po.Id
```

Self Join

Une table peut être jointe sur ce que l'on appelle une auto-jointure, en combinant des enregistrements de la table avec d'autres enregistrements de la même table. Les auto-jointures sont généralement utilisées dans les requêtes où une hiérarchie est définie dans les colonnes de la table.

Considérez les exemples de données dans une table appelée `Employees` :

ID	prénom	Boss_ID
1	Bob	3
2	Jim	1
3	Sam	2

Le `Boss_ID` chaque employé correspond à l' `ID` un autre employé. Pour récupérer une liste d'employés avec le nom de leur patron respectif, la table peut être jointe sur elle-même à l'aide de ce mappage. Notez que joindre une table de cette manière nécessite l'utilisation d'un alias (`Bosses` dans ce cas) sur la deuxième référence à la table pour se distinguer de la table d'origine.

```
SELECT Employees.Name,
    Bosses.Name AS Boss
FROM Employees
INNER JOIN Employees AS Bosses
    ON Employees.Boss_ID = Bosses.ID
```

L'exécution de cette requête génère les résultats suivants:

prénom	Patron
Bob	Sam
Jim	Bob
Sam	Jim

Supprimer en utilisant Join

Les jointures peuvent également être utilisées dans une instruction `DELETE` . Étant donné un schéma comme suit:

```
CREATE TABLE Users (
    UserId int NOT NULL,
    AccountId int NOT NULL,
    RealName nvarchar(200) NOT NULL
)

CREATE TABLE Preferences (
    UserId int NOT NULL,
    SomeSetting bit NOT NULL
)
```

Nous pouvons supprimer des lignes de la table `Preferences` , en les filtrant par un prédicat de la table `Users` comme suit:

```
DELETE p
FROM Users u
INNER JOIN Preferences p ON u.UserId = p.UserId
WHERE u.AccountId = 1234
```

Ici, `p` est un alias de `Preferences` défini dans la clause `FROM` de l'instruction et nous `AccountId` uniquement les lignes qui ont un `AccountId` correspondant de la table `Users` .

Tourner accidentellement une jointure externe en une jointure interne

Les jointures externes renvoient toutes les lignes d'une ou des deux tables, ainsi que les lignes correspondantes.

```
Table People
PersonID FirstName
       1 Alice
       2 Bob
       3 Eve

Table Scores
PersonID Subject Score
       1 Math    100
       2 Math     54
       2 Science  98
```

Gauche rejoignant les tables:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
```

Résultats:

```
PersonID FirstName PersonID Subject Score
       1 Alice            1 Math    100
       2 Bob              2 Math     54
       2 Bob              2 Science  98
       3 Eve           NULL NULL   NULL
```

Si vous vouliez renvoyer toutes les personnes, avec tous les scores de mathématiques applicables, une erreur commune est d'écrire:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
where Subject = 'Math'
```

Cela supprimerait Eve de vos résultats, en plus de supprimer le score scientifique de Bob, puisque `Subject` est `NULL` pour elle.

La syntaxe correcte pour supprimer des enregistrements non-Math en conservant tous les individus dans la table `People` serait:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
and b.Subject = 'Math'
```

Lire Joindre en ligne: https://riptutorial.com/fr/sql-server/topic/1008/joindre

Chapitre 60: JSON dans Sql Server

Syntaxe

- **JSON_VALUE** (expression, path) - extrait une valeur scalaire d'une chaîne JSON.
- **JSON_QUERY** (expression [, chemin]) - Extrait un objet ou un tableau d'une chaîne JSON.
- **OPENJSON** (jsonExpression [, chemin]) - fonction table-valeur qui analyse le texte JSON et renvoie les objets et propriétés dans JSON sous forme de lignes et de colonnes.
- **ISJSON** (expression) - **Vérifie** si une chaîne contient un JSON valide.
- **JSON_MODIFY** (expression, path, newValue) - Met à jour la valeur d'une propriété dans une chaîne JSON et renvoie la chaîne JSON mise à jour.

Paramètres

Paramètres	Détails
expression	Généralement, le nom d'une variable ou d'une colonne contenant du texte JSON.
chemin	Une expression de chemin JSON qui spécifie la propriété à mettre à jour. path a la syntaxe suivante: [append] [lax | strict] $. <chemin json>
jsonExpression	Est une expression de caractère Unicode contenant le texte JSON.

Remarques

La fonction OPENJSON est uniquement disponible sous le niveau de compatibilité 130. Si le niveau de compatibilité de votre base de données est inférieur à 130, SQL Server ne pourra pas trouver et exécuter la fonction OPENJSON. Actuellement, toutes les bases de données SQL Azure sont définies sur 120 par défaut. Vous pouvez modifier le niveau de compatibilité d'une base de données à l'aide de la commande suivante:

```
ALTER DATABASE <Database-Name-Here> SET COMPATIBILITY_LEVEL = 130
```

Examples

Formater les résultats de la requête en tant que JSON avec FOR JSON

Données de la table d'entrée (table People)

Id	prénom	Âge
1	John	23

Id	prénom	Âge
2	Jeanne	31

Question

```
SELECT Id, Name, Age
FROM People
FOR JSON PATH
```

Résultat

```
[
    {"Id":1,"Name":"John","Age":23},
    {"Id":2,"Name":"Jane","Age":31}
]
```

Parse JSON text

Les fonctions **JSON_VALUE** et **JSON_QUERY analysent le** texte JSON et renvoient des valeurs scalaires ou des objets / tableaux sur le chemin du texte JSON.

```
DECLARE @json NVARCHAR(100) = '{"id": 1, "user":{"name":"John"}, "skills":["C#","SQL"]}'

SELECT
    JSON_VALUE(@json, '$.id') AS Id,
    JSON_VALUE(@json, '$.user.name') AS Name,
    JSON_QUERY(@json, '$.user') AS UserObject,
    JSON_QUERY(@json, '$.skills') AS Skills,
    JSON_VALUE(@json, '$.skills[0]') AS Skill0
```

Résultat

Id	prénom	UserObject	Compétences	Compétence0
1	John	{"name": "John"}	["C #", "SQL"]	C #

Joindre des entités JSON parent et enfant à l'aide de CROSS APPLY OPENJSON

Joindre des objets parents avec leurs entités enfants, par exemple, nous voulons une table relationnelle de chaque personne et de ses loisirs

```
DECLARE @json nvarchar(1000) =
N'[
    {
        "id":1,
        "user":{"name":"John"},
        "hobbies":[
            {"name": "Reading"},
            {"name": "Surfing"}
```

```
        ]
    },
    {
        "id":2,
        "user":{"name":"Jane"},
        "hobbies":[
            {"name": "Programming"},
            {"name": "Running"}
        ]
    }
]'
```

Question

```
SELECT
    JSON_VALUE(person.value, '$.id') as Id,
    JSON_VALUE(person.value, '$.user.name') as PersonName,
    JSON_VALUE(hobbies.value, '$.name') as Hobby
FROM OPENJSON (@json) as person
    CROSS APPLY OPENJSON(person.value, '$.hobbies') as hobbies
```

Cette requête peut également être écrite à l'aide de la clause WITH.

```
SELECT
    Id, person.PersonName, Hobby
FROM OPENJSON (@json)
WITH(
    Id int '$.id',
    PersonName nvarchar(100) '$.user.name',
    Hobbies nvarchar(max) '$.hobbies' AS JSON
) as person
CROSS APPLY OPENJSON(Hobbies)
WITH(
    Hobby nvarchar(100) '$.name'
)
```

Résultat

Id	Nom d'une personne	Loisir
1	John	En train de lire
1	John	Surfant
2	Jeanne	La programmation
2	Jeanne	Fonctionnement

Index sur les propriétés JSON en utilisant des colonnes calculées

Lors du stockage de documents JSON dans SQL Server, nous devons pouvoir filtrer et trier efficacement les résultats des requêtes sur les propriétés des documents JSON.

```
CREATE TABLE JsonTable
(
    id int identity primary key,
    jsonInfo nvarchar(max),
    CONSTRAINT [Content should be formatted as JSON]
    CHECK (ISJSON(jsonInfo)>0)
)
```

```
INSERT INTO JsonTable
VALUES(N'{"Name":"John","Age":23}'),
(N'{"Name":"Jane","Age":31}'),
(N'{"Name":"Bob","Age":37}'),
(N'{"Name":"Adam","Age":65}')
GO
```

Étant donné le tableau ci-dessus Si nous voulons trouver la ligne avec le nom = 'Adam', nous exécuterons la requête suivante.

```
SELECT *
FROM JsonTable Where
JSON_VALUE(jsonInfo, '$.Name') = 'Adam'
```

Cependant, cela nécessitera que SQL Server exécute une table complète qui, sur une grande table, n'est pas efficace.

Pour accélérer cela, nous aimerions ajouter un index, mais nous ne pouvons pas référencer directement les propriétés dans le document JSON. La solution consiste à ajouter une colonne calculée sur le chemin JSON `$.Name` , puis à ajouter un index sur la colonne calculée.

```
ALTER TABLE JsonTable
ADD vName as JSON_VALUE(jsonInfo, '$.Name')

CREATE INDEX idx_name
ON JsonTable(vName)
```

Maintenant, lorsque nous exécutons la même requête, au lieu d'une analyse de table complète, SQL Server utilise un index pour rechercher dans l'index non clusterisé et rechercher les lignes qui répondent aux conditions spécifiées.

Remarque: Pour que SQL Server utilise l'index, vous devez créer la colonne calculée avec la même expression que celle que vous prévoyez d'utiliser dans vos requêtes - dans cet exemple, `JSON_VALUE(jsonInfo, '$.Name')` . Cependant, vous pouvez également utiliser le nom de la colonne calculée `vName`

Mettre en forme une ligne de tableau en tant qu'objet JSON unique à l'aide de FOR JSON

L' option **WITHOUT_ARRAY_WRAPPER de la** clause *FOR JSON* supprime les crochets de tableau de la sortie JSON. Ceci est utile si vous retournez une seule ligne dans la requête.

> Remarque: cette option produira une sortie JSON non valide si plusieurs lignes sont

renvoyées.

Données de la table d'entrée (table People)

Id	prénom	Âge
1	John	23
2	Jeanne	31

Question

```
SELECT Id, Name, Age
FROM People
WHERE Id = 1
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

Résultat

```
{"Id":1,"Name":"John","Age":23}
```

Analyser le texte JSON en utilisant la fonction OPENJSON

La fonction **OPENJSON** analyse le texte JSON et renvoie plusieurs sorties. Les valeurs à renvoyer sont spécifiées à l'aide des chemins définis dans la clause WITH. Si aucun chemin n'est spécifié pour une colonne, le nom de la colonne est utilisé comme chemin. Cette fonction renvoie les valeurs aux types SQL définis dans la clause WITH. L'option AS JSON doit être spécifiée dans la définition de la colonne si un objet / tableau doit être renvoyé.

```
DECLARE @json NVARCHAR(100) = '{"id": 1, "user":{"name":"John"}, "skills":["C#","SQL"]}'

SELECT *
FROM OPENJSON (@json)
    WITH(Id int '$.id',
        Name nvarchar(100) '$.user.name',
        UserObject nvarchar(max) '$.user' AS JSON,
        Skills nvarchar(max) '$.skills' AS JSON,
        Skill0 nvarchar(20) '$.skills[0]')
```

Résultat

Id	prénom	UserObject	Compétences	Compétence0
1	John	{"name": "John"}	["C #", "SQL"]	C #

Lire JSON dans Sql Server en ligne: https://riptutorial.com/fr/sql-server/topic/2568/json-dans-sql-server

Chapitre 61: La fonction STUFF

Paramètres

Paramètre	Détails
expression_caractère	la chaîne existante dans vos données
la position de départ	la position dans `character_expression` pour supprimer la `length` , puis insérez la chaîne `replacement_string`
longueur	le nombre de caractères à supprimer de `character_expression`
remplacement_string	la séquence de caractères à insérer dans `character_expression`

Examples

Remplacement de caractères de base avec STUFF ()

La fonction `STUFF()` insère une chaîne dans une autre chaîne en supprimant d'abord un nombre spécifié de caractères. L'exemple suivant supprime "Svr" et le remplace par "Serveur". Cela se produit en spécifiant la position de `start_position` et la `length` du remplacement.

```
SELECT STUFF('SQL Svr Documentation', 5, 3, 'Server')
```

L'exécution de cet exemple entraînera le renvoi de la `SQL Server Documentation` au lieu de la `SQL Svr Documentation.`

Utilisation de FOR XML pour concaténer des valeurs à partir de plusieurs lignes

Une utilisation courante de la fonction `FOR XML` consiste à concaténer les valeurs de plusieurs lignes.

Voici un exemple d'utilisation de la table Customers :

```
SELECT
    STUFF( (SELECT ';' + Email
        FROM Customers
        where (Email is not null and Email <> '')
        ORDER BY Email ASC
        FOR XML PATH('')),
    1, 1, '')
```

Dans l'exemple ci-dessus, `FOR XML PATH('')` est utilisé pour concaténer les adresses électroniques, en utilisant `;` comme caractère de délimiteur. En outre, le but de `STUFF` est de

supprimer le principal ; de la chaîne concaténée. `STUFF` également implicitement la chaîne concaténée de XML à varchar.

Remarque: le résultat de l'exemple ci-dessus sera codé en XML, ce qui signifie qu'il remplacera `<` caractères par `<` etc. Si vous ne le voulez pas, modifiez `FOR XML PATH(''))` en `FOR XML PATH, TYPE).value('.[1]','varchar(MAX)')` , par exemple:

```
SELECT
    STUFF( (SELECT ';' + Email
        FROM Customers
        where (Email is not null and Email <> '')
        ORDER BY Email ASC
        FOR XML PATH, TYPE).value('.[1]','varchar(900)'),
    1, 1, '')
```

Cela peut être utilisé pour obtenir un résultat similaire à `GROUP_CONCAT` dans MySQL ou `string_agg` dans PostgreSQL 9.0+, bien que nous utilisions des sous-requêtes au lieu des agrégats GROUP BY. (En guise d'alternative, vous pouvez installer un agrégat défini par l'utilisateur tel que celui-ci si vous recherchez une fonctionnalité plus proche de celle de `GROUP_CONCAT`).

Obtenir des noms de colonne séparés par une virgule (pas une liste)

```
/*
The result can be use for fast way to use columns on Insertion/Updates.
Works with tables and views.

Example: eTableColumns  'Customers'
ColumnNames
-----------------------------------------------------
Id, FName, LName, Email, PhoneNumber, PreferredContact

INSERT INTO Customers (Id, FName, LName, Email, PhoneNumber, PreferredContact)
    VALUES (5, 'Ringo', 'Star', 'two@beatles.now', NULL, 'EMAIL')
*/
CREATE PROCEDURE eTableColumns (@Table VARCHAR(100))
AS
SELECT ColumnNames =
   STUFF( (SELECT ', ' +  c.name
FROM
    sys.columns c
INNER JOIN
    sys.types t ON c.user_type_id = t.user_type_id
WHERE
    c.object_id = OBJECT_ID( @Table)
        FOR XML PATH, TYPE).value('.[1]','varchar(2000)'),
    1, 1, '')
GO
```

choses pour les virgules séparées dans le serveur SQL

`FOR XML PATH` et `STUFF` pour concaténer les plusieurs lignes en une seule ligne:

```
  select distinct t1.id,
      STUFF(
```

```
            (SELECT ', ' + convert(varchar(10), t2.date, 120)
             FROM yourtable t2
             where t1.id = t2.id
             FOR XML PATH (''))
             , 1, 1, '')  AS date
    from yourtable t1;
```

Exemple de base de la fonction STUFF ().

STUFF (Original_Expression, Start, Length, Replacement_expression)

La fonction STUFF () insère l'expression Remplacement_, à la position de début spécifiée, et supprime les caractères spécifiés à l'aide du paramètre Longueur.

```
Select FirstName, LastName,Email, STUFF(Email, 2, 3, '******') as StuffedEmail From Employee
```

L'exécution de cet exemple entraînera le renvoi de la table donnée

Prénom	Nom de famille	Email	StuffedEmail
Jomes	chasseur	James@hotmail.com	J*****s@hotmail.com
Shyam	rathod	Shyam@hotmail.com	S*****m@hotmail.com
RAM	shinde	Ram@hotmail.com	R ***** hotmail.com

Lire La fonction STUFF en ligne: https://riptutorial.com/fr/sql-server/topic/703/la-fonction-stuff

Chapitre 62: Les variables

Syntaxe

- DECLARE @VariableName DataType [= Valeur];
- SET @VariableName = Valeur;

Examples

Déclarez une variable de table

```
DECLARE @Employees TABLE
(
    EmployeeID INT NOT NULL PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    ManagerID INT NULL
)
```

Lorsque vous créez une table normale, vous utilisez la syntaxe `CREATE TABLE Name (Columns)` . Lors de la création d'une variable de table, vous utilisez la `DECLARE @Name TABLE (Columns)` .

Pour référencer la variable de table dans une `SELECT` , SQL Server exige que vous donniez à la variable de table un alias, sinon vous obtenez une erreur:

> Doit déclarer la variable scalaire "@TableVariableName".

c'est à dire

```
DECLARE @Table1 TABLE (Example INT)
DECLARE @Table2 TABLE (Example INT)

/*
-- the following two commented out statements would generate an error:
SELECT *
FROM @Table1
INNER JOIN @Table2 ON @Table1.Example = @Table2.Example

SELECT *
FROM @Table1
WHERE @Table1.Example = 1
*/

-- but these work fine:
SELECT *
FROM @Table1 T1
INNER JOIN @Table2 T2 ON T1.Example = T2.Example

SELECT *
FROM @Table1 Table1
WHERE Table1.Example = 1
```

Mise à jour d'une variable à l'aide de SET

```
DECLARE @VariableName INT
SET @VariableName = 1
PRINT @VariableName
```

1

En utilisant `SET` , vous ne pouvez mettre à jour qu'une seule variable à la fois.

Mise à jour des variables à l'aide de SELECT

En utilisant `SELECT` , vous pouvez mettre à jour plusieurs variables à la fois.

```
DECLARE @Variable1 INT, @Variable2 VARCHAR(10)
SELECT @Variable1 = 1, @Variable2 = 'Hello'
PRINT @Variable1
PRINT @Variable2
```

1

Bonjour

Lorsque vous utilisez `SELECT` pour mettre à jour une variable à partir d'une colonne de table, s'il existe plusieurs valeurs, il utilisera la *dernière* valeur. (Les règles d'ordre normal s'appliquent - si aucun tri n'est donné, la commande n'est pas garantie).

```
CREATE TABLE #Test (Example INT)
INSERT INTO #Test VALUES (1), (2)

DECLARE @Variable INT
SELECT @Variable = Example
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

2

```
SELECT TOP 1 @Variable = Example
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

1

Si la requête ne renvoie aucune ligne, la valeur de la variable ne changera pas:

```
SELECT TOP 0 @Variable = Example
```

```
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

1

Déclarez plusieurs variables à la fois, avec des valeurs initiales

```
DECLARE
  @Var1 INT = 5,
  @Var2 NVARCHAR(50) = N'Hello World',
  @Var3 DATETIME = GETDATE()
```

Opérateurs d'affectation composés

SQL Server 2008 R2

Opérateurs composés pris en charge:

- `+=` Ajouter et assigner
- `-=` Soustraire et attribuer
- `*=` Multiplier et assigner
- `/=` Diviser et assigner
- `%=` Modulo et assign
- `&=` Bitwise AND et assigner
- `^=` Bit à bit XOR et assigner
- `|=` OU bit à bit et attribuer

Exemple d'utilisation:

```
DECLARE @test INT = 42;
SET @test += 1;
PRINT @test;    --43
SET @test -= 1;
PRINT @test;    --42
SET @test *= 2
PRINT @test;    --84
SET @test /= 2;
PRINT @test;    --42
```

Mettre à jour les variables en les sélectionnant dans une table

Selon la structure de vos données, vous pouvez créer des variables qui se mettent à jour dynamiquement.

```
DECLARE @CurrentID int = (SELECT TOP 1 ID FROM Table ORDER BY CreateDate desc)


DECLARE @Year int = 2014
DECLARE @CurrentID int = (SELECT ID FROM Table WHERE Year = @Year)
```

Dans la plupart des cas, vous souhaiterez vous assurer que votre requête ne renvoie qu'une valeur lorsque vous utilisez cette méthode.

Lire Les variables en ligne: https://riptutorial.com/fr/sql-server/topic/2566/les-variables

Chapitre 63: Magasin de requêtes

Examples

Activer le magasin de requêtes sur la base de données

Le magasin de requêtes peut être activé sur la base de données à l'aide de la commande suivante:

```
ALTER DATABASE tpch SET QUERY_STORE = ON
```

SQL Server / Azure SQL Database collectera des informations sur les requêtes exécutées et fournira des informations dans les vues sys.query_store:

- sys.query_store_query
- sys.query_store_query_text
- sys.query_store_plan
- sys.query_store_runtime_stats
- sys.query_store_runtime_stats_interval
- sys.database_query_store_options
- sys.query_context_settings

Obtenir des statistiques d'exécution pour les requêtes / plans SQL

La requête suivante renverra des informations sur les séries, leurs plans et les statistiques moyennes concernant leur durée, le temps processeur, les lectures physiques et logiques.

```
SELECT Txt.query_text_id, Txt.query_sql_text, Pl.plan_id,
        avg_duration, avg_cpu_time,
        avg_physical_io_reads, avg_logical_io_reads
FROM sys.query_store_plan AS Pl
JOIN sys.query_store_query AS Qry
    ON Pl.query_id = Qry.query_id
JOIN sys.query_store_query_text AS Txt
    ON Qry.query_text_id = Txt.query_text_id
JOIN sys.query_store_runtime_stats Stats
    ON Pl.plan_id = Stats.plan_id
```

Supprimer des données du magasin de requêtes

Si vous souhaitez supprimer un plan de requête ou de requête du magasin de requêtes, vous pouvez utiliser les commandes suivantes:

```
EXEC sp_query_store_remove_query 4;
EXEC sp_query_store_remove_plan 3;
```

Les paramètres de ces procédures stockées sont un identifiant de requête / plan extrait des vues

système.

Vous pouvez également supprimer les statistiques d'exécution d'un plan particulier sans supprimer le plan du magasin:

```
EXEC sp_query_store_reset_exec_stats 3;
```

Paramètre fourni à cet identifiant de plan de procédure.

Plan de forçage pour la requête

L'optimiseur de requêtes SQL choisira le plan possible qu'il pourra trouver pour certaines requêtes. Si vous trouvez un plan qui fonctionne de manière optimale pour certaines requêtes, vous pouvez forcer QO à toujours utiliser ce plan à l'aide de la procédure stockée suivante:

```
EXEC sp_query_store_unforce_plan @query_id, @plan_id
```

A partir de là, QO utilisera toujours le plan fourni pour la requête.

Si vous souhaitez supprimer cette liaison, vous pouvez utiliser la procédure stockée suivante:

```
EXEC sp_query_store_force_plan @query_id, @plan_id
```

À partir de ce moment, QO tentera à nouveau de trouver le meilleur plan.

Lire Magasin de requêtes en ligne: https://riptutorial.com/fr/sql-server/topic/7349/magasin-de-requetes

Chapitre 64: Masquage dynamique des données

Examples

Masquer l'adresse e-mail à l'aide du masquage dynamique des données

Si vous avez une colonne email, vous pouvez la masquer avec email () mask:

```
ALTER TABLE  Company
ALTER COLUMN Email ADD MASKED WITH (FUNCTION = 'email()')
```

Lorsque l'utilisateur essaie de sélectionner des e-mails dans la table de la société, il obtient les valeurs suivantes:

mXXX@XXXX.com

zXXX@XXXX.com

rXXX@XXXX.com

Ajouter un masque partiel sur la colonne

Vous pouvez ajouter un masque partiel sur la colonne qui affichera quelques caractères du début et de la fin de la chaîne et afficher le masque à la place des caractères au milieu:

```
ALTER TABLE  Company
ALTER COLUMN Phone ADD MASKED WITH (FUNCTION = 'partial(5,"XXXXXXX",2)')
```

Dans les paramètres de la fonction partielle, vous pouvez spécifier le nombre de valeurs affichées au début, le nombre de valeurs affichées à la fin et le modèle affiché au milieu.

Lorsque l'utilisateur essaie de sélectionner des e-mails dans la table de la société, il obtient les valeurs suivantes:

(381) XXXXXXX39

(360) XXXXXXX01

(415) XXXXXXX05

Affichage de la valeur aléatoire de la plage à l'aide du masque random ()

Le masque aléatoire affichera un nombre de rundom de la plage spécifiée au lieu de la valeur réelle:

```
ALTER TABLE  Product
ALTER COLUMN Price ADD MASKED WITH (FUNCTION = 'random(100,200)')
```

Notez que dans certains cas, la valeur affichée peut correspondre à la valeur réelle dans la colonne (si le nombre sélectionné de manière aléatoire correspond à la valeur dans la cellule).

Ajout d'un masque par défaut dans la colonne

Si vous ajoutez un masque par défaut sur la colonne, au lieu de la valeur réelle dans l'instruction SELECT sera affiché le masque:

```
ALTER TABLE  Company
ALTER COLUMN Postcode ADD MASKED WITH (FUNCTION = 'default()')
```

Contrôler qui peut voir les données non masquées

Vous pouvez accorder aux utilisateurs privilégiés le droit de voir les valeurs non masquées à l'aide de l'instruction suivante:

```
GRANT UNMASK TO MyUser
```

Si certains utilisateurs ont déjà l'autorisation de démasquer, vous pouvez révoquer cette autorisation:

```
REVOKE UNMASK TO MyUser
```

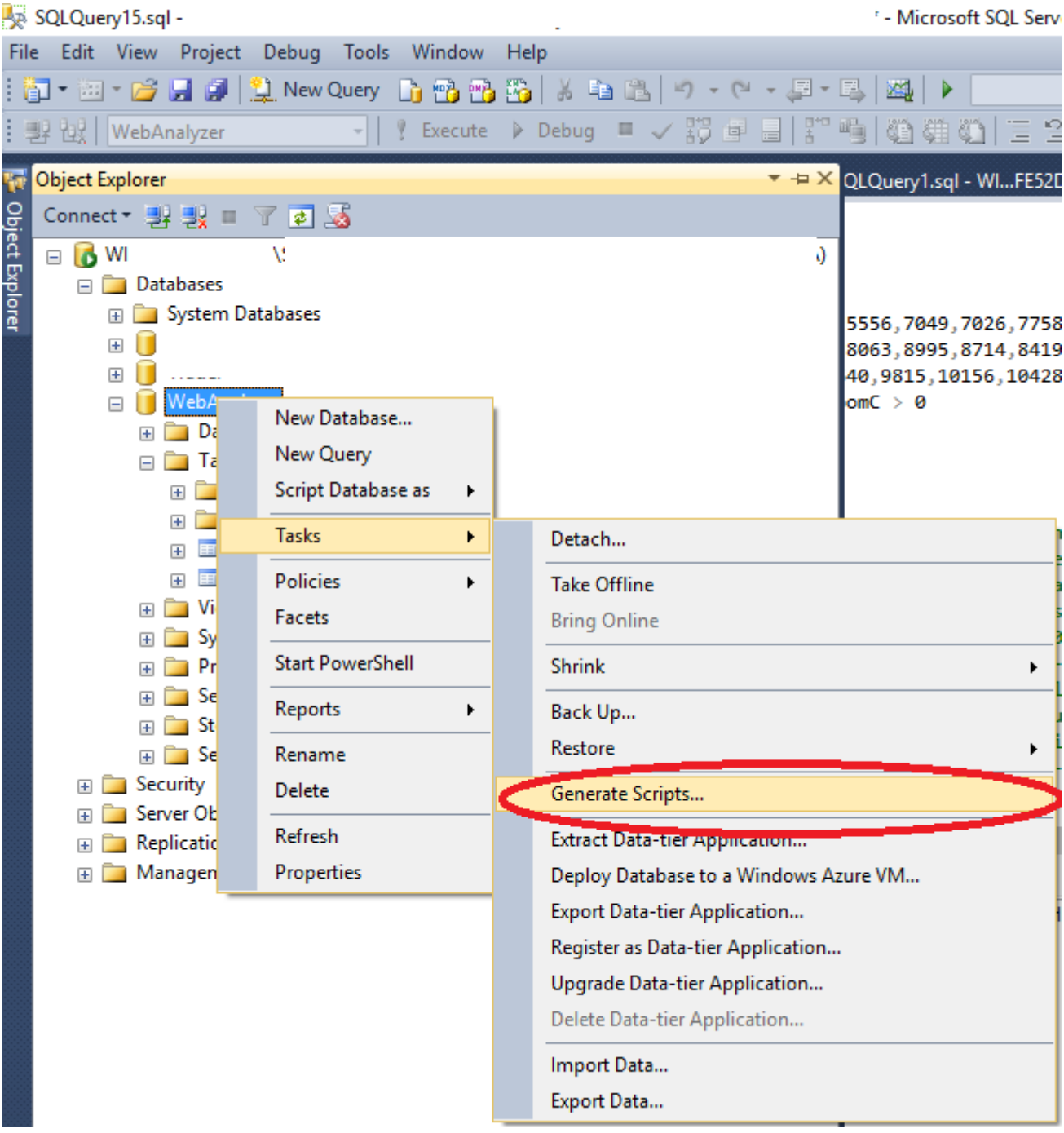
Lire Masquage dynamique des données en ligne: https://riptutorial.com/fr/sql-server/topic/7052/masquage-dynamique-des-donnees

Chapitre 65: Migration

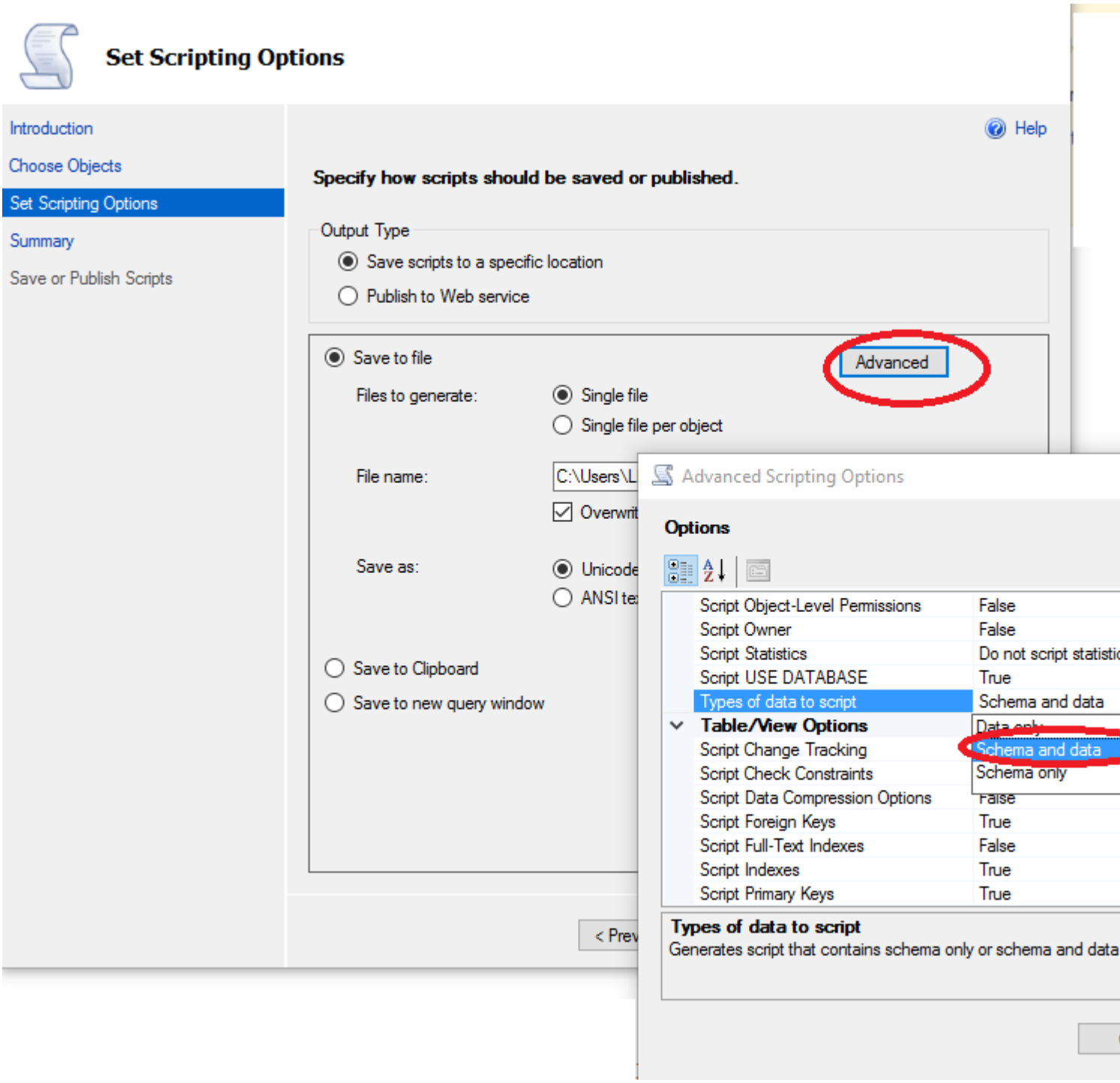
Examples

Comment générer des scripts de migration

1. Cliquez avec le `bouton droit de la souris` sur la base de données que vous souhaitez migrer, puis -> `Tasks` -> `Generate Scripts...`



2. L'assistant s'ouvrira, cliquez sur `Next` puis choisissez les objets que vous souhaitez migrer et cliquez à nouveau sur `Next` , puis cliquez sur `Advanced` un peu plus bas et dans `Types of data to script` choisissez `Schema and data` (sauf si vous ne voulez que des structures)



3. Cliquez deux fois sur `Next` et `Finish` et vous devriez avoir votre base de données scriptée dans le fichier `.sql` .

4. exécutez le fichier `.sql` sur votre nouveau serveur et vous devriez avoir terminé.

Lire Migration en ligne: https://riptutorial.com/fr/sql-server/topic/4451/migration

Chapitre 66: Modifier le texte JSON

Examples

Modifier la valeur en texte JSON sur le chemin spécifié

La fonction JSON_MODIFY utilise le texte JSON comme paramètre d'entrée et modifie une valeur sur le chemin spécifié à l'aide du troisième argument:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Price":34.99}'
set @json = JSON_MODIFY(@json, '$.Price', 39.99)
print @json -- Output: {"Id":1,"Name":"Toy Car","Price":39.99}
```

En conséquence, nous aurons un nouveau texte JSON avec "Prix": 39.99 et aucune autre valeur ne sera modifiée. Si l'objet sur le chemin spécifié n'existe pas, JSON_MODIFY insérera la paire clé: valeur.

Pour supprimer la paire clé: valeur, mettez NULL comme nouvelle valeur:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Price":34.99}'
set @json = JSON_MODIFY(@json, '$.Price', NULL)
print @json -- Output: {"Id":1,"Name":"Toy Car"}
```

JSON_MODIFY supprimera par défaut la clé si elle n'a pas de valeur, vous pouvez donc l'utiliser pour supprimer une clé.

Ajouter une valeur scalaire dans un tableau JSON

JSON_MODIFY a le mode 'append' qui ajoute de la valeur dans le tableau.

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Tags":["toy","game"]}'
set @json = JSON_MODIFY(@json, 'append $.Tags', 'sales')
print @json -- Output: {"Id":1,"Name":"Toy Car","Tags":["toy","game","sales"]}
```

Si le tableau sur le chemin spécifié n'existe pas, JSON_MODIFY (append) créera un nouveau tableau avec un seul élément:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car","Price":34.99}'
set @json = JSON_MODIFY(@json, 'append $.Tags', 'sales')
print @json -- Output {"Id":1,"Name":"Toy Car","Tags":["sales"]}
```

Insérer un nouvel objet JSON dans le texte JSON

La fonction JSON_MODIFY vous permet d'insérer des objets JSON dans du texte JSON:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car"}'
set @json = JSON_MODIFY(@json, '$.Price',
```

```
                        JSON_QUERY('{"Min":34.99,"Recommended":45.49}'))
print @json
-- Output: {"Id":1,"Name":"Toy Car","Price":{"Min":34.99,"Recommended":45.49}}
```

Comme le troisième paramètre est un texte, vous devez l'emballer avec la fonction JSON_QUERY pour "lancer" du texte en JSON. Sans cette "distribution", JSON_MODIFY traitera le troisième paramètre comme du texte brut et des caractères d'échappement avant de l'insérer en tant que valeur de chaîne. Sans JSON_QUERY, les résultats seront:

```
{"Id":1,"Name":"Toy Car","Price":'{\"Min\":34.99,\"Recommended\":45.49}'}
```

JSON_MODIFY insérera cet objet s'il n'existe pas ou le supprimera si la valeur du troisième paramètre est NULL.

Insérer un nouveau tableau JSON généré avec la requête FOR JSON

Vous pouvez générer un objet JSON en utilisant la requête SELECT standard avec la clause FOR JSON et l'insérer dans le texte JSON en tant que troisième paramètre:

```
declare @json nvarchar(4000) = N'{"Id":17,"Name":"WWI"}'
set @json = JSON_MODIFY(@json, '$.tables',
                        (select name from sys.tables FOR JSON PATH) )
print @json

(1 row(s) affected)
{"Id":1,"Name":"master","tables":[{"name":"Colors"},{"name":"Colors_Archive"},{"name":"OrderLines"},{"n
```

JSON_MODIFY saura que la requête select avec la clause FOR JSON génère un tableau JSON valide et l'insère simplement dans le texte JSON.

> Vous pouvez utiliser toutes les options FOR JSON dans la requête SELECT, à l' **exception de WITHOUT_ARRAY_WRAPPER** , qui générera un objet unique au lieu d'un tableau JSON. Voir un autre exemple dans cette rubrique pour voir comment insérer un objet JSON unique.

Insérer un objet JSON unique généré avec la clause FOR JSON

Vous pouvez générer un objet JSON à l'aide d'une requête SELECT standard avec la clause FOR JSON et l'option WITHOUT_ARRAY_WRAPPER, puis l'insérer dans un texte JSON en tant que troisième paramètre:

```
declare @json nvarchar(4000) = N'{"Id":17,"Name":"WWI"}'
set @json = JSON_MODIFY(@json, '$.table',
                        JSON_QUERY(
                         (select name, create_date, schema_id
                           from sys.tables
                           where name = 'Colors'
                           FOR JSON PATH, WITHOUT_ARRAY_WRAPPER)))
print @json
```

```
(1 row(s) affected)
{"Id":17,"Name":"WWI","table":{"name":"Colors","create_date":"2016-06-
02T10:04:03.280","schema_id":13}}
```

FOR JSON avec l'option WITHOUT_ARRAY_WRAPPER peut générer du texte JSON non valide si la requête SELECT renvoie plusieurs résultats (vous devez utiliser TOP 1 ou filtrer par clé primaire dans ce cas). Par conséquent, JSON_MODIFY supposera que le résultat renvoyé est simplement un texte brut et l'échappe comme tout autre texte si vous ne l'emballez pas avec la fonction JSON_QUERY.

> Vous devez envelopper la requête **FOR JSON, WITHOUT_ARRAY_WRAPPER** avec la fonction **JSON_QUERY** afin de **convertir le** résultat en JSON.

Lire Modifier le texte JSON en ligne: https://riptutorial.com/fr/sql-server/topic/6883/modifier-le-texte-json

Chapitre 67: Modules compilés nativement (Hekaton)

Examples

Procédure stockée compilée nativement

Dans une procédure avec compilation native, le code T-SQL est compilé en dll et exécuté en code C natif. Pour créer une procédure stockée compilée native, vous devez:

- Utilisez la syntaxe standard CREATE PROCEDURE
- Définissez l'option NATIVE_COMPILATION dans la définition de procédure stockée
- Utilisez l'option SCHEMABINDING dans la définition de procédure stockée
- Définir l'option EXECUTE AS OWNER dans la définition de procédure stockée

Au lieu du bloc standard BEGIN END, vous devez utiliser le bloc BEGIN ATOMIC:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Exemple:

```
CREATE PROCEDURE usp_LoadMemOptTable (@maxRows INT, @FullName NVARCHAR(200))
WITH
    NATIVE_COMPILATION,
    SCHEMABINDING,
    EXECUTE AS OWNER
AS
BEGIN ATOMIC
WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
    DECLARE @i INT = 1
    WHILE @i <= @maxRows
    BEGIN
        INSERT INTO dbo.MemOptTable3 VALUES(@i, @FullName, GETDATE())
        SET @i = @i+1
    END
END
GO
```

Fonction scalaire compilée nativement

Le code dans la fonction compilée nativement sera transformé en code C et compilé en dll. Pour créer une fonction scalaire native compilée, vous devez:

- Utiliser la syntaxe standard CREATE FUNCTION
- Définissez l'option NATIVE_COMPILATION dans la définition de fonction

- Utilisez l'option SCHEMABINDING dans la définition de fonction

Au lieu du bloc standard BEGIN END, vous devez utiliser le bloc BEGIN ATOMIC:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Exemple:

```
CREATE FUNCTION [dbo].[udfMultiply]( @v1 int, @v2 int )
RETURNS bigint
WITH NATIVE_COMPILATION, SCHEMABINDING
AS
BEGIN ATOMIC WITH (TRANSACTION ISOLATION LEVEL = SNAPSHOT, LANGUAGE = N'English')

    DECLARE @ReturnValue bigint;
    SET @ReturnValue = @v1 * @v2;

    RETURN (@ReturnValue);
END

-- usage sample:
SELECT dbo.udfMultiply(10, 12)
```

Fonction de valeur de table inline native

La fonction de valeur de table compilée native renvoie la table en tant que résultat. Le code dans la fonction compilée nativement sera transformé en code C et compilé en dll. Seules les fonctions de table en ligne sont prises en charge dans la version 2016. Pour créer une fonction de valeur de table native, vous devez:

- Utiliser la syntaxe standard CREATE FUNCTION
- Définissez l'option NATIVE_COMPILATION dans la définition de fonction
- Utilisez l'option SCHEMABINDING dans la définition de fonction

Au lieu du bloc standard BEGIN END, vous devez utiliser le bloc BEGIN ATOMIC:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Exemple:

```
CREATE FUNCTION [dbo].[udft_NativeGetBusinessDoc]
(
   @RunDate VARCHAR(25)
)
RETURNS TABLE
WITH SCHEMABINDING,
     NATIVE_COMPILATION
```

```
AS
     RETURN
(
    SELECT BusinessDocNo,
           ProductCode,
           UnitID,
           ReasonID,
           PriceID,
           RunDate,
           ReturnPercent,
           Qty,
           RewardAmount,
           ModifyDate,
           UserID
    FROM dbo.[BusinessDocDetail_11]
    WHERE RunDate >= @RunDate
);
```

Lire Modules compilés nativement (Hekaton) en ligne: https://riptutorial.com/fr/sql-server/topic/6089/modules-compiles-nativement--hekaton-

Chapitre 68: Niveaux d'isolation des transactions

Syntaxe

- SET ISOLATION LEVEL {READ UNCOMMITTED | LIRE ENGAGÉ | REPEATABLE READ | SNAPSHOT | SERIALIZABLE} [;]

Remarques

Référence MSDN: SET NIVEAU D'ISOLATION DE TRANSACTION

Examples

Lire non engagé

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

C'est le niveau d'isolement le plus permissif, en ce sens qu'il ne provoque aucun blocage. Il spécifie que les instructions peuvent lire toutes les lignes, y compris celles qui ont été écrites dans des transactions mais pas encore validées (c'est-à-dire qu'elles sont toujours en transaction). Ce niveau d'isolement peut être sujet à des "lectures incorrectes".

Lire commise

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

Ce niveau d'isolement est le deuxième plus permissif. Il empêche les lectures sales. Le comportement de `READ COMMITTED` dépend du paramétrage de `READ_COMMITTED_SNAPSHOT` :

- Si cette option est définie sur OFF (paramètre par défaut), la transaction utilise des verrous partagés pour empêcher les autres transactions de modifier les lignes utilisées par la transaction en cours et pour empêcher la transaction en cours de lire les lignes modifiées par d'autres transactions.
- S'il est défini sur ON, l' `READCOMMITTEDLOCK` table `READCOMMITTEDLOCK` peut être utilisé pour demander un verrouillage partagé au lieu du contrôle de version de ligne pour les transactions exécutées en mode `READ COMMITTED` .

Remarque: `READ COMMITTED` est le comportement par défaut de SQL Server.

Que sont les "lectures sales"?

Les lectures sales (ou les lectures non validées) sont des lectures de lignes qui sont modifiées par une transaction ouverte.

Ce comportement peut être répliqué à l'aide de deux requêtes distinctes: une pour ouvrir une transaction et écrire des données dans une table sans être validée, l'autre pour sélectionner les données à écrire (mais pas encore validées) avec ce niveau d'isolation.

Requête 1 - Préparez une transaction mais ne la terminez pas:

```
CREATE TABLE dbo.demo (
    col1 INT,
    col2 VARCHAR(255)
);
GO
--This row will get committed normally:
BEGIN TRANSACTION;
    INSERT INTO dbo.demo(col1, col2)
    VALUES (99, 'Normal transaction');
COMMIT TRANSACTION;
--This row will be "stuck" in an open transaction, causing a dirty read
BEGIN TRANSACTION;
    INSERT INTO dbo.demo(col1, col2)
    VALUES (42, 'Dirty read');
--Do not COMMIT TRANSACTION or ROLLBACK TRANSACTION here
```

Requête 2 - Lisez les lignes, y compris la transaction ouverte:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM dbo.demo;
```

Résultats:

```
col1        col2
----------- ---------------------------------------
99          Normal transaction
42          Dirty read
```

PS: n'oubliez pas de nettoyer ces données de démonstration:

```
COMMIT TRANSACTION;
DROP TABLE dbo.demo;
GO
```

Répétable Lire

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

Ce niveau d'isolation de transaction est légèrement moins permissif que `READ COMMITTED` , dans la

mesure où les verrous partagés sont placés sur toutes les données lues par chaque instruction de la transaction et sont conservés **jusqu'à la fin de la transaction** .

Remarque: utilisez cette option uniquement lorsque cela est nécessaire, car il est plus susceptible de provoquer une dégradation des performances de la base de données et des blocages que `READ COMMITTED` .

Instantané

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
```

Indique que les données lues par une instruction dans une transaction seront la version cohérente sur le plan des transactions des données qui existaient au début de la transaction, c.-à-d. Qu'elles ne liront que les données validées avant le début de la transaction.

`SNAPSHOT` transactions `SNAPSHOT` ne demandent ni ne provoquent de verrous sur les données en cours de lecture, car elles ne lisent que la version (ou l'instantané) des données qui existaient au début de la transaction.

Une transaction s'exécutant au niveau d'isolement `SNAPSHOT` , en lecture seule, ses propres modifications de données en cours d'exécution. Par exemple, une transaction peut mettre à jour certaines lignes, puis lire les lignes mises à jour, mais cette modification ne sera visible que par la transaction en cours tant qu'elle n'est pas validée.

Remarque: L'option de base de données `ALLOW_SNAPSHOT_ISOLATION` doit être définie sur ON avant que le niveau d'isolement `SNAPSHOT` puisse être utilisé.

Sérialisable

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SERIALIZEABLE
```

Ce niveau d'isolement est le plus restrictif. Il demande une **plage verrouille** la plage des valeurs de clé lues par chaque instruction dans la transaction. Cela signifie également que les instructions `INSERT` des autres transactions seront bloquées si les lignes à insérer se trouvent dans la plage verrouillée par la transaction en cours.

Cette option a le même effet que de définir `HOLDLOCK` sur toutes les tables de toutes les `SELECT` d'une transaction.

Remarque: Cet isolement de transaction a la plus faible concurrence et ne doit être utilisé que lorsque cela est nécessaire.

Lire Niveaux d'isolation des transactions en ligne: https://riptutorial.com/fr/sql-

server/topic/5114/niveaux-d-isolation-des-transactions

Chapitre 69: Niveaux d'isolement et verrouillage

Remarques

J'ai trouvé ce lien - il est utile comme référence: "Niveaux d'isolement"

Examples

Exemples de définition du niveau d'isolement

Exemple de définition du niveau d'isolement:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM Products WHERE ProductId=1;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ; --return to the default one
```

1. `READ UNCOMMITTED` - signifie qu'une requête dans la transaction en cours ne peut pas accéder aux données modifiées d'une autre transaction qui n'est pas encore validée - pas de lecture incorrecte! MAIS, des lectures et des lectures fantômes non reproductibles sont possibles, car les données peuvent toujours être modifiées par d'autres transactions.

2. `REPEATABLE READ` - signifie qu'une requête dans la transaction en cours ne peut pas accéder aux données modifiées d'une autre transaction qui n'est pas encore `REPEATABLE READ` - pas de `REPEATABLE READ` ! Aucune autre transaction ne peut modifier les données lues par la transaction en cours tant qu'elle n'est pas terminée, ce qui élimine les lectures NON REPEATABLE. MAIS, si une autre transaction insère NEW ROWS et que la requête est exécutée plus d'une fois, les lignes fantômes peuvent apparaître à partir de la seconde lecture (si elle correspond à l'instruction where de la requête).

3. `SNAPSHOT` - ne peut renvoyer que des données qui existent au début de la requête. Assure la cohérence des données. Il empêche les lectures sales, les lectures non répétables et les lectures fantômes. Pour l'utiliser, la configuration de la base de données est requise:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

4. `READ COMMITTED` - isolation par défaut du serveur SQL. Cela empêche la lecture des données modifiées par une autre transaction jusqu'à leur validation. Il utilise le verrouillage partagé et le contrôle de version des lignes sur les tables, ce qui évite les lectures anormales. Cela dépend de la configuration de la base de données READ_COMMITTED_SNAPSHOT - si elle est activée - le contrôle de version des lignes est utilisé. pour activer - utilisez ceci:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;
```

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED; --return to the default one
```

5. `SERIALIZABLE` - utilise des verrous physiques acquis et conservés jusqu'à la fin de la transaction, ce qui empêche les lectures incorrectes, les lectures fantômes, les lectures non reproductibles. MAIS, cela a un impact sur les performances de la base de données, car les transactions simultanées sont sérialisées et exécutées une par une.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
```

Lire Niveaux d'isolement et verrouillage en ligne: https://riptutorial.com/fr/sql-server/topic/5331/niveaux-d-isolement-et-verrouillage

Chapitre 70: Noms d'alias dans Sql Server

Introduction

Voici quelques façons différentes de fournir des noms d'alias aux colonnes de Sql Server

Examples

En utilisant AS

C'est la méthode ANSI SQL qui fonctionne dans tous les SGBDR. Approche largement utilisée.

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FirstName +' '+ LastName As FullName
FROM   AliasNameDemo
```

En utilisant =

C'est mon approche préférée. Rien ne concerne la performance juste un choix personnel. Cela rend le code propre. Vous pouvez voir les noms de colonne résultants facilement au lieu de faire défiler le code si vous avez une grande expression.

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FullName = FirstName +' '+ LastName
FROM   AliasNameDemo
```

Donner un alias après le nom de la table dérivée

C'est une approche étrange que la plupart des gens ne savent même pas qu'il existe.

```
CREATE TABLE AliasNameDemo(id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT *
FROM   (SELECT firstname + ' ' + lastname
        FROM   AliasNameDemo) a (fullname)
```

- **Démo**

Sans utiliser AS

Cette syntaxe sera similaire à l'utilisation du mot clé `AS` . Juste nous ne devons pas utiliser le mot-clé `AS`

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FirstName +' '+ LastName FullName
FROM   AliasNameDemo
```

Lire Noms d'alias dans Sql Server en ligne: https://riptutorial.com/fr/sql-server/topic/10784/noms-d-alias-dans-sql-server

Chapitre 71: NULL

Introduction

Dans SQL Server, `NULL` représente les données manquantes ou inconnues. Cela signifie que `NULL` n'est pas vraiment une valeur; c'est mieux décrit comme un espace réservé pour une valeur. C'est aussi la raison pour laquelle vous ne pouvez comparer `NULL` à aucune valeur, ni même à un autre `NULL` .

Remarques

SQL Server fournit d'autres méthodes pour gérer les `ISNULL()` `IS NOT NULL` , telles que `IS NULL` , `IS NOT NULL` , `ISNULL()` , `COALESCE()` et autres.

Examples

Comparaison NULL

`NULL` est un cas particulier en matière de comparaisons.

Supposons les données suivantes.

```
id someVal
 ----
 0 NULL
 1 1
 2 2
```

Avec une requête:

```
 SELECT id
 FROM table
 WHERE someVal = 1
```

retournerait l'identifiant `1`

```
 SELECT id
 FROM table
 WHERE someVal <> 1
```

retournerait l'identifiant `2`

```
 SELECT id
 FROM table
 WHERE someVal IS NULL
```

retournerait l'identifiant `0`

```
SELECT id
FROM table
WHERE someVal IS NOT NULL
```

renverrait les deux identifiants `1` et `2` .

Si vous souhaitez que les valeurs NULL soient "comptées" en tant que valeurs dans une comparaison `=` , `<>` , elles doivent d'abord être converties en un type de données dénombrable:

```
SELECT id
FROM table
WHERE ISNULL(someVal, -1) <> 1
```

OU

```
SELECT id
FROM table
WHERE someVal IS NULL OR someVal <> 1
```

renvoie `0` et `2`

Ou vous pouvez changer votre paramètre ANSI Null .

ANSI NULLS

À partir de MSDN

> Dans une future version de SQL Server, ANSI_NULLS sera toujours activé et toute application définissant explicitement l'option sur OFF générera une erreur. Évitez d'utiliser cette fonctionnalité dans les nouveaux travaux de développement et prévoyez de modifier les applications qui utilisent actuellement cette fonctionnalité.

`ANSI NULLS` étant défini sur off, permet une comparaison `=` / `<>` des valeurs NULL.

Compte tenu des données suivantes:

```
id someVal
 ----
 0 NULL
 1 1
 2 2
```

Et avec ANSI NULLS, cette requête:

```
SELECT id
FROM table
WHERE someVal = NULL
```

ne produirait aucun résultat. Cependant, la même requête, avec ANSI NULLS désactivé:

```
set ansi_nulls off

SELECT id
FROM table
WHERE someVal = NULL
```

Retournerait l'identifiant `0` .

ISNULL ()

La fonction `IsNull()` accepte deux paramètres et renvoie le deuxième paramètre si le premier est `null` .

Paramètres:

1. cocher l'expression. Toute expression de n'importe quel type de données.
2. valeur de remplacement. C'est la valeur qui serait renvoyée si l'expression de vérification est nulle. La valeur de remplacement doit être d'un type de données pouvant être implicitement converti dans le type de données de l'expression de contrôle.

La fonction `IsNull()` renvoie le même type de données que l'expression de vérification.

```
DECLARE @MyInt int -- All variables are null until they are set with values.

SELECT ISNULL(@MyInt, 3) -- Returns 3.
```

Voir aussi `COALESCE` , ci-dessus

Est nul / n'est pas nul

Étant donné que null n'est pas une valeur, vous ne pouvez pas utiliser d'opérateurs de comparaison avec des valeurs NULL.
Pour vérifier si une colonne ou une variable contient null, vous devez utiliser la valeur `is null` :

```
DECLARE @Date date = '2016-08-03'
```

L'instruction suivante sélectionne la valeur `6` , car toutes les comparaisons avec des valeurs nulles sont considérées comme fausses ou inconnues:

```
SELECT CASE WHEN @Date = NULL THEN 1
            WHEN @Date <> NULL THEN 2
            WHEN @Date > NULL THEN 3
            WHEN @Date < NULL THEN 4
            WHEN @Date IS NULL THEN 5
            WHEN @Date IS NOT NULL THEN 6
```

En définissant le contenu de la variable @Date sur `null` et réessayez, l'instruction suivante renverra `5` :

```
SET @Date = NULL -- Note that the '=' here is an assignment operator!
```

```
SELECT CASE WHEN @Date = NULL THEN 1
            WHEN @Date <> NULL THEN 2
            WHEN @Date > NULL THEN 3
            WHEN @Date < NULL THEN 4
            WHEN @Date IS NULL THEN 5
            WHEN @Date IS NOT NULL THEN 6
```

COALESCE ()

`COALESCE ()` Evalue les arguments dans l'ordre et retourne la valeur actuelle de la première expression qui n'est pas initialement évaluée à `NULL` .

```
DECLARE @MyInt int -- variable is null until it is set with value.
DECLARE @MyInt2 int -- variable is null until it is set with value.
DECLARE @MyInt3 int -- variable is null until it is set with value.

SET @MyInt3  = 3

SELECT COALESCE (@MyInt, @MyInt2 ,@MyInt3 ,5) -- Returns 3 : value of @MyInt3.
```

Bien qu'ISNULL () fonctionne de manière similaire à COALESCE (), la fonction ISNULL () n'accepte que deux paramètres - un à vérifier et l'autre à utiliser si le premier paramètre est NULL. Voir aussi `ISNULL` , ci-dessous

NULL avec NOT IN SubQuery

Tout en ne manipulant pas la sous-requête avec la valeur NULL dans la sous-requête, nous devons éliminer NULLS pour obtenir les résultats attendus.

```
create table #outertable (i int)
create table #innertable (i int)

insert into #outertable (i) values (1), (2),(3),(4), (5)
insert into #innertable (i) values (2), (3), (null)

select * from #outertable where i in (select i from #innertable)
--2
--3
--So far so good

select * from #outertable where i not in (select i from #innertable)
--Expectation here is to get 1,4,5 but it is not. It will get empty results because of the
NULL it executes as {select * from #outertable where i not in (null)}

--To fix this
select * from #outertable where i not in (select i from #innertable where i is not null)
--you will get expected results
--1
--4
--5
```

Bien que la gestion de la sous-requête avec null soit prudente avec votre sortie attendue

Lire NULL en ligne: https://riptutorial.com/fr/sql-server/topic/5044/null

Chapitre 72: OLTP en mémoire (Hekaton)

Examples

Créer une table optimisée pour la mémoire

```
-- Create demo database
CREATE DATABASE SQL2016_Demo
 ON  PRIMARY
(
    NAME = N'SQL2016_Demo',
    FILENAME = N'C:\Dump\SQL2016_Demo.mdf',
    SIZE = 5120KB,
    FILEGROWTH = 1024KB
 )
 LOG ON
 (
    NAME = N'SQL2016_Demo_log',
    FILENAME = N'C:\Dump\SQL2016_Demo_log.ldf',
    SIZE = 1024KB,
    FILEGROWTH = 10%
 )
GO

use SQL2016_Demo
go

-- Add Filegroup by MEMORY_OPTIMIZED_DATA type
ALTER DATABASE SQL2016_Demo
    ADD FILEGROUP MemFG CONTAINS MEMORY_OPTIMIZED_DATA
GO


--Add a file to defined filegroup
ALTER DATABASE SQL2016_Demo ADD FILE
    (
        NAME = MemFG_File1,
        FILENAME = N'C:\Dump\MemFG_File1' -- your file path, check directory exist before
executing this code
    )
TO FILEGROUP MemFG
GO

--Object Explorer -- check database created
GO

-- create memory optimized table 1
CREATE TABLE dbo.MemOptTable1
(
    Column1     INT         NOT NULL,
    Column2     NVARCHAR(4000)  NULL,
    SpidFilter  SMALLINT    NOT NULL   DEFAULT (@@spid),

    INDEX ix_SpidFiler NONCLUSTERED (SpidFilter),
    INDEX ix_SpidFilter HASH (SpidFilter) WITH (BUCKET_COUNT = 64),

    CONSTRAINT CHK_soSessionC_SpidFilter
```

```
        CHECK ( SpidFilter = @@spid ),
)
    WITH
        (MEMORY_OPTIMIZED = ON,
         DURABILITY = SCHEMA_AND_DATA);  --or DURABILITY = SCHEMA_ONLY
go

-- create memory optimized table 2
CREATE TABLE MemOptTable2
(
    ID INT NOT NULL PRIMARY KEY NONCLUSTERED HASH WITH (BUCKET_COUNT = 10000),
    FullName NVARCHAR(200) NOT NULL,
    DateAdded DATETIME NOT NULL
) WITH (MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA)
GO
```

Afficher les fichiers et les tables .dll créés pour les tables optimisées pour la mémoire

```
SELECT
    OBJECT_ID('MemOptTable1') AS MemOptTable1_ObjectID,
    OBJECT_ID('MemOptTable2') AS MemOptTable2_ObjectID
GO

SELECT
    name,description
FROM sys.dm_os_loaded_modules
WHERE name LIKE '%XTP%'
GO
```

Afficher toutes les tables optimisées pour la mémoire:

```
SELECT
    name,type_desc,durability_desc,Is_memory_Optimized
FROM sys.tables
    WHERE Is_memory_Optimized = 1
GO
```

Types de tables et tables temporaires optimisés en mémoire

Par exemple, il s'agit d'un type de table traditionnel basé sur tempdb:

```
CREATE TYPE dbo.testTableType AS TABLE
(
   col1 INT NOT NULL,
   col2 CHAR(10)
);
```

Pour optimiser ce type de table en mémoire, ajoutez simplement l'option `memory_optimized=on` et ajoutez un index s'il n'y en a aucun sur le type d'origine:

```
CREATE TYPE dbo.testTableType AS TABLE
(
   col1 INT NOT NULL,
```

```
   col2 CHAR(10)
)WITH (MEMORY_OPTIMIZED=ON);
```

La table temporaire globale est comme ceci:

```
CREATE TABLE ##tempGlobalTabel
(
    Col1   INT   NOT NULL ,
    Col2   NVARCHAR(4000)
);
```

Table temporaire globale optimisée pour la mémoire:

```
CREATE TABLE dbo.tempGlobalTabel
(
    Col1   INT   NOT NULL   INDEX ix NONCLUSTERED,
    Col2   NVARCHAR(4000)
)
    WITH
        (MEMORY_OPTIMIZED = ON,
         DURABILITY = SCHEMA_ONLY);
```

Pour optimiser en mémoire les tables temporaires globales (## temp):

1. Créez une nouvelle table optimisée pour la mémoire `SCHEMA_ONLY` avec le même schéma que la table globale `##temp`
    - S'assurer que la nouvelle table a au moins un index
2. Modifiez toutes les références à `##temp` dans vos instructions Transact-SQL à la nouvelle température de la table optimisée pour la mémoire.
3. Remplacez les instructions `DROP TABLE ##temp` dans votre code par `DELETE FROM temp` , pour nettoyer le contenu
4. Supprimez les instructions `CREATE TABLE ##temp` de votre code - elles sont désormais redondantes

plus d'informations

Déclarez les variables de table optimisées par la mémoire

Pour des performances plus rapides, vous pouvez optimiser en mémoire votre variable de table. Voici le T-SQL pour une variable de table traditionnelle:

```
DECLARE @tvp TABLE
(
    col1   INT NOT NULL ,
    Col2   CHAR(10)
);
```

Pour définir des variables optimisées en mémoire, vous devez d'abord créer un type de table optimisé en mémoire, puis en déclarer une variable:

```
CREATE TYPE dbo.memTypeTable
```

```
AS TABLE
(
    Col1  INT NOT NULL INDEX ix1,
    Col2  CHAR(10)
)
WITH
    (MEMORY_OPTIMIZED = ON);
```

Ensuite, nous pouvons utiliser le type de table comme ceci:

```
DECLARE @tvp memTypeTable
insert INTO @tvp
values (1,'1'),(2,'2'),(3,'3'),(4,'4'),(5,'5'),(6,'6')

SELECT * FROM @tvp
```

Résultat:

```
Col1    Col2
1       1
2       2
3       3
4       4
5       5
6       6
```

Créer une table temporelle avec version du système optimisée pour la mémoire

```
CREATE TABLE [dbo].[MemOptimizedTemporalTable]
(
    [BusinessDocNo] [bigint] NOT NULL,
    [ProductCode] [int] NOT NULL,
    [UnitID] [tinyint] NOT NULL,
    [PriceID] [tinyint] NOT NULL,
    [SysStartTime] [datetime2](7) GENERATED ALWAYS AS ROW START NOT NULL,
    [SysEndTime] [datetime2](7) GENERATED ALWAYS AS ROW END NOT NULL,
    PERIOD FOR SYSTEM_TIME ([SysStartTime], [SysEndTime]),

    CONSTRAINT [PK_MemOptimizedTemporalTable]  PRIMARY KEY NONCLUSTERED
    (
        [BusinessDocNo] ASC,
        [ProductCode] ASC
    )
)
WITH (
    MEMORY_OPTIMIZED = ON , DURABILITY = SCHEMA_AND_DATA, -- Memory Optimized Option ON
    SYSTEM_VERSIONING = ON (HISTORY_TABLE = [dbo].[MemOptimizedTemporalTable_History] ,
DATA_CONSISTENCY_CHECK = ON )
)
```

plus d'informations

Lire OLTP en mémoire (Hekaton) en ligne: https://riptutorial.com/fr/sql-server/topic/5295/oltp-en-memoire--hekaton-

Chapitre 73: OPENJSON

Examples

Get key: paires de valeurs à partir du texte JSON

La fonction OPENJSON analyse le texte JSON et renvoie toutes les paires clé-valeur au premier niveau de JSON:

```
declare @json NVARCHAR(4000) = N'{"Name":"Joe","age":27,"skills":["C#","SQL"]}';
SELECT * FROM OPENJSON(@json);
```

clé	valeur	type
prénom	Joe	1
âge	27	2
compétences	["C #", "SQL"]	4

Le type de colonne décrit le type de valeur, à savoir null (0), string (1), number (2), boolean (3), array (4) et object (5).

Transformer le tableau JSON en un ensemble de lignes

La fonction OPENJSON analyse la collection d'objets JSON et renvoie des valeurs à partir du texte JSON sous la forme d'un ensemble de lignes.

```
declare @json nvarchar(4000) = N'[
  {"Number":"SO43659","Date":"2011-05-31T00:00:00","Customer":
"MSFT","Price":59.99,"Quantity":1},
  {"Number":"SO43661","Date":"2011-06-
01T00:00:00","Customer":"Nokia","Price":24.99,"Quantity":3}
]'

SELECT    *
FROM OPENJSON (@json)
    WITH (
          Number   varchar(200),
          Date     datetime,
          Customer varchar(200),
          Quantity int
  )
```

Dans la clause WITH est spécifié le schéma de retour de la fonction OPENJSON. Les clés des objets JSON sont extraites par les noms de colonne. Si une clé dans JSON n'est pas spécifiée dans la clause WITH (par exemple Price dans cet exemple), elle sera ignorée. Les valeurs sont automatiquement converties en types spécifiés.

Nombre	Rendez-vous amoureux	Client	Quantité
SO43659	2011-05-31T00: 00: 00	MSFT	1
SO43661	2011-06-01T00: 00: 00	Nokia	3

Transformer des champs JSON imbriqués en un ensemble de lignes

La fonction OPENJSON analyse la collection d'objets JSON et renvoie des valeurs à partir du texte JSON sous la forme d'un ensemble de lignes. Si les valeurs de l'objet d'entrée sont imbriquées, un paramètre de mappage supplémentaire peut être spécifié dans chaque colonne de la clause WITH:

```
declare @json nvarchar(4000) = N'[
  {"data":{"num":"SO43659","date":"2011-05-
31T00:00:00"},"info":{"customer":"MSFT","Price":59.99,"qty":1}},
  {"data":{"number":"SO43661","date":"2011-06-
01T00:00:00"},"info":{"customer":"Nokia","Price":24.99,"qty":3}}
]'

SELECT    *
FROM OPENJSON (@json)
    WITH (
          Number   varchar(200) '$.data.num',
          Date     datetime '$.data.date',
          Customer varchar(200) '$.info.customer',
          Quantity int '$.info.qty',
  )
```

Dans la clause WITH est spécifié le schéma de retour de la fonction OPENJSON. Après le type est spécifié chemin d'accès aux noeuds JSON où la valeur renvoyée doit être trouvée. Les clés des objets JSON sont extraites par ces chemins. Les valeurs sont automatiquement converties en types spécifiés.

Nombre	Rendez-vous amoureux	Client	Quantité
SO43659	2011-05-31T00: 00: 00	MSFT	1
SO43661	2011-06-01T00: 00: 00	Nokia	3

Extraire des sous-objets JSON internes

OPENJSON peut extraire des fragments d'objets JSON dans le texte JSON. Dans la définition de colonne qui référence le sous-objet JSON, définissez les options de type nvarchar (max) et AS JSON:

```
declare @json nvarchar(4000) = N'[
  {"Number":"SO43659","Date":"2011-05-
31T00:00:00","info":{"customer":"MSFT","Price":59.99,"qty":1}},
  {"Number":"SO43661","Date":"2011-06-
01T00:00:00","info":{"customer":"Nokia","Price":24.99,"qty":3}}
```

```
]'

SELECT    *
FROM OPENJSON (@json)
    WITH (
          Number   varchar(200),
          Date     datetime,
          Info nvarchar(max) '$.info' AS JSON
  )
```

La colonne Info sera mappée à l'objet "Info". Les résultats seront:

Nombre	Rendez-vous amoureux	Info
SO43659	2011-05-31T00: 00: 00	{"customer": "MSFT", "Price": 59.99, "qty": 1}
SO43661	2011-06-01T00: 00: 00	{"customer": "Nokia", "Price": 24,99, "qty": 3}

Travailler avec des sous-tableaux JSON imbriqués

JSON peut avoir une structure complexe avec des tableaux internes. Dans cet exemple, nous avons un tableau d'ordres avec des sous-tableaux imbriqués de OrderItems.

```
declare @json nvarchar(4000) = N'[
  {"Number":"SO43659","Date":"2011-05-31T00:00:00",
    "Items":[{"Price":11.99,"Quantity":1},{"Price":12.99,"Quantity":5}]},
  {"Number":"SO43661","Date":"2011-06-01T00:00:00",

"Items":[{"Price":21.99,"Quantity":3},{"Price":22.99,"Quantity":2},{"Price":23.99,"Quantity":2}]}

]'
```

Nous pouvons analyser les propriétés de niveau racine en utilisant OPENJSON qui renverra le fragment AS du tableau AS JSON. Ensuite, nous pouvons appliquer OPENJSON à nouveau sur le tableau Items et ouvrir la table JSON interne. La table de premier niveau et la table interne seront "jointes" comme dans la jointure entre les tables standard:

```
SELECT    *
FROM
    OPENJSON (@json)
    WITH (  Number varchar(200), Date datetime,
            Items nvarchar(max) AS JSON )
        CROSS APPLY
            OPENJSON (Items)
            WITH ( Price float, Quantity int)
```

Résultats:

Nombre	Rendez-vous amoureux	Articles	Prix	Quantité
SO43659	2011-05-31 00:	[{"Prix": 11.99, "Quantité": 1}, {"Prix":	11.99	1

Nombre	Rendez-vous amoureux	Articles	Prix	Quantité
	00: 00.000	12.99, "Quantité": 5}]		
SO43659	2011-05-31 00: 00: 00.000	[{"Prix": 11.99, "Quantité": 1}, {"Prix": 12.99, "Quantité": 5}]	12,99	5
SO43661	2011-06-01 00: 00: 00.000	[{"Prix": 21.99, "Quantity": 3}, {"Price": 22.99, "Quantity": 2}, {"Price": 23.99, "Quantity": 2}]	21,99	3
SO43661	2011-06-01 00: 00: 00.000	[{"Prix": 21.99, "Quantity": 3}, {"Price": 22.99, "Quantity": 2}, {"Price": 23.99, "Quantity": 2}]	22.99	2
SO43661	2011-06-01 00: 00: 00.000	[{"Prix": 21.99, "Quantity": 3}, {"Price": 22.99, "Quantity": 2}, {"Price": 23.99, "Quantity": 2}]	23,99	2

Lire OPENJSON en ligne: https://riptutorial.com/fr/sql-server/topic/5030/openjson

Chapitre 74: Opérations de base DDL dans MS SQL Server

Examples

Commencer

Cette section décrit une base **DDL** (= « **D** ata **D** ÉFINITION **L** angue ») commande pour créer une base de données, une table dans une base de données, une vue et enfin une procédure stockée.

Créer une base de données

La commande SQL suivante crée une nouvelle base de données `Northwind` sur le serveur actuel, à l'aide du chemin d'accès `C:\Program Files\Microsoft SQL Server\MSSQL11.INSTSQL2012\MSSQL\DATA\` :

```
USE [master]
GO

CREATE DATABASE [Northwind]
 CONTAINMENT = NONE
 ON  PRIMARY
 (
  NAME = N'Northwind',
  FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.INSTSQL2012\MSSQL\DATA\Northwind.mdf' , SIZE = 5120KB , MAXSIZE = UNLIMITED,
FILEGROWTH = 1024KB
 )
 LOG ON
 (
  NAME = N'Northwind_log',
  FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL11.INSTSQL2012\MSSQL\DATA\Northwind_log.ldf' , SIZE = 1536KB , MAXSIZE = 2048GB ,
FILEGROWTH = 10%
 )
GO

ALTER DATABASE [Northwind] SET COMPATIBILITY_LEVEL = 110
GO
```

Remarque: Une base de données T-SQL se compose de deux fichiers, le fichier de base de données `*.mdf` et son journal de transactions `*.ldf` . Les deux doivent être spécifiés lors de la création d'une nouvelle base de données.

Créer une table

La commande SQL suivante crée une nouvelle table `Categories` dans la base de données en

cours, à l'aide du schéma `dbo` (vous pouvez changer de contexte de base de données avec `Use <DatabaseName>`):

```
CREATE TABLE dbo.Categories(
    CategoryID int IDENTITY NOT NULL,
    CategoryName nvarchar(15) NOT NULL,
    Description ntext NULL,
    Picture image NULL,
      CONSTRAINT PK_Categories PRIMARY KEY CLUSTERED
      (
         CategoryID ASC
      )
      WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
            ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON PRIMARY
) ON PRIMARY TEXTIMAGE_ON PRIMARY
```

Créer une vue

La commande SQL suivante crée une nouvelle vue `Summary_of_Sales_by_Year` dans la base de données en cours, à l'aide du schéma `dbo` (vous pouvez changer le contexte de la base de données avec `Use <DatabaseName>`):

```
CREATE VIEW dbo.Summary_of_Sales_by_Year AS
  SELECT ord.ShippedDate, ord.OrderID, ordSub.Subtotal
  FROM Orders ord
  INNER JOIN [Order Subtotals] ordSub ON ord.OrderID = ordSub.OrderID
```

Cela rejoindra les tables `Orders` et `[Order Subtotals]` pour afficher les colonnes `ShippedDate` , `OrderID` et `Subtotal` . Comme la table `[Order Subtotals]` a un nom vide dans la base de données Northwind, elle doit être placée entre crochets.

Créer une procédure

La commande SQL suivante crée une nouvelle procédure stockée `CustOrdersDetail` dans la base de données en cours, à l'aide du schéma `dbo` (vous pouvez changer de contexte de base de données avec `Use <DatabaseName>`):

```
CREATE PROCEDURE dbo.MyCustOrdersDetail @OrderID int, @MinQuantity int=0
AS BEGIN
  SELECT ProductName,
    UnitPrice=ROUND(Od.UnitPrice, 2),
    Quantity,
    Discount=CONVERT(int, Discount * 100),
    ExtendedPrice=ROUND(CONVERT(money, Quantity * (1 - Discount) * Od.UnitPrice), 2)
  FROM Products P, [Order Details] Od
  WHERE Od.ProductID = P.ProductID and Od.OrderID = @OrderID
  and Od.Quantity>=@MinQuantity
END
```

Cette procédure stockée, après sa création, peut être appelée comme suit:

```
exec dbo.MyCustOrdersDetail 10248
```

qui retournera tous les détails de la commande avec @ OrderId = 10248 (et la quantité> = 0 par défaut). Ou vous pouvez spécifier le paramètre facultatif

```
exec dbo.MyCustOrdersDetail 10248, 10
```

qui ne renverra que les commandes avec une quantité minimum de 10 (ou plus).

Lire Opérations de base DDL dans MS SQL Server en ligne: https://riptutorial.com/fr/sql-server/topic/5463/operations-de-base-ddl-dans-ms-sql-server

Chapitre 75: Options avancées

Examples

Activer et afficher les options avancées

```
Exec sp_configure 'show advanced options' ,1
RECONFIGURE
GO
-- Show all configure
sp_configure
```

Activer la compression par défaut

```
Exec sp_configure 'backup compression default',1
GO
RECONFIGURE;
```

Définir le facteur de remplissage par défaut

```
sp_configure 'fill factor', 100;
GO
RECONFIGURE;
```

Le serveur doit être redémarré avant que la modification puisse prendre effet.

Définir l'intervalle de récupération du système

```
USE master;
GO
-- Set recovery every 3 min
EXEC sp_configure 'recovery interval', '3';
RECONFIGURE WITH OVERRIDE;
```

Activer l'autorisation cmd

```
EXEC sp_configure 'xp_cmdshell', 1
GO
RECONFIGURE
```

Définir la taille maximale de la mémoire du serveur

```
USE master
EXEC sp_configure 'max server memory (MB)', 64
RECONFIGURE WITH OVERRIDE
```

Définir le nombre de tâches de point de contrôle

```
EXEC sp_configure "number of checkpoint tasks", 4
```

Lire Options avancées en ligne: https://riptutorial.com/fr/sql-server/topic/5185/options-avancees

Chapitre 76: Pagination

Introduction

Décalage de lignes et pagination dans différentes versions de SQL Server

Syntaxe

- SELECT * FROM TableName ORDER BY id OFFSET 10 ROWS FETCH NEXT 10 ROWS SEULEMENT;

Examples

Pagination utilisant ROW_NUMBER avec une expression de table commune

SQL Server 2008

La fonction `ROW_NUMBER` peut attribuer un numéro incrémentiel à chaque ligne d'un jeu de résultats. Combiné à une expression de table commune qui utilise un opérateur `BETWEEN` , il est possible de créer des "pages" de jeux de résultats. Par exemple: la première page contenant les résultats 1-10, la deuxième page contenant les résultats 11-20, la troisième page contenant les résultats 21-30, etc.

```
WITH data
AS
(
    SELECT ROW_NUMBER() OVER (ORDER BY name) AS row_id,
        object_id,
        name,
        type,
        create_date
    FROM sys.objects
)
SELECT *
FROM data
WHERE row_id BETWEEN 41 AND 50
```

Note: Il n'est pas possible d'utiliser `ROW_NUMBER` dans une clause `WHERE` comme:

```
SELECT object_id,
    name,
    type,
    create_date
FROM sys.objects
WHERE ROW_NUMBER() OVER (ORDER BY name) BETWEEN 41 AND 50
```

Bien que cela soit plus pratique, SQL Server renverra l'erreur suivante dans ce cas:

Msg 4108, niveau 15, état 1, ligne 6

Les fonctions fenêtrées ne peuvent apparaître que dans les clauses SELECT ou ORDER BY.

Pagination avec OFFSET FETCH

SQL Server 2012

La clause `OFFSET FETCH` implémente la pagination de manière plus concise. Avec elle, il est possible de sauter les lignes N1 (spécifiées dans `OFFSET`) et de renvoyer les N2 lignes suivantes (spécifiées dans `FETCH`):

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 40 ROWS FETCH NEXT 10 ROWS ONLY
```

La clause `ORDER BY` est requise pour fournir des résultats déterministes.

Paginaton avec requête interne

Dans les versions antérieures de SQL Server, les développeurs devaient utiliser le double tri associé au mot clé `TOP` pour renvoyer des lignes dans une page:

```
 SELECT TOP 10 *
 FROM
 (
    SELECT
    TOP 50 object_id,
        name,
        type,
        create_date
    FROM sys.objects
    ORDER BY name ASC
) AS data
ORDER BY name DESC
```

La requête interne renverra les 50 premières lignes classées par `name` . Ensuite, la requête externe inversera l'ordre de ces 50 lignes et sélectionnera les 10 premières lignes (il s'agira des 10 dernières lignes du groupe avant l'inversion).

Paging dans différentes versions de SQL Server

SQL Server 2012/2014

```
DECLARE @RowsPerPage INT = 10, @PageNumber INT = 4
SELECT OrderId, ProductId
FROM OrderDetail
```

```
ORDER BY OrderId
OFFSET (@PageNumber - 1) * @RowsPerPage ROWS
FETCH NEXT @RowsPerPage ROWS ONLY
```

SQL Server 2005/2008 / R2

```
DECLARE @RowsPerPage INT = 10, @PageNumber INT = 4
SELECT OrderId, ProductId
FROM (
    SELECT OrderId, ProductId, ROW_NUMBER() OVER (ORDER BY OrderId) AS RowNum
    FROM OrderDetail) AS OD
WHERE OD.RowNum BETWEEN ((@PageNumber - 1 ) * @RowsPerPage) + 1
AND @RowsPerPage * @PageNumber
```

SQL Server 2000

```
DECLARE @RowsPerPage INT = 10, @PageNumber INT = 4
SELECT OrderId, ProductId
FROM (SELECT TOP (@RowsPerPage) OrderId, ProductId
        FROM (SELECT TOP ((@PageNumber)*@RowsPerPage) OrderId, ProductId
                FROM OrderDetail
                ORDER BY OrderId) AS OD
    ORDER BY OrderId DESC) AS OD2
ORDER BY OrderId ASC
```

SQL Server 2012/2014 utilisant ORDER BY OFFSET et FETCH NEXT

Pour obtenir les 10 lignes suivantes, lancez cette requête:

```
SELECT * FROM TableName ORDER BY id OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

Points clés à considérer lors de son utilisation:

- `ORDER BY` est obligatoire pour utiliser la clause `OFFSET` et `FETCH` .
- `OFFSET` clause `OFFSET` est obligatoire avec `FETCH` . Vous ne pouvez jamais utiliser, `ORDER BY` … `FETCH` .
- `TOP` ne peut pas être combiné avec `OFFSET` et `FETCH` dans la même expression de requête.

Lire Pagination en ligne: https://riptutorial.com/fr/sql-server/topic/6874/pagination

Chapitre 77: PAR GROUPE

Examples

Groupement simple

Tableau des commandes

N ° de client	ProductId	Quantité	Prix
1	2	5	100
1	3	2	200
1	4	1	500
2	1	4	50
3	5	6	700

Lors du regroupement par une colonne spécifique, seules les valeurs uniques de cette colonne sont renvoyées.

```
SELECT customerId
FROM orders
GROUP BY customerId;
```

Valeur de retour:

N ° de client
1
2
3

Les fonctions d'agrégat comme `count()` s'appliquent à chaque groupe et non à la table complète:

```
SELECT customerId,
       COUNT(productId) as numberOfProducts,
       sum(price) as totalPrice
FROM orders
GROUP BY customerId;
```

Valeur de retour:

N ° de client	nombreProduits	prix total
1	3	800
2	1	50
3	1	700

GROUP BY plusieurs colonnes

On peut vouloir grouper par plus d'une colonne

```
declare @temp table(age int, name varchar(15))

insert into @temp
select 18, 'matt' union all
select 21, 'matt' union all
select 21, 'matt' union all
select 18, 'luke' union all
select 18, 'luke' union all
select 21, 'luke' union all
select 18, 'luke' union all
select 21, 'luke'

SELECT Age, Name, count(1) count
FROM @temp
GROUP BY Age, Name
```

regroupera par âge et par nom et produira:

Âge	prénom	compter
18	Luke	3
21	Luke	2
18	mat	1
21	mat	2

Regrouper avec plusieurs tables, plusieurs colonnes

Group by est souvent utilisé avec une déclaration de jointure. Supposons que nous ayons deux tables. Le premier est la table des étudiants:

Id	Nom complet	Âge
1	Matt Jones	20
2	Frank Blue	21

Id	Nom complet	Âge
3	Anthony Angel	18

Le deuxième tableau est la table des matières que chaque élève peut prendre:

Subject_Id	Assujettir
1	Mathématiques
2	PE
3	La physique

Et comme un étudiant peut participer à de nombreuses matières et qu'un sujet peut être suivi par de nombreux étudiants (donc une relation N: N), nous devons avoir un troisième tableau "limite". Appelons la table Students_subjects:

Subject_Id	Carte d'étudiant
1	1
2	2
2	1
3	2
1	3
1	1

Maintenant, disons que nous voulons connaître le nombre de sujets que chaque étudiant fréquente. Ici, l'instruction `GROUP BY` autonome n'est pas suffisante car les informations ne sont pas disponibles via une seule table. Par conséquent, nous devons utiliser `GROUP BY` avec l'instruction `JOIN` :

```
Select Students.FullName, COUNT(Subject Id) as SubjectNumber FROM Students_Subjects
LEFT JOIN Students
ON Students_Subjects.Student_id = Students.Id
GROUP BY Students.FullName
```

Le résultat de la requête donnée est le suivant:

Nom complet	Numéro d'objet
Matt Jones	3
Frank Blue	2

Nom complet	Numéro d'objet
Anthony Angel	1

Pour un exemple encore plus complexe de l'utilisation de GROUP BY, supposons que l'élève puisse assigner le même sujet à son nom plus d'une fois (comme indiqué dans le tableau Students_Subjects). Dans ce scénario, nous pourrions être en mesure de compter le nombre de fois que chaque sujet a été attribué à un élève par GROUPing par plus d'une colonne:

```
SELECT Students.FullName, Subjects.Subject,
COUNT(Students_subjects.Subject_id) AS NumberOfOrders
FROM ((Students_Subjects
INNER JOIN Students
ON Students_Subjcets.Student_id=Students.Id)
INNER JOIN Subjects
ON Students_Subjects.Subject_id=Subjects.Subject_id)
GROUP BY Fullname,Subject
```

Cette requête donne le résultat suivant:

Nom complet	Assujettir	Numéro d'objet
Matt Jones	Mathématiques	2
Matt Jones	PE	1
Frank Blue	PE	1
Frank Blue	La physique	1
Anthony Angel	Mathématiques	1

AYANT

Étant donné que la clause `WHERE` est évaluée avant `GROUP BY` , vous ne pouvez pas utiliser `WHERE` pour réduire les résultats du regroupement (généralement une fonction d'agrégat, telle que `COUNT(*)`). Pour répondre à ce besoin, la clause `HAVING` peut être utilisée.

Par exemple, en utilisant les données suivantes:

```
DECLARE @orders TABLE(OrderID INT, Name NVARCHAR(100))

INSERT INTO @orders VALUES
( 1, 'Matt' ),
( 2, 'John' ),
( 3, 'Matt' ),
( 4, 'Luke' ),
( 5, 'John' ),
( 6, 'Luke' ),
( 7, 'John' ),
( 8, 'John' ),
( 9, 'Luke' ),
```

```
( 10, 'John' ),
( 11, 'Luke' )
```

Si nous voulons obtenir le nombre de commandes que chaque personne a passées, nous utiliserons

```
SELECT Name, COUNT(*) AS 'Orders'
FROM @orders
GROUP BY Name
```

et obtenir

prénom	Ordres
Mat	2
John	5
Luke	4

Cependant, si nous voulons limiter cela aux individus qui ont placé plus de deux ordres, nous pouvons ajouter une clause `HAVING` .

```
SELECT Name, COUNT(*) AS 'Orders'
FROM @orders
GROUP BY Name
HAVING COUNT(*) > 2
```

va céder

prénom	Ordres
John	5
Luke	4

Notez que, tout comme `GROUP BY` , les colonnes placées dans `HAVING` doivent correspondre exactement à leurs homologues dans l' `SELECT` . Si dans l'exemple ci-dessus, nous avions plutôt dit

```
SELECT Name, COUNT(DISTINCT OrderID)
```

notre clause `HAVING` devrait dire

```
HAVING COUNT(DISTINCT OrderID) > 2
```

GROUP BY avec ROLLUP et CUBE

L'opérateur ROLLUP est utile pour générer des rapports contenant des sous-totaux et des totaux.

- CUBE génère un jeu de résultats qui affiche des agrégats pour toutes les combinaisons de valeurs dans les colonnes sélectionnées.
- ROLLUP génère un jeu de résultats qui affiche des agrégats pour une hiérarchie de valeurs dans les colonnes sélectionnées.

Article	Couleur	Quantité
Table	Bleu	124
Table	rouge	223
chaise	Bleu	101
chaise	rouge	210

```
SELECT CASE WHEN (GROUPING(Item) = 1) THEN 'ALL'
            ELSE ISNULL(Item, 'UNKNOWN')
       END AS Item,
       CASE WHEN (GROUPING(Color) = 1) THEN 'ALL'
            ELSE ISNULL(Color, 'UNKNOWN')
       END AS Color,
       SUM(Quantity) AS QtySum
FROM Inventory
GROUP BY Item, Color WITH ROLLUP

Item                 Color                QtySum
-------------------- -------------------- --------------------------
Chair                Blue                 101.00
Chair                Red                  210.00
Chair                ALL                  311.00
Table                Blue                 124.00
Table                Red                  223.00
Table                ALL                  347.00
ALL                  ALL                  658.00
```

(7 rangs affectés)

Si le mot clé ROLLUP de la requête est remplacé par CUBE, le jeu de résultats CUBE est identique, sauf que ces deux lignes supplémentaires sont renvoyées à la fin:

```
ALL                  Blue                 225.00
ALL                  Red                  433.00
```

https://technet.microsoft.com/en-us/library/ms189305(v=sql.90).aspx

Lire PAR GROUPE en ligne: https://riptutorial.com/fr/sql-server/topic/3231/par-groupe

Chapitre 78: Paramètres de la table

Remarques

Les paramètres de valeur de table (TVP en abrégé) sont des paramètres transmis à une procédure stockée ou à une fonction contenant des données structurées dans une table. L'utilisation de paramètres à valeur de table nécessite la création d'un type de tableau défini par l'utilisateur pour le paramètre utilisé.

Les paramètres à valeur ajoutée sont des paramètres en lecture seule.

Examples

Utilisation d'un paramètre de table pour insérer plusieurs lignes dans une table

Tout d'abord, définissez un type de table défini à utiliser:

```
CREATE TYPE names as TABLE
(
    FirstName varchar(10),
    LastName varchar(10)
)
GO
```

Créez la procédure stockée:

```
CREATE PROCEDURE prInsertNames
(
    @Names dbo.Names READONLY -- Note: You must specify the READONLY
)
AS

INSERT INTO dbo.TblNames (FirstName, LastName)
SELECT FirstName, LastName
FROM @Names
GO
```

Exécution de la procédure stockée:

```
DECLARE @names dbo.Names
INSERT INTO @Names VALUES
('Zohar', 'Peled'),
('First', 'Last')

EXEC dbo.prInsertNames @Names
```

Lire Paramètres de la table en ligne: https://riptutorial.com/fr/sql-server/topic/5285/parametres-de-la-table

Chapitre 79: Parsename

Syntaxe

- PARSENAME ('nom_objet', object_piece)

Paramètres

'nom_objet'	object_piece
Nom de l'objet pour lequel récupérer la partie d'objet spécifiée. nom_objet est sysname. Ce paramètre est un nom d'objet éventuellement qualifié. Si toutes les parties du nom de l'objet sont qualifiées, ce nom peut comporter quatre parties: le nom du serveur, le nom de la base de données, le nom du propriétaire et le nom de l'objet.	La partie objet est-elle à retourner? object_piece est de type int et peut avoir ces valeurs: 1 = Nom de l'objet 2 = Nom du schéma 3 = Nom de la base de données 4 = Nom du serveur

Examples

PARSENAME

```
Declare @ObjectName nVarChar(1000)
Set @ObjectName = 'HeadOfficeSQL1.Northwind.dbo.Authors'

SELECT
 PARSENAME(@ObjectName, 4) as Server
,PARSENAME(@ObjectName, 3) as DB
,PARSENAME(@ObjectName, 2) as Owner
,PARSENAME(@ObjectName, 1) as Object
```

Résultats:

Serveur	DB
Siège socialSQL1	Vent du nord

Propriétaire	Objet
dbo	Auteurs

Lire Parsename en ligne: https://riptutorial.com/fr/sql-server/topic/5775/parsename

Chapitre 80: Partitionnement

Examples

Récupérer les valeurs limites des partitions

```
SELECT        ps.name AS PartitionScheme
            , fg.name AS [FileGroup]
            , prv.*
            , LAG(prv.Value) OVER (PARTITION BY ps.name ORDER BY ps.name, boundary_id) AS
PreviousBoundaryValue

FROM        sys.partition_schemes ps
INNER JOIN    sys.destination_data_spaces dds
            ON dds.partition_scheme_id = ps.data_space_id
INNER JOIN    sys.filegroups fg
            ON dds.data_space_id = fg.data_space_id
INNER JOIN    sys.partition_functions f
            ON f.function_id = ps.function_id
INNER JOIN    sys.partition_range_values prv
            ON f.function_id = prv.function_id
            AND dds.destination_id = prv.boundary_id
```

Changement de partition

Selon cette [page TechNet Microsoft] [1],

> **Les données de partitionnement** vous permettent de gérer et d'accéder à des sous-ensembles de vos données rapidement et efficacement, tout en conservant l'intégrité de l'ensemble de la collecte de données.

Lorsque vous appelez la requête suivante, les données ne sont pas déplacées physiquement. seules les métadonnées concernant l'emplacement des données sont modifiées.

```
ALTER TABLE [SourceTable] SWITCH TO [TargetTable]
```

Les tables doivent avoir les mêmes colonnes avec les mêmes types de données et paramètres NULL, elles doivent être dans le même groupe de fichiers et la nouvelle table cible doit être vide. Voir le lien de la page ci-dessus pour plus d'informations sur le changement de partitions.

[1]: https://technet.microsoft.com/en-us/library/ms191160(v=sql.105).aspx La propriété `IDENTITY` colonne peut être différente.

Récupère les valeurs de la table de partition, de la colonne, du schéma, de la fonction, du total et du min-max à l'aide d'une seule requête

```
SELECT DISTINCT
    object_name(i.object_id) AS [Object Name],
    c.name AS [Partition Column],
    s.name AS [Partition Scheme],
```

```
    pf.name AS [Partition Function],
    prv.tot AS [Partition Count],
    prv.miVal AS [Min Boundry Value],
    prv.maVal AS [Max Boundry Value]
FROM sys.objects o
INNER JOIN sys.indexes i ON i.object_id = o.object_id
INNER JOIN sys.columns c ON c.object_id = o.object_id
INNER JOIN sys.index_columns ic ON ic.object_id = o.object_id
    AND ic.column_id = c.column_id
    AND ic.partition_ordinal = 1
INNER JOIN sys.partition_schemes s ON i.data_space_id = s.data_space_id
INNER JOIN sys.partition_functions pf ON pf.function_id = s.function_id
OUTER APPLY(SELECT
                COUNT(*) tot, MIN(value) miVal, MAX(value) maVal
            FROM sys.partition_range_values prv
            WHERE prv.function_id = pf.function_id) prv
--WHERE object_name(i.object_id) = 'table_name'
ORDER BY OBJECT_NAME(i.object_id)
```

Juste Décommentez `where` l' article et remplacer `table_name` avec le `actual table name` la `actual table name` pour voir le détail de l' objet désiré.

Lire Partitionnement en ligne: https://riptutorial.com/fr/sql-server/topic/3212/partitionnement

Chapitre 81: PHANTOM lu

Introduction

Dans les systèmes de base de données, l'isolation détermine la manière dont l'intégrité des transactions est visible par les autres utilisateurs et systèmes. Elle définit donc comment et quand les modifications apportées par une opération deviennent visibles par les autres. La lecture fantôme peut se produire lorsque vous obtenez des données qui ne sont pas encore validées dans la base de données.

Remarques

Vous pouvez lire les différents `ISOLATION LEVEL` sur MSDN

Examples

Niveau d'isolement LISEZ ICI

Créer un exemple de table sur une base de données exemple

```
CREATE TABLE [dbo].[Table_1](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [title] [varchar](50) NULL,
 CONSTRAINT [PK_Table_1] PRIMARY KEY CLUSTERED
(
    [Id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
```

Maintenant, ouvrez un éditeur de première requête (sur la base de données) insérez le code ci-dessous et exécutez (**ne touchez pas le --rollback**) dans ce cas, vous insérez une ligne dans la base de données mais ne **validez pas les** modifications.

```
begin tran

INSERT INTO Table_1 values('Title 1')

SELECT * FROM [Test].[dbo].[Table_1]

--rollback
```

Maintenant, ouvrez un Second Query Editor (sur la base de données), insérez le code ci-dessous et exécutez

```
begin tran

set transaction isolation level READ UNCOMMITTED
```

```
SELECT * FROM [Test].[dbo].[Table_1]
```

Vous remarquerez peut-être que sur le second éditeur, vous pouvez voir la ligne nouvellement créée (mais non validée) de la première transaction. Sur le premier éditeur, exécutez la restauration (sélectionnez le mot de restauration et exécutez-le).

```
-- Rollback the first transaction
rollback
```

Exécutez la requête sur le second éditeur et vous voyez que l'enregistrement disparaît (lecture fantôme), cela se produit parce que vous dites à la 2ème transaction que vous obtenez toutes les lignes, y compris les non validées.

Cela se produit lorsque vous modifiez le niveau d'isolement avec

```
set transaction isolation level READ UNCOMMITTED
```

Lire PHANTOM lu en ligne: https://riptutorial.com/fr/sql-server/topic/8235/phantom-lu

Chapitre 82: PIVOT / UNPIVOT

Syntaxe

- SELECT `<non-pivoted column>` ,
  [première colonne pivotée] AS `<column name>` ,
  [deuxième colonne pivotée] AS `<column name>` ,
  ...
  [dernière colonne pivotée] AS `<column name>`
  DE
  (`<SELECT query that produces the data>`)
  AS `<alias for the source query>`
  PIVOT
  (
  `<aggregation function>` (`<column being aggregated>`)
  POUR
  [`<column that contains the values that will become column headers>`]
  IN ([première colonne pivotée], [deuxième colonne pivotée],
  ... [dernière colonne pivotée])
) AS `<alias for the pivot table>` `<optional ORDER BY clause>` ;

Remarques

A l'aide des opérateurs PIVOT et UNPIVOT, vous transformez une table en déplaçant les lignes (valeurs de colonne) d'une table vers des colonnes et inversement. Dans le cadre de cette transformation, les fonctions d'agrégation peuvent être appliquées aux valeurs de la table.

Examples

Pivot simple - colonnes statiques

À l'aide de la table des ventes d'articles de la base de données exemple , laissez-nous calculer et afficher la quantité totale que nous avons vendue pour chaque produit.

Cela peut être facilement fait avec un group by, mais supposons que nous "tournions" notre table de résultats de manière à ce que pour chaque ID de produit nous ayons une colonne.

```
SELECT [100], [145]
  FROM (SELECT ItemId , Quantity
          FROM #ItemSalesTable
       ) AS pivotIntermediate
 PIVOT (   SUM(Quantity)
           FOR ItemId IN ([100], [145])
       ) AS pivotTable
```

Puisque nos nouvelles colonnes sont des nombres (dans la table source), nous devons mettre entre crochets `[]`

Cela nous donnera une sortie comme

100	145
45	18

PIVOT simple et UNPIVOT (T-SQL)

Voici un exemple simple qui montre le prix moyen d'un article par jour de la semaine.

Tout d'abord, supposons que nous ayons un tableau qui conserve des enregistrements quotidiens de tous les prix des articles.

```
CREATE TABLE tbl_stock(item NVARCHAR(10), weekday NVARCHAR(10), price INT);

INSERT INTO tbl_stock VALUES
('Item1', 'Mon', 110), ('Item2', 'Mon', 230), ('Item3', 'Mon', 150),
('Item1', 'Tue', 115), ('Item2', 'Tue', 231), ('Item3', 'Tue', 162),
('Item1', 'Wed', 110), ('Item2', 'Wed', 240), ('Item3', 'Wed', 162),
('Item1', 'Thu', 109), ('Item2', 'Thu', 228), ('Item3', 'Thu', 145),
('Item1', 'Fri', 120), ('Item2', 'Fri', 210), ('Item3', 'Fri', 125),
('Item1', 'Mon', 122), ('Item2', 'Mon', 225), ('Item3', 'Mon', 140),
('Item1', 'Tue', 110), ('Item2', 'Tue', 235), ('Item3', 'Tue', 154),
('Item1', 'Wed', 125), ('Item2', 'Wed', 220), ('Item3', 'Wed', 142);
```

La table devrait ressembler à la suivante:

```
+========+=========+=======+
|   item | weekday | price |
+========+=========+=======+
|  Item1 |    Mon  |   110 |
+--------+---------+-------+
|  Item2 |    Mon  |   230 |
+--------+---------+-------+
|  Item3 |    Mon  |   150 |
+--------+---------+-------+
|  Item1 |    Tue  |   115 |
+--------+---------+-------+
|  Item2 |    Tue  |   231 |
+--------+---------+-------+
|  Item3 |    Tue  |   162 |
+--------+---------+-------+
|          . . .           |
+--------+---------+-------+
|  Item2 |    Wed  |   220 |
+--------+---------+-------+
|  Item3 |    Wed  |   142 |
+--------+---------+-------+
```

Afin d'effectuer une agrégation consistant à trouver le prix moyen par article pour chaque jour de la semaine, nous allons utiliser l'opérateur relationnel `PIVOT` pour faire pivoter le `weekday` de la

colonne de l'expression de la valeur d'une table en valeurs de ligne agrégées comme ci-dessous:

```
SELECT * FROM tbl_stock
PIVOT (
    AVG(price) FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) pvt;
```

Résultat:

```
+--------+------+------+------+------+------+
|  item  |  Mon |  Tue |  Wed |  Thu |  Fri |
+--------+------+------+------+------+------+
|  Item1 |  116 |  112 |  117 |  109 |  120 |
|  Item2 |  227 |  233 |  230 |  228 |  210 |
|  Item3 |  145 |  158 |  152 |  145 |  125 |
+--------+------+------+------+------+------+
```

Enfin, pour effectuer l'opération inverse de `PIVOT` , nous pouvons utiliser l'opérateur relationnel `UNPIVOT` pour faire pivoter les colonnes en lignes comme ci-dessous:

```
SELECT * FROM tbl_stock
PIVOT (
    AVG(price) FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) pvt
UNPIVOT (
    price FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) unpvt;
```

Résultat:

```
+=======+========+=========+
|  item |  price | weekday |
+=======+========+=========+
| Item1 |    116 |     Mon |
+-------+--------+---------+
| Item1 |    112 |     Tue |
+-------+--------+---------+
| Item1 |    117 |     Wed |
+-------+--------+---------+
| Item1 |    109 |     Thu |
+-------+--------+---------+
| Item1 |    120 |     Fri |
+-------+--------+---------+
| Item2 |    227 |     Mon |
+-------+--------+---------+
| Item2 |    233 |     Tue |
+-------+--------+---------+
| Item2 |    230 |     Wed |
+-------+--------+---------+
| Item2 |    228 |     Thu |
+-------+--------+---------+
| Item2 |    210 |     Fri |
+-------+--------+---------+
| Item3 |    145 |     Mon |
+-------+--------+---------+
| Item3 |    158 |     Tue |
```

```
+-------+--------+---------+
| Item3 |    152 |     Wed |
+-------+--------+---------+
| Item3 |    145 |     Thu |
+-------+--------+---------+
| Item3 |    125 |     Fri |
+-------+--------+---------+
```

PIVOT dynamique

Un problème avec la requête `PIVOT` est que vous devez spécifier toutes les valeurs dans la sélection `IN` si vous souhaitez les voir sous forme de colonnes. Un moyen rapide de contourner ce problème consiste à créer une sélection dynamique IN faisant de votre dynamique `PIVOT` .

Pour la démonstration, nous utiliserons une table `Books` dans la base de données d'une `Bookstore` . Nous supposons que la table est tout à fait normalisée et comporte les colonnes suivantes

```
Table: Books
-----------------------------
BookId (Primary Key Column)
Name
Language
NumberOfPages
EditionNumber
YearOfPrint
YearBoughtIntoStore
ISBN
AuthorName
Price
NumberOfUnitsSold
```

Le script de création de la table sera comme suit:

```
CREATE TABLE [dbo].[BookList](
      [BookId] [int] NOT NULL,
      [Name] [nvarchar](100) NULL,
      [Language] [nvarchar](100) NULL,
      [NumberOfPages] [int] NULL,
      [EditionNumber] [nvarchar](10) NULL,
      [YearOfPrint] [int] NULL,
      [YearBoughtIntoStore] [int] NULL,
[NumberOfBooks] [int] NULL,
[ISBN] [nvarchar](30) NULL,
      [AuthorName] [nvarchar](200) NULL,
      [Price] [money] NULL,
      [NumberOfUnitsSold] [int] NULL,
 CONSTRAINT [PK_BookList] PRIMARY KEY CLUSTERED
(
      [BookId] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS =
ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

Maintenant, si nous avons besoin d'interroger sur la base de données et de déterminer le nombre

de livres en anglais, russe, allemand, hindi et latin achetés dans la librairie chaque année et de présenter notre production dans un petit format de rapport, nous pouvons utiliser la requête PIVOT

```
SELECT * FROM
  (
   SELECT YearBoughtIntoStore AS [Year Bought],[Language], NumberOfBooks
   FROM BookList
  ) sourceData
 PIVOT
  (
  SUM(NumberOfBooks)
  FOR [Language] IN (English, Russian, German, Hindi, Latin)
  ) pivotrReport
```

Le cas particulier est lorsque nous n'avons pas une liste complète des langues, nous allons donc utiliser le SQL dynamique comme ci-dessous

```
DECLARE @query VARCHAR(4000)
DECLARE @languages VARCHAR(2000)
SELECT @languages =
        STUFF((SELECT DISTINCT '],['+LTRIM([Language])FROM [dbo].[BookList]
        ORDER BY '],['+LTRIM([Language]) FOR XML PATH('') ),1,2,'') + ']'
SET @query=
'SELECT * FROM
  (SELECT YearBoughtIntoStore AS [Year Bought],[Language],NumberOfBooks
   FROM BookList) sourceData
PIVOT(SUM(NumberOfBooks)FOR [Language] IN ('+ @languages +')) pivotrReport' EXECUTE(@query)
```

Lire PIVOT / UNPIVOT en ligne: https://riptutorial.com/fr/sql-server/topic/591/pivot---unpivot

Chapitre 83: POUR JSON

Examples

POUR JSON PATH

Formate les résultats de la requête SELECT en tant que texte JSON. La clause FOR JSON PATH est ajoutée après la requête:

```
SELECT top 3 object_id, name, type, principal_id FROM sys.objects
FOR JSON PATH
```

Les noms de colonne seront utilisés comme clés dans JSON et les valeurs de cellule seront générées en tant que valeurs JSON. Le résultat de la requête serait un tableau d'objets JSON:

```
[
  {"object_id":3,"name":"sysrscols","type":"S "},
  {"object_id":5,"name":"sysrowsets","type":"S "},
  {"object_id":6,"name":"sysclones","type":"S "}
]
```

Les valeurs NULL dans la colonne principal_id seront ignorées (elles ne seront pas générées).

POUR JSON PATH avec alias de colonne

FOR JSON PATH vous permet de contrôler le format du JSON de sortie en utilisant des alias de colonnes:

```
SELECT top 3 object_id as id, name as [data.name], type as [data.type]
FROM sys.objects
FOR JSON PATH
```

L'alias de colonne sera utilisé comme nom de clé. Les alias de colonne séparés par des points (data.name et data.type) seront générés en tant qu'objets imbriqués. Si deux colonnes ont le même préfixe en notation point, elles seront regroupées dans un seul objet (données dans cet exemple):

```
[
  {"id":3,"data":{"name":"sysrscols","type":"S "}},
  {"id":5,"data":{"name":"sysrowsets","type":"S "}},
  {"id":6,"data":{"name":"sysclones","type":"S "}}
]
```

Clause FOR JSON sans wrapper de tableau (objet unique en sortie)

L'option WITHOUT_ARRAY_WRAPPER permet de générer un objet unique au lieu du tableau. Utilisez cette option si vous savez que vous allez renvoyer une seule ligne / objet:

```
SELECT top 3 object_id, name, type, principal_id
FROM sys.objects
WHERE object_id = 3
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

Un seul objet sera retourné dans ce cas:

```
{"object_id":3,"name":"sysrscols","type":"S "}
```

INCLUDE_NULL_VALUES

La clause FOR JSON ignore les valeurs NULL dans les cellules. Si vous souhaitez générer des "clés": paires nulles pour les cellules contenant des valeurs NULL, ajoutez l'option INCLUDE_NULL_VALUES dans la requête:

```
SELECT top 3 object_id, name, type, principal_id
FROM sys.objects
FOR JSON PATH, INCLUDE_NULL_VALUES
```

Les valeurs NULL dans la colonne principal_id seront générées:

```
[
  {"object_id":3,"name":"sysrscols","type":"S ","principal_id":null},
  {"object_id":5,"name":"sysrowsets","type":"S ","principal_id":null},
  {"object_id":6,"name":"sysclones","type":"S ","principal_id":null}
]
```

Emballage des résultats avec l'objet ROOT

Enroule le tableau JSON renvoyé dans un objet racine supplémentaire avec la clé spécifiée:

```
SELECT top 3 object_id, name, type FROM sys.objects
FOR JSON PATH, ROOT('data')
```

Le résultat de la requête serait un tableau d'objets JSON à l'intérieur de l'objet wrapper:

```
{
  "data":[
            {"object_id":3,"name":"sysrscols","type":"S "},
            {"object_id":5,"name":"sysrowsets","type":"S "},
            {"object_id":6,"name":"sysclones","type":"S "}
          ]
}
```

POUR JSON AUTO

Niche automatiquement les valeurs de la deuxième table en tant que sous-tableau imbriqué d'objets JSON:

```
SELECT top 5 o.object_id, o.name, c.column_id, c.name
```

```
FROM sys.objects o
    JOIN sys.columns c ON o.object_id = c.object_id
FOR JSON AUTO
```

Le résultat de la requête serait un tableau d'objets JSON:

```
[
  {
   "object_id":3,
   "name":"sysrscols",
   "c":[
        {"column_id":12,"name":"bitpos"},
        {"column_id":6,"name":"cid"}
       ]
  },
  {
    "object_id":5,
    "name":"sysrowsets",
    "c":[
         {"column_id":13,"name":"colguid"},
         {"column_id":3,"name":"hbcolid"},
         {"column_id":8,"name":"maxinrowlen"}
     ]
  }
]
```

Création d'une structure JSON imbriquée personnalisée

Si vous avez besoin d'une structure JSON complexe qui ne peut pas être créée avec FOR JSON PATH ou FOR JSON AUTO, vous pouvez personnaliser votre sortie JSON en plaçant les sous-requêtes FOR JSON en tant qu'expressions de colonne:

```
SELECT top 5 o.object_id, o.name,
        (SELECT column_id, c.name
            FROM sys.columns c WHERE o.object_id = c.object_id
            FOR JSON PATH) as columns,
        (SELECT parameter_id, name
            FROM sys.parameters p WHERE o.object_id = p.object_id
            FOR JSON PATH) as parameters
FROM sys.objects o
FOR JSON PATH
```

Chaque sous-requête produira un résultat JSON qui sera inclus dans le contenu JSON principal.

Lire POUR JSON en ligne: https://riptutorial.com/fr/sql-server/topic/4661/pour-json

Chapitre 84: POUR XML PATH

Remarques

Il existe également plusieurs autres modes `FOR XML` :

- `FOR XML RAW` - Crée un élément `<row>` par ligne.
- `FOR XML AUTO` - Tente de générer automatiquement une hiérarchie.
- `FOR XML EXPLICIT` - Fournit plus de contrôle sur la forme du XML, mais est plus encombrant que `FOR XML PATH` .

Examples

Bonjour le monde XML

```
SELECT 'Hello World' FOR XML PATH('example')
```

```
<example>Hello World</example>
```

Spécifier les espaces de noms

SQL Server 2008

```
WITH XMLNAMESPACES (
    DEFAULT 'http://www.w3.org/2000/svg',
    'http://www.w3.org/1999/xlink' AS xlink
)
SELECT
    'example.jpg' AS 'image/@xlink:href',
    '50px' AS 'image/@width',
    '50px' AS 'image/@height'
FOR XML PATH('svg')
```

```
<svg xmlns:xlink="http://www.w3.org/1999/xlink" xmlns="http://www.w3.org/2000/svg">
    <image xlink:href="firefox.jpg" width="50px" height="50px"/>
</svg>
```

Spécification de la structure à l'aide d'expressions XPath

```
SELECT
    'XPath example' AS 'head/title',
    'This example demonstrates ' AS 'body/p',
    'https://www.w3.org/TR/xpath/' AS 'body/p/a/@href',
    'XPath expressions' AS 'body/p/a'
FOR XML PATH('html')
```

```
<html>
```

```
    <head>
        <title>XPath example</title>
    </head>
    <body>
        <p>This example demonstrates <a href="https://www.w3.org/TR/xpath/">XPath
expressions</a></p>
    </body>
</html>
```

Dans `FOR XML PATH` , les colonnes sans nom deviennent des nœuds de texte. `NULL` ou `''` devient donc des nœuds de texte vides. Remarque: vous pouvez convertir une colonne nommée en une colonne non nommée en utilisant `AS *`

```
DECLARE @tempTable TABLE (Ref INT, Des NVARCHAR(100), Qty INT)
INSERT INTO @tempTable VALUES (100001, 'Normal', 1), (100002, 'Foobar', 1), (100003, 'Hello
World', 2)

SELECT ROW_NUMBER() OVER (ORDER BY Ref) AS '@NUM',
     'REF' AS 'FLD/@NAME', REF AS 'FLD', '',
     'DES' AS 'FLD/@NAME', DES AS 'FLD', '',
     'QTY' AS 'FLD/@NAME', QTY AS 'FLD'
FROM @tempTable
FOR XML PATH('LIN'), ROOT('row')
```

```
<row>
  <LIN NUM="1">
    <FLD NAME="REF">100001</FLD>
    <FLD NAME="DES">Normal</FLD>
    <FLD NAME="QTY">1</FLD>
  </LIN>
  <LIN NUM="2">
    <FLD NAME="REF">100002</FLD>
    <FLD NAME="DES">Foobar</FLD>
    <FLD NAME="QTY">1</FLD>
  </LIN>
  <LIN NUM="3">
    <FLD NAME="REF">100003</FLD>
    <FLD NAME="DES">Hello World</FLD>
    <FLD NAME="QTY">2</FLD>
  </LIN>
</row>
```

L'utilisation de nœuds de texte (vides) permet de séparer le nœud de sortie précédent du nœud suivant, afin que SQL Server sache démarrer un nouvel élément pour la colonne suivante. Sinon, cela devient confus lorsque l'attribut existe déjà sur ce qu'il pense être l'élément "actuel".

Par exemple, sans les chaînes vides entre l'élément et l'attribut dans l' `SELECT` , SQL Server génère une erreur:

> La colonne centrée sur les attributs 'FLD / @ NAME' ne doit pas figurer après un frère non centré sur les attributs dans la hiérarchie XML de FOR XML PATH.

Notez également que cet exemple a également inclus le XML dans un élément racine nommé `row` , spécifié par `ROOT('row')`

Utilisation de FOR XML PATH pour concaténer des valeurs

Le `FOR XML PATH` peut être utilisé pour concaténer des valeurs dans string. L'exemple ci-dessous concatène des valeurs dans une chaîne `CSV` :

```
DECLARE @DataSource TABLE
(
    [rowID] TINYINT
   ,[FirstName] NVARCHAR(32)
);

INSERT INTO @DataSource ([rowID], [FirstName])
VALUES (1, 'Alex')
      ,(2, 'Peter')
      ,(3, 'Alexsandyr')
      ,(4, 'George');

SELECT STUFF
(
    (
        SELECT ',' + [FirstName]
        FROM @DataSource
        ORDER BY [rowID] DESC
        FOR XML PATH(''), TYPE
    ).value('.', 'NVARCHAR(MAX)')
    ,1
    ,1
    ,''
);
```

Quelques notes importantes:

- la clause `ORDER BY` peut être utilisée pour ordonner les valeurs de manière préférée
- Si une valeur plus longue est utilisée comme séparateur de concaténation, le paramètre de fonction `STUFF` doit également être modifié.

```
SELECT STUFF
(
    (
        SELECT '---' + [FirstName]
        FROM @DataSource
        ORDER BY [rowID] DESC
        FOR XML PATH(''), TYPE
    ).value('.', 'NVARCHAR(MAX)')
    ,1
    ,3 -- the "3" could also be represented as: LEN('---') for clarity
    ,''
);
```

- comme l'option `TYPE` et la fonction `.value` sont utilisées, la concaténation fonctionne avec la `NVARCHAR(MAX)`

Lire POUR XML PATH en ligne: https://riptutorial.com/fr/sql-server/topic/727/pour-xml-path

Chapitre 85: Privilèges ou autorisations

Examples

Règles simples

Octroi de l'autorisation de créer des tables

```
USE AdventureWorks;
GRANT CREATE TABLE TO MelanieK;
GO
```

Octroi de l'autorisation SHOWPLAN à un rôle d'application

```
USE AdventureWorks2012;
GRANT SHOWPLAN TO AuditMonitor;
GO
```

Octroi de CREATE VIEW avec GRANT OPTION

```
USE AdventureWorks2012;
GRANT CREATE VIEW TO CarmineEs WITH GRANT OPTION;
GO
```

Accorder tous les droits à un utilisateur sur une base de données spécifique

```
use YourDatabase
go
exec sp_addrolemember 'db_owner', 'UserName'
go
```

Lire Privilèges ou autorisations en ligne: https://riptutorial.com/fr/sql-server/topic/5333/privileges-ou-autorisations

Chapitre 86: Procédures stockées

Introduction

Dans SQL Server, une procédure est un programme stocké dans lequel vous pouvez transmettre des paramètres. Il ne renvoie pas de valeur comme le fait une fonction. Cependant, il peut renvoyer un état de réussite / échec à la procédure qui l'a appelé.

Syntaxe

- CREATE {PROCEDURE | PROC} [nom_schéma.] Nom_procédure
- [@parameter [type_schema_name.] type de données
- [VARYING] [= par défaut] [OUT | SORTIE | LECTURE SEULEMENT]
- , @parameter [type_schema_name.] type de données
- [VARYING] [= par défaut] [OUT | SORTIE | LECTURE SEULEMENT]]
- [AVEC {CRYPTAGE | RECOMPILE | EXECUTE AS Clause}]
- [POUR RÉPLICATION]
- COMME
- COMMENCER
- [déclaration_section]
- exécutable_section
- FIN;

Examples

Créer et exécuter une procédure stockée de base

Utilisation de la table des `Authors` dans la base de données de la bibliothèque

```
CREATE PROCEDURE GetName
(
    @input_id INT = NULL,      --Input parameter,  id of the person, NULL default
    @name VARCHAR(128) = NULL  --Input parameter, name of the person, NULL default
)
AS
BEGIN
    SELECT Name + ' is from ' + Country
    FROM Authors
    WHERE Id = @input_id OR Name = @name
END
GO
```

Vous pouvez exécuter une procédure avec plusieurs syntaxes différentes. Tout d'abord, vous pouvez utiliser `EXECUTE` ou `EXEC`

```
EXECUTE GetName @id = 1
EXEC Getname @name = 'Ernest Hemingway'
```

De plus, vous pouvez omettre la commande EXEC. De plus, vous n'avez pas besoin de spécifier le paramètre que vous transmettez, lorsque vous transmettez tous les paramètres.

```
GetName NULL, 'Ernest Hemingway'
```

Lorsque vous souhaitez spécifier les paramètres d'entrée dans un ordre différent de celui dans lequel ils ont été déclarés dans la procédure, vous pouvez spécifier le nom du paramètre et affecter des valeurs. Par exemple

```
 CREATE PROCEDURE dbo.sProcTemp
 (
    @Param1 INT,
    @Param2 INT
)
AS
BEGIN

    SELECT
        Param1 = @Param1,
        Param2 = @Param2

END
```

l'ordre normal d'exécution de cette procédure est de spécifier la valeur pour @ Param1 d'abord, puis @ Param2 seconde. Donc ça va ressembler à ça

```
  EXEC dbo.sProcTemp @Param1 = 0,@Param2=1
```

Mais il est également possible que vous puissiez utiliser les éléments suivants

```
  EXEC dbo.sProcTemp @Param2 = 0,@Param1=1
```

dans ce cas, vous spécifiez la valeur pour @ param2 en premier et @ Param1 seconde. Ce qui signifie que vous n'avez pas à conserver le même ordre que celui déclaré dans la procédure, mais vous pouvez avoir n'importe quelle commande comme vous le souhaitez. mais vous devrez spécifier à quel paramètre vous définissez la valeur

Accéder à la procédure stockée à partir de n'importe quelle base de données

Et vous pouvez également créer une procédure avec un préfixe `sp_` ces procédures, comme toutes les procédures stockées du système, peuvent être exécutées sans spécifier la base de données en raison du comportement par défaut de SQL Server. Lorsque vous exécutez une procédure stockée commençant par "sp_", SQL Server recherche d'abord la procédure dans la base de données master. Si la procédure est introuvable dans master, elle recherche dans la base de données active. Si vous avez une procédure stockée à laquelle vous souhaitez accéder depuis toutes vos bases de données, créez-la dans master et utilisez un nom incluant le préfixe "sp_".

```
Use Master
```

```
CREATE PROCEDURE sp_GetName
(
    @input_id INT = NULL,       --Input parameter,  id of the person, NULL default
    @name VARCHAR(128) = NULL  --Input parameter, name of the person, NULL default
)
AS
BEGIN
    SELECT Name + ' is from ' + Country
    FROM Authors
    WHERE Id = @input_id OR Name = @name
END
GO
```

PROCÉDURE STOCKÉE avec paramètres OUT

Les procédures stockées peuvent renvoyer des valeurs à l'aide du mot clé `OUTPUT` dans sa liste de paramètres.

Création d'une procédure stockée avec un seul paramètre de sortie

```
CREATE PROCEDURE SprocWithOutParams
(
    @InParam VARCHAR(30),
    @OutParam VARCHAR(30) OUTPUT
)
AS
BEGIN
    SELECT @OutParam = @InParam + ' must come out'
    RETURN
END
GO
```

Exécuter la procédure stockée

```
DECLARE @OutParam VARCHAR(30)
EXECUTE SprocWithOutParams 'what goes in', @OutParam OUTPUT
PRINT @OutParam
```

Création d'une procédure stockée avec plusieurs paramètres sortants

```
CREATE PROCEDURE SprocWithOutParams2
(
    @InParam VARCHAR(30),
    @OutParam VARCHAR(30) OUTPUT,
```

```
    @OutParam2 VARCHAR(30) OUTPUT
)
AS
BEGIN
    SELECT @OutParam = @InParam +' must come out'
    SELECT @OutParam2 = @InParam +' must come out'
    RETURN
END
GO
```

Exécuter la procédure stockée

```
DECLARE @OutParam VARCHAR(30)
DECLARE @OutParam2 VARCHAR(30)
EXECUTE SprocWithOutParams2 'what goes in', @OutParam OUTPUT, @OutParam2 OUTPUT
PRINT @OutParam
PRINT @OutParam2
```

Procédure stockée avec If ... Else et Insert In operation

Créer une table exemple `Employee` :

```
CREATE TABLE Employee
(
    Id INT,
    EmpName VARCHAR(25),
    EmpGender VARCHAR(6),
    EmpDeptId INT
)
```

Crée une procédure stockée qui vérifie si les valeurs transmises dans la procédure stockée ne sont pas nulles ou non et effectuent une opération d'insertion dans la table Employee.

```
CREATE PROCEDURE spSetEmployeeDetails
(
    @ID int,
    @Name VARCHAR(25),
    @Gender VARCHAR(6),
    @DeptId INT
)
AS
BEGIN
    IF (
        (@ID IS NOT NULL AND LEN(@ID) !=0)
        AND (@Name IS NOT NULL AND LEN(@Name) !=0)
        AND (@Gender IS NOT NULL AND LEN(@Gender) !=0)
        AND (@DeptId IS NOT NULL AND LEN(@DeptId) !=0)
    )
    BEGIN
        INSERT INTO Employee
        (
            Id,
            EmpName,
            EmpGender,
```

```
            EmpDeptId
        )
        VALUES
        (
            @ID,
            @Name,
            @Gender,
            @DeptId
        )
    END
ELSE
    PRINT 'Incorrect Parameters'
END
GO
```

Exécuter la procédure stockée

```
DECLARE @ID INT,
    @Name VARCHAR(25),
    @Gender VARCHAR(6),
    @DeptId INT

EXECUTE spSetEmployeeDetails
    @ID = 1,
    @Name = 'Subin Nepal',
    @Gender = 'Male',
    @DeptId = 182666
```

SQL dynamique dans la procédure stockée

SQL dynamique nous permet de générer et d'exécuter des instructions SQL au moment de l'exécution. Le SQL dynamique est nécessaire lorsque nos instructions SQL contiennent un identifiant susceptible de changer à différents moments de la compilation.

Exemple simple de SQL dynamique:

```
CREATE PROC sp_dynamicSQL
@table_name      NVARCHAR(20),
@col_name        NVARCHAR(20),
@col_value       NVARCHAR(20)
AS
BEGIN
DECLARE  @Query  NVARCHAR(max)
SET      @Query = 'SELECT * FROM ' + @table_name
SET      @Query = @Query + ' WHERE ' + @col_name + ' = ' + ''''+@col_value+''''
EXEC     (@Query)
END
```

Dans la requête SQL ci-dessus, nous pouvons voir que nous pouvons utiliser la requête ci-dessus en définissant des valeurs dans `@table_name, @col_name, and @col_value` au moment de l'exécution. La requête est générée à l'exécution et exécutée. C'est une technique dans laquelle nous pouvons créer des scripts entiers sous forme de chaîne dans une variable et l'exécuter. Nous pouvons créer des requêtes plus complexes en utilisant le concept SQL dynamique et le concept de concaténation. Ce concept est très puissant lorsque vous souhaitez créer un script pouvant

être utilisé dans plusieurs conditions.

Exécution de la procédure stockée

```
DECLARE @table_name     NVARCHAR(20) = 'ITCompanyInNepal',
        @col_name       NVARCHAR(20) = 'Headquarter',
        @col_value      NVARCHAR(20) = 'USA'

EXEC    sp_dynamicSQL   @table_name,
                        @col_name,
                        @col_value
```

Tableau que j'ai utilisé

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyOne	Kathmandu	USA	300
2	CompanyTwo	Kathmandu	USA	260
3	CompanyThree	Kathmandu	Nepal	300
4	CompanyFour	Kathmandu	Nepal	180
6	CompanySix	Janakpur	USA	50
7	CompanySeven	Janakpur	Australia	100
8	CompanyEight	Birganj	Australia	150
9	CompanyNine	Biratnagar	Canada	200
10	CompanyTen	Pokhara	India	85

Sortie

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyOne	Kathmandu	USA	300
2	CompanyTwo	Kathmandu	USA	260
6	CompanySix	Janakpur	USA	50
1	CompanyA	Banglore	USA	400
2	CompanyB	Banglore	USA	450

Boucle simple

Premièrement, `#systables` des données dans une table temporaire nommée `#systables` et un numéro de ligne incrémenté afin de pouvoir interroger un enregistrement à la fois

```
select
    o.name,
    row_number() over (order by o.name) as rn
into
    #systables
from
    sys.objects as o
where
    o.type = 'S'
```

Ensuite, nous déclarons des variables pour contrôler la boucle et stocker le nom de la table dans cet exemple

```
declare
    @rn int = 1,
    @maxRn int = (
                    select
                        max(rn)
                    from
                        #systables as s
                    )
declare    @tablename sys name
```

Maintenant, nous pouvons faire une boucle en utilisant un simple moment. Nous incrémentons `@rn` dans l'instruction `select` , mais cela pourrait aussi être une instruction séparée pour ex `set @rn = @rn + 1` cela dépendra de vos besoins. Nous utilisons également la valeur de `@rn` avant de l'incrémenter pour sélectionner un seul enregistrement de `#systables` . Enfin, nous imprimons le nom de la table.

```
while @rn <= @maxRn
    begin

        select
            @tablename = name,
            @rn = @rn + 1
        from
            #systables as s
        where
            s.rn = @rn

        print @tablename
    end
```

Boucle simple

```
CREATE PROCEDURE SprocWithSimpleLoop
(
    @SayThis VARCHAR(30),
    @ThisManyTimes INT
)
AS
BEGIN
    WHILE @ThisManyTimes > 0
    BEGIN
        PRINT @SayThis;
        SET @ThisManyTimes = @ThisManyTimes - 1;
    END

    RETURN;
END
GO
```

Lire Procédures stockées en ligne: https://riptutorial.com/fr/sql-server/topic/3213/procedures-stockees

Chapitre 87: Récupérer des informations sur la base de données

Remarques

Comme pour les autres systèmes de bases de données relationnelles, SQL Server expose les métadonnées relatives à vos bases de données.

Cela est fourni via le schéma ISO Standard `INFORMATION_SCHEMA` ou les affichages de catalogue `sys` spécifiques à SQL Server.

Examples

Compter le nombre de tables dans une base de données

Cette requête renvoie le nombre de tables dans la base de données spécifiée.

```
USE YourDatabaseName
SELECT COUNT(*) from INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
```

Voici une autre façon de procéder pour toutes les tables utilisateur avec SQL Server 2008+. La référence est ici .

```
SELECT COUNT(*) FROM sys.tables
```

Récupérer une liste de toutes les procédures stockées

Les requêtes suivantes renvoient une liste de toutes les procédures stockées dans la base de données, avec des informations de base sur chaque procédure stockée:

SQL Server 2005

```
SELECT *
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_TYPE = 'PROCEDURE'
```

Les `ROUTINE_NAME` , `ROUTINE_SCHEMA` et `ROUTINE_DEFINITION` sont généralement les plus utiles.

SQL Server 2005

```
SELECT *
FROM sys.objects
WHERE type = 'P'
```

SQL Server 2005

```
SELECT *
FROM sys.procedures
```

Notez que cette version présente un avantage par rapport à la sélection à partir de sys.objects car elle inclut les colonnes supplémentaires `is_auto_executed` , `is_execution_replicated` , `is_repl_serializable` et `skips_repl_constraints` .

SQL Server 2005

```
SELECT *
FROM sysobjects
WHERE type = 'P'
```

Notez que la sortie contient de nombreuses colonnes qui ne seront jamais liées à une procédure stockée.

La prochaine série de requêtes renverra toutes les procédures stockées dans la base de données qui incluent la chaîne 'SearchTerm':

SQL Server 2005

```
SELECT o.name
FROM syscomments c
INNER JOIN sysobjects o
    ON c.id=o.id
WHERE o.xtype = 'P'
    AND c.TEXT LIKE '%SearchTerm%'
```

SQL Server 2005

```
SELECT p.name
FROM sys.sql_modules AS m
INNER JOIN sys.procedures AS p
    ON m.object_id = p.object_id
WHERE definition LIKE '%SearchTerm%'
```

Obtenir la liste de toutes les bases de données sur un serveur

Méthode 1: la requête ci **-** dessous sera applicable pour la version SQL Server 2000+ (contient 12 colonnes)

```
SELECT * FROM dbo.sysdatabases
```

Méthode 2: Ci **-** dessous, extraire des informations sur les bases de données avec plus d'informations (ex: Etat, Isolation, modèle de récupération, etc.)

Note: Ceci est une vue catalogue et sera disponible les versions SQL SERVER 2005+

```
SELECT * FROM sys.databases
```

Méthode 3: Pour voir uniquement les noms de bases de données, vous pouvez utiliser

sp_MSForEachDB non documenté

```
EXEC sp_MSForEachDB 'SELECT ''?'' AS DatabaseName'
```

Méthode 4: Ci **-** dessous, SP vous aidera à fournir la taille de la base de données avec le nom, le propriétaire, le statut, etc. des bases de données.

```
EXEC sp_helpdb
```

Méthode 5 De même, la procédure stockée ci-dessous indiquera le nom de la base de données, la taille de la base de données et les remarques.

```
EXEC sp_databases
```

Fichiers de base de données

Afficher tous les fichiers de données pour toutes les bases de données avec des informations sur la taille et la croissance

```
SELECT  d.name AS 'Database',
        d.database_id,
        SF.fileid,
        SF.name AS 'LogicalFileName',
        CASE SF.status & 0x100000
            WHEN 1048576 THEN 'Percentage'
            WHEN 0 THEN 'MB'
        END AS 'FileGrowthOption',
        Growth AS GrowthUnit,
        ROUND(((CAST(Size AS FLOAT)*8)/1024)/1024,2) [SizeGB], -- Convert 8k pages to GB
        Maxsize,
        filename AS PhysicalFileName

FROM    Master.SYS.SYSALTFILES SF
Join    Master.SYS.Databases d on sf.fileid = d.database_id

Order by d.name
```

Récupérer les options de base de données

La requête suivante renvoie les options de base de données et les métadonnées:

```
select * from sys.databases WHERE name = 'MyDatabaseName';
```

Afficher la taille de toutes les tables dans la base de données actuelle

```
SELECT
    s.name + '.' + t.NAME AS TableName,
    SUM(a.used_pages)*8 AS 'TableSizeKB'  --a page in SQL Server is 8kb
FROM sys.tables t
    JOIN sys.schemas s on t.schema_id = s.schema_id
    LEFT JOIN sys.indexes i ON t.OBJECT_ID = i.object_id
```

```
    LEFT JOIN sys.partitions p ON i.object_id = p.OBJECT_ID AND i.index_id = p.index_id
    LEFT JOIN sys.allocation_units a ON p.partition_id = a.container_id
GROUP BY
    s.name, t.name
ORDER BY
    --Either sort by name:
    s.name + '.' + t.NAME
    --Or sort largest to smallest:
    --SUM(a.used_pages) desc
```

Déterminer le chemin d'accès d'une connexion Windows

Cela affichera le type d'utilisateur et le chemin d'accès (quel groupe Windows à partir duquel l'utilisateur obtient ses autorisations).

```
xp_logininfo 'DOMAIN\user'
```

Récupérer des tables contenant une colonne connue

Cette requête renvoie tous les `COLUMNS` et leurs `TABLES` associées pour un nom de colonne donné. Il est conçu pour vous montrer quelles tables (inconnues) contiennent une colonne spécifiée (connue)

```
SELECT
    c.name AS ColName,
    t.name AS TableName
FROM
    sys.columns c
    JOIN sys.tables t ON c.object_id = t.object_id
WHERE
    c.name LIKE '%MyName%'
```

Voir si des fonctionnalités spécifiques à l'entreprise sont utilisées

Il est parfois utile de vérifier que votre travail sur Developer Edition n'a pas introduit de dépendance sur les fonctionnalités limitées à l'édition Enterprise.

Vous pouvez le faire en utilisant la vue système `sys.dm_db_persisted_sku_features` , comme ceci:

```
SELECT * FROM sys.dm_db_persisted_sku_features
```

Contre la base de données elle-même.

Cela listera les fonctionnalités utilisées, le cas échéant.

Rechercher et renvoyer toutes les tables et colonnes contenant une valeur de colonne spécifiée

Ce script, à partir d' ici et de là , renverra toutes les tables et toutes les colonnes contenant une valeur spécifiée. Ceci est puissant pour trouver où une certaine valeur est dans une base de

données. Cela peut être fastidieux, il est donc suggéré de l'exécuter d'abord dans un environnement de sauvegarde / test.

```
DECLARE @SearchStr nvarchar(100)
SET @SearchStr = '## YOUR STRING HERE ##'


    -- Copyright © 2002 Narayana Vyas Kondreddi. All rights reserved.
    -- Purpose: To search all columns of all tables for a given search string
    -- Written by: Narayana Vyas Kondreddi
    -- Site: http://vyaskn.tripod.com
    -- Updated and tested by Tim Gaunt
    -- http://www.thesitedoctor.co.uk
    --
http://blogs.thesitedoctor.co.uk/tim/2010/02/19/Search+Every+Table+And+Field+In+A+SQL+Server+Database+U

    -- Tested on: SQL Server 7.0, SQL Server 2000, SQL Server 2005 and SQL Server 2010
    -- Date modified: 03rd March 2011 19:00 GMT
    CREATE TABLE #Results (ColumnName nvarchar(370), ColumnValue nvarchar(3630))

    SET NOCOUNT ON

    DECLARE @TableName nvarchar(256), @ColumnName nvarchar(128), @SearchStr2 nvarchar(110)
    SET  @TableName = ''
    SET @SearchStr2 = QUOTENAME('%' + @SearchStr + '%','''')

    WHILE @TableName IS NOT NULL

    BEGIN
        SET @ColumnName = ''
        SET @TableName =
        (
            SELECT MIN(QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME))
            FROM     INFORMATION_SCHEMA.TABLES
            WHERE         TABLE_TYPE = 'BASE TABLE'
                AND    QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME) > @TableName
                AND    OBJECTPROPERTY(
                        OBJECT_ID(
                            QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME)
                             ), 'IsMSShipped'
                               ) = 0
        )

        WHILE (@TableName IS NOT NULL) AND (@ColumnName IS NOT NULL)

        BEGIN
            SET @ColumnName =
            (
                SELECT MIN(QUOTENAME(COLUMN_NAME))
                FROM     INFORMATION_SCHEMA.COLUMNS
                WHERE         TABLE_SCHEMA    = PARSENAME(@TableName, 2)
                    AND    TABLE_NAME    = PARSENAME(@TableName, 1)
                    AND    DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar', 'int',
'decimal')
                    AND    QUOTENAME(COLUMN_NAME) > @ColumnName
            )

            IF @ColumnName IS NOT NULL

            BEGIN
```

```
                INSERT INTO #Results
                EXEC
                (
                    'SELECT ''' + @TableName + '.' + @ColumnName + ''', LEFT(' + @ColumnName +
', 3630) FROM ' + @TableName + ' (NOLOCK) ' +
                    ' WHERE ' + @ColumnName + ' LIKE ' + @SearchStr2
                )
            END
        END
    END

    SELECT ColumnName, ColumnValue FROM #Results

DROP TABLE #Results
- See more at: http://thesitedoctor.co.uk/blog/search-every-table-and-field-in-a-sql-server-
database-updated#sthash.bBEqfJVZ.dpuf
```

Obtenez tous les schémas, tables, colonnes et index

```
SELECT
    s.name AS [schema],
    t.object_id AS [table_object_id],
    t.name AS [table_name],
    c.column_id,
    c.name AS [column_name],
    i.name AS [index_name],
    i.type_desc AS [index_type]
FROM sys.schemas AS s
INNER JOIN sys.tables AS t
    ON s.schema_id = t.schema_id
INNER JOIN sys.columns AS c
    ON t.object_id = c.object_id
LEFT JOIN sys.index_columns AS ic
    ON c.object_id = ic.object_id and c.column_id = ic.column_id
LEFT JOIN sys.indexes AS i
    ON ic.object_id = i.object_id and ic.index_id = i.index_id
ORDER BY [schema], [table_name], c.column_id;
```

Renvoyer une liste de travaux SQL Agent, avec des informations de planification

```
USE msdb
Go


SELECT dbo.sysjobs.Name AS 'Job Name',
      'Job Enabled' = CASE dbo.sysjobs.Enabled
            WHEN 1 THEN 'Yes'
            WHEN 0 THEN 'No'
      END,
      'Frequency' = CASE dbo.sysschedules.freq_type
            WHEN 1 THEN 'Once'
            WHEN 4 THEN 'Daily'
            WHEN 8 THEN 'Weekly'
            WHEN 16 THEN 'Monthly'
            WHEN 32 THEN 'Monthly relative'
            WHEN 64 THEN 'When SQLServer Agent starts'
```

```
      END,
      'Start Date' = CASE active_start_date
            WHEN 0 THEN null
            ELSE
            substring(convert(varchar(15),active_start_date),1,4) + '/' +
            substring(convert(varchar(15),active_start_date),5,2) + '/' +
            substring(convert(varchar(15),active_start_date),7,2)
      END,
      'Start Time' = CASE len(active_start_time)
            WHEN 1 THEN cast('00:00:0' + right(active_start_time,2) as char(8))
            WHEN 2 THEN cast('00:00:' + right(active_start_time,2) as char(8))
            WHEN 3 THEN cast('00:0'
                        + Left(right(active_start_time,3),1)
                        +':' + right(active_start_time,2) as char (8))
            WHEN 4 THEN cast('00:'
                        + Left(right(active_start_time,4),2)
                        +':' + right(active_start_time,2) as char (8))
            WHEN 5 THEN cast('0'
                        + Left(right(active_start_time,5),1)
                        +':' + Left(right(active_start_time,4),2)
                        +':' + right(active_start_time,2) as char (8))
            WHEN 6 THEN cast(Left(right(active_start_time,6),2)
                        +':' + Left(right(active_start_time,4),2)
                        +':' + right(active_start_time,2) as char (8))
      END,

      CASE len(run_duration)
            WHEN 1 THEN cast('00:00:0'
                        + cast(run_duration as char) as char (8))
            WHEN 2 THEN cast('00:00:'
                        + cast(run_duration as char) as char (8))
            WHEN 3 THEN cast('00:0'
                        + Left(right(run_duration,3),1)
                        +':' + right(run_duration,2) as char (8))
            WHEN 4 THEN cast('00:'
                        + Left(right(run_duration,4),2)
                        +':' + right(run_duration,2) as char (8))
            WHEN 5 THEN cast('0'
                        + Left(right(run_duration,5),1)
                        +':' + Left(right(run_duration,4),2)
                        +':' + right(run_duration,2) as char (8))
            WHEN 6 THEN cast(Left(right(run_duration,6),2)
                        +':' + Left(right(run_duration,4),2)
                        +':' + right(run_duration,2) as char (8))
      END as 'Max Duration',
    CASE(dbo.sysschedules.freq_subday_interval)
            WHEN 0 THEN 'Once'
            ELSE cast('Every '
                        + right(dbo.sysschedules.freq_subday_interval,2)
                        + ' '
                        +     CASE(dbo.sysschedules.freq_subday_type)
                                    WHEN 1 THEN 'Once'
                                    WHEN 4 THEN 'Minutes'
                                    WHEN 8 THEN 'Hours'
                              END as char(16))
    END as 'Subday Frequency'
FROM dbo.sysjobs
LEFT OUTER JOIN dbo.sysjobschedules
ON dbo.sysjobs.job_id = dbo.sysjobschedules.job_id
INNER JOIN dbo.sysschedules ON dbo.sysjobschedules.schedule_id = dbo.sysschedules.schedule_id
LEFT OUTER JOIN (SELECT job_id, max(run_duration) AS run_duration
```

```
            FROM dbo.sysjobhistory
            GROUP BY job_id) Q1
ON dbo.sysjobs.job_id = Q1.job_id
WHERE Next_run_time = 0

UNION

SELECT dbo.sysjobs.Name AS 'Job Name',
      'Job Enabled' = CASE dbo.sysjobs.Enabled
            WHEN 1 THEN 'Yes'
            WHEN 0 THEN 'No'
      END,
      'Frequency' = CASE dbo.sysschedules.freq_type
            WHEN 1 THEN 'Once'
            WHEN 4 THEN 'Daily'
            WHEN 8 THEN 'Weekly'
            WHEN 16 THEN 'Monthly'
            WHEN 32 THEN 'Monthly relative'
            WHEN 64 THEN 'When SQLServer Agent starts'
      END,
      'Start Date' = CASE next_run_date
            WHEN 0 THEN null
            ELSE
            substring(convert(varchar(15),next_run_date),1,4) + '/' +
            substring(convert(varchar(15),next_run_date),5,2) + '/' +
            substring(convert(varchar(15),next_run_date),7,2)
      END,
      'Start Time' = CASE len(next_run_time)
            WHEN 1 THEN cast('00:00:0' + right(next_run_time,2) as char(8))
            WHEN 2 THEN cast('00:00:' + right(next_run_time,2) as char(8))
            WHEN 3 THEN cast('00:0'
                        + Left(right(next_run_time,3),1)
                        +':' + right(next_run_time,2) as char (8))
            WHEN 4 THEN cast('00:'
                        + Left(right(next_run_time,4),2)
                        +':' + right(next_run_time,2) as char (8))
            WHEN 5 THEN cast('0' + Left(right(next_run_time,5),1)
                        +':' + Left(right(next_run_time,4),2)
                        +':' + right(next_run_time,2) as char (8))
            WHEN 6 THEN cast(Left(right(next_run_time,6),2)
                        +':' + Left(right(next_run_time,4),2)
                        +':' + right(next_run_time,2) as char (8))
      END,

      CASE len(run_duration)
            WHEN 1 THEN cast('00:00:0'
                        + cast(run_duration as char) as char (8))
            WHEN 2 THEN cast('00:00:'
                        + cast(run_duration as char) as char (8))
            WHEN 3 THEN cast('00:0'
                        + Left(right(run_duration,3),1)
                        +':' + right(run_duration,2) as char (8))
            WHEN 4 THEN cast('00:'
                        + Left(right(run_duration,4),2)
                        +':' + right(run_duration,2) as char (8))
            WHEN 5 THEN cast('0'
                        + Left(right(run_duration,5),1)
                        +':' + Left(right(run_duration,4),2)
                        +':' + right(run_duration,2) as char (8))
            WHEN 6 THEN cast(Left(right(run_duration,6),2)
                        +':' + Left(right(run_duration,4),2)
```

```
                        +':' + right(run_duration,2) as char (8))
      END as 'Max Duration',
    CASE(dbo.sysschedules.freq_subday_interval)
            WHEN 0 THEN 'Once'
            ELSE cast('Every '
                        + right(dbo.sysschedules.freq_subday_interval,2)
                        + ' '
                        +     CASE(dbo.sysschedules.freq_subday_type)
                                    WHEN 1 THEN 'Once'
                                    WHEN 4 THEN 'Minutes'
                                    WHEN 8 THEN 'Hours'
                                END as char(16))
    END as 'Subday Frequency'
FROM dbo.sysjobs
LEFT OUTER JOIN dbo.sysjobschedules ON dbo.sysjobs.job_id = dbo.sysjobschedules.job_id
INNER JOIN dbo.sysschedules ON dbo.sysjobschedules.schedule_id = dbo.sysschedules.schedule_id
LEFT OUTER JOIN (SELECT job_id, max(run_duration) AS run_duration
            FROM dbo.sysjobhistory
            GROUP BY job_id) Q1
ON dbo.sysjobs.job_id = Q1.job_id
WHERE Next_run_time <> 0

ORDER BY [Start Date],[Start Time]
```

Récupérer des informations sur les opérations de sauvegarde et de restauration

Pour obtenir la liste de toutes les opérations de sauvegarde effectuées sur l'instance de base de données en cours:

```
SELECT sdb.Name AS DatabaseName,
    COALESCE(CONVERT(VARCHAR(50), bus.backup_finish_date, 120),'-') AS LastBackUpDateTime
FROM sys.sysdatabases sdb
    LEFT OUTER JOIN msdb.dbo.backupset bus ON bus.database_name = sdb.name
ORDER BY sdb.name, bus.backup_finish_date DESC
```

Pour obtenir la liste de toutes les opérations de restauration effectuées sur l'instance de base de données en cours:

```
SELECT
    [d].[name] AS database_name,
    [r].restore_date AS last_restore_date,
    [r].[user_name],
    [bs].[backup_finish_date] AS backup_creation_date,
    [bmf].[physical_device_name] AS [backup_file_used_for_restore]
FROM master.sys.databases [d]
    LEFT OUTER JOIN msdb.dbo.[restorehistory] r ON r.[destination_database_name] = d.Name
    INNER JOIN msdb.dbo.backupset [bs] ON [r].[backup_set_id] = [bs].[backup_set_id]
    INNER JOIN msdb.dbo.backupmediafamily bmf ON [bs].[media_set_id] = [bmf].[media_set_id]
ORDER BY [d].[name], [r].restore_date DESC
```

Trouver chaque mention d'un champ dans la base de données

```
SELECT DISTINCT
 o.name AS Object_Name,o.type_desc
```

```
FROM sys.sql_modules m
   INNER JOIN sys.objects  o ON m.object_id=o.object_id
WHERE m.definition Like '%myField%'
ORDER BY 2,1
```

`myField` mentions de `myField` dans SProcs, Views, etc.

Lire Récupérer des informations sur la base de données en ligne: https://riptutorial.com/fr/sql-server/topic/697/recuperer-des-informations-sur-la-base-de-donnees

Chapitre 88: Récupérer des informations sur votre instance

Examples

Récupérer des serveurs locaux et distants

Pour récupérer une liste de tous les serveurs enregistrés sur l'instance:

```
EXEC sp_helpserver;
```

Obtenir des informations sur les sessions en cours et les exécutions de requêtes

```
sp_who2
```

Cette procédure peut être utilisée pour rechercher des informations sur les sessions de serveur SQL en cours. Comme il s'agit d'une procédure, il est souvent utile de stocker les résultats dans une table temporaire ou une variable de table afin de pouvoir commander, filtrer et transformer les résultats selon vos besoins.

La liste ci-dessous peut être utilisée pour une version `sp_who2` de `sp_who2` :

```
-- Create a variable table to hold the results of sp_who2 for querying purposes

DECLARE @who2 TABLE (
      SPID INT NULL,
      Status VARCHAR(1000) NULL,
      Login SYSNAME NULL,
      HostName SYSNAME NULL,
      BlkBy SYSNAME NULL,
      DBName SYSNAME NULL,
      Command VARCHAR(8000) NULL,
      CPUTime INT NULL,
      DiskIO INT NULL,
      LastBatch VARCHAR(250) NULL,
      ProgramName VARCHAR(250) NULL,
      SPID2 INT NULL, -- a second SPID for some reason...?
      REQUESTID INT NULL
)

INSERT INTO @who2
EXEC sp_who2

SELECT    *
FROM    @who2 w
WHERE    1=1
```

Exemples:

```
-- Find specific user sessions:
SELECT  *
FROM    @who2 w
WHERE   1=1
    and  login = 'userName'

-- Find longest CPUTime queries:
SELECT  top 5 *
FROM    @who2 w
WHERE   1=1
order   by CPUTime desc
```

Récupérer l'édition et la version de l'instance

```
SELECT    SERVERPROPERTY('ProductVersion') AS ProductVersion,
          SERVERPROPERTY('ProductLevel') AS ProductLevel,
          SERVERPROPERTY('Edition') AS Edition,
          SERVERPROPERTY('EngineEdition') AS EngineEdition;
```

Récupération de la disponibilité de l'instance en jours

```
SELECT  DATEDIFF(DAY, login_time, getdate()) UpDays
FROM    master..sysprocesses
WHERE   spid = 1
```

Informations sur la version de SQL Server

Pour découvrir l'édition, le niveau de produit et le numéro de version de SQL Server, ainsi que le nom de la machine hôte et le type de serveur:

```
SELECT    SERVERPROPERTY('MachineName') AS Host,
          SERVERPROPERTY('InstanceName') AS Instance,
          DB_NAME() AS DatabaseContext,
          SERVERPROPERTY('Edition') AS Edition,
          SERVERPROPERTY('ProductLevel') AS ProductLevel,
          CASE SERVERPROPERTY('IsClustered')
            WHEN 1 THEN 'CLUSTERED'
            ELSE 'STANDALONE' END AS ServerType,
          @@VERSION AS VersionNumber;
```

Informations générales sur les bases de données, les tables, les procédures stockées et leur recherche.

Requête pour rechercher les derniers sp exécutés dans la base de données

```
SELECT execquery.last_execution_time AS [Date Time], execsql.text AS [Script]
FROM sys.dm_exec_query_stats AS execquery
CROSS APPLY sys.dm_exec_sql_text(execquery.sql_handle) AS execsql
ORDER BY execquery.last_execution_time DESC
```

Interroger à travers des procédures stockées

```
SELECT o.type_desc AS ROUTINE_TYPE,o.[name] AS ROUTINE_NAME,
m.definition AS ROUTINE_DEFINITION
FROM sys.sql_modules AS m INNER JOIN sys.objects AS o
ON m.object_id = o.object_id WHERE m.definition LIKE '%Keyword%'
order by ROUTINE_NAME
```

Requête pour rechercher une colonne à partir de toutes les tables de la base de données

```
SELECT t.name AS table_name,
SCHEMA_NAME(schema_id) AS schema_name,
c.name AS column_name
FROM sys.tables AS t
INNER JOIN sys.columns c ON t.OBJECT_ID = c.OBJECT_ID
where c.name like 'Keyword%'
ORDER BY schema_name, table_name;
```

Requête pour vérifier les détails de la restauration

```
WITH LastRestores AS
(
SELECT
    DatabaseName = [d].[name] ,
    [d].[create_date] ,
    [d].[compatibility_level] ,
    [d].[collation_name] ,
    r.*,
    RowNum = ROW_NUMBER() OVER (PARTITION BY d.Name ORDER BY r.[restore_date] DESC)
FROM master.sys.databases d
LEFT OUTER JOIN msdb.dbo.[restorehistory] r ON r.[destination_database_name] = d.Name
)
SELECT *
FROM [LastRestores]
WHERE [RowNum] = 1
```

Interroger pour trouver le journal

```
select top 100 * from databaselog
Order by Posttime desc
```

Interroger pour vérifier les détails de Sps

```
SELECT name, create_date, modify_date
FROM sys.objects
WHERE type = 'P'
Order by modify_date desc
```

Lire Récupérer des informations sur votre instance en ligne: https://riptutorial.com/fr/sql-server/topic/2029/recuperer-des-informations-sur-votre-instance

Chapitre 89: Rendez-vous

Syntaxe

- EOMONTH (*start_date* [, month_to_add])

Remarques

Selon https://msdn.microsoft.com/en-us/library/ms187819.aspx , `DateTime` s ne sont précis qu'à 3 ms.

Arrondi de date / heure Les valeurs de date / heure de seconde fraction fractionnaire sont arrondies aux incréments de .000, .003 ou .007 secondes, comme indiqué dans le tableau suivant.

Valeur spécifiée par l'utilisateur	Valeur stockée du système
01/01/98 23: 59: 59.999	1998-01-02 00: 00: 00.000
------	------
01/01/98 23: 59: 59.995	1998-01-01 23: 59: 59.997
01/01/98 23: 59: 59.996	
01/01/98 23: 59: 59.997	
01/01/98 23: 59: 59,998	
------	------
01/01/98 23: 59: 59.992	1998-01-01 23: 59: 59.993
01/01/98 23: 59: 59.993	
01/01/98 23: 59: 59.994	
------	------
01/01/98 23: 59: 59.990	1998-01-01 23: 59: 59.990
01/01/98 23: 59: 59.991	
------	------

Si vous avez besoin de plus de précision, utilisez `time` , `datetime2` ou `datetimeoffset` .

Examples

Formatage de la date et de l'heure avec CONVERT

Vous pouvez utiliser la fonction CONVERT pour convertir un type de données datetime en une chaîne formatée.

```
SELECT GETDATE() AS [Result] -- 2016-07-21 07:56:10.927
```

Vous pouvez également utiliser des codes intégrés pour convertir dans un format spécifique. Voici les options intégrées à SQL Server:

```
DECLARE @convert_code INT = 100 -- See Table Below
SELECT CONVERT(VARCHAR(30), GETDATE(), @convert_code) AS [Result]
```

@convert_code	Résultat
100	"21 juillet 2016 7:56"
101	"21/07/2016"
102	"2016.07.21"
103	"21/07/2016"
104	"21.07.2016"
105	"21-07-2016"
106	"21 juil. 2016"
107	"21 juillet 2016"
108	"07:57:05"
109	"21 juillet 2016 7: 57: 45: 707"
110	"21-07-2016"
111	"2016/07/21"
112	"20160721"
113	"21 juil. 2016 07: 57: 59: 553"
114	"07: 57: 59: 553"
120	"2016-07-21 07:57:59"

@convert_code	Résultat
121	"2016-07-21 07: 57: 59.553"
126	"2016-07-21T07: 58: 34.340"
127	"2016-07-21T07: 58: 34.340"
130	"16 ???? 1437 7: 58: 34: 340AM"
131	"16/10/1437 7: 58: 34: 340AM"

```
SELECT GETDATE() AS [Result]                                -- 2016-07-21 07:56:10.927
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),100) AS [Result] -- Jul 21 2016  7:56AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),101) AS [Result] -- 07/21/2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),102) AS [Result] -- 2016.07.21
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),103) AS [Result] -- 21/07/2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),104) AS [Result] -- 21.07.2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),105) AS [Result] -- 21-07-2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),106) AS [Result] -- 21 Jul 2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),107) AS [Result] -- Jul 21, 2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),108) AS [Result] -- 07:57:05
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),109) AS [Result] -- Jul 21 2016  7:57:45:707AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),110) AS [Result] -- 07-21-2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),111) AS [Result] -- 2016/07/21
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),112) AS [Result] -- 20160721
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),113) AS [Result] -- 21 Jul 2016 07:57:59:553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),114) AS [Result] -- 07:57:59:553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),120) AS [Result] -- 2016-07-21 07:57:59
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),121) AS [Result] -- 2016-07-21 07:57:59.553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),126) AS [Result] -- 2016-07-21T07:58:34.340
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),127) AS [Result] -- 2016-07-21T07:58:34.340
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),130) AS [Result] -- 16 ???? 1437  7:58:34:340AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),131) AS [Result] -- 16/10/1437  7:58:34:340AM
```

Formatage de la date et de l'heure à l'aide de FORMAT

SQL Server 2012

Vous pouvez utiliser la nouvelle fonction: `FORMAT()` .

En utilisant cela, vous pouvez transformer vos champs `DATETIME` en votre propre format `VARCHAR` personnalisé.

Exemple

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'dddd, MMMM dd, yyyy hh:mm:ss tt')
```

Lundi 5 septembre 2016 12:01:02

Arguments

Étant donné que `DATETIME` cours de formatage est `2016-09-05 00:01:02.333` , le tableau ci-dessous indique leur résultat pour l'argument fourni.

Argument	Sortie
aaaa	2016
yy	16
MMMM	septembre
MM	09
M	9
dddd	Lundi
ddd	Lun
dd	05
ré	5
HH	00
H	0
hh	12
h	12
mm	01
m	1
ss	02
s	2
tt	UN M
t	UNE
fff	333
ff	33
F	3

Vous pouvez également fournir un seul argument à la fonction `FORMAT()` pour générer une sortie

pré-formatée:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'U')
```

Lundi 5 septembre 2016 04:01:02

Argument unique	Sortie
ré	Lundi 5 septembre 2016
ré	9/5/2016
F	Lundi 5 septembre 2016 12:01:02
F	Lundi 5 septembre 2016 00h01
g	05/09/2016 12:01:02 AM
g	05/09/2016 00:01 AM
M	Septembre 05
O	2016-09-05T00: 01: 02.3330000
R	Lun. 05 sept. 2016 00:01:02 GMT
s	2016-09-05T00: 01: 02
T	12h01: 02h
t	00h01
U	Lundi 5 septembre 2016 04:01:02
tu	2016-09-05 00: 01: 02Z
Y	Septembre 2016

Remarque: la liste ci-dessus utilise la culture `en-US` . Une culture différente peut être spécifiée pour `FORMAT()` via le troisième paramètre:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'U', 'zh-cn')
```

2016 9 5 4:01:02

Récupère le DateTime actuel

Les fonctions `GETDATE` et `GETUTCDATE` chacune la date et l'heure actuelles sans décalage de fuseau horaire.

La valeur de retour des deux fonctions est basée sur le système d'exploitation de l'ordinateur sur lequel l'instance de SQL Server est exécutée.

La valeur de retour de GETDATE représente l'heure actuelle dans le même fuseau horaire que le système d'exploitation. La valeur de retour de GETUTCDATE représente l'heure UTC actuelle.

L'une ou l'autre fonction peut être incluse dans la clause `SELECT` d'une requête ou dans le cadre d'une expression booléenne dans la clause `WHERE` .

Exemples:

```
-- example query that selects the current time in both the server time zone and UTC
SELECT GETDATE() as SystemDateTime, GETUTCDATE() as UTCDateTime

-- example query records with EventDate in the past.
SELECT * FROM MyEvents WHERE EventDate < GETDATE()
```

Il existe quelques autres fonctions intégrées qui renvoient des variations différentes de la date et heure actuelle:

```
SELECT
    GETDATE(),          --2016-07-21 14:27:37.447
    GETUTCDATE(),       --2016-07-21 18:27:37.447
    CURRENT_TIMESTAMP,  --2016-07-21 14:27:37.447
    SYSDATETIME(),      --2016-07-21 14:27:37.4485768
    SYSDATETIMEOFFSET(),--2016-07-21 14:27:37.4485768 -04:00
    SYSUTCDATETIME()    --2016-07-21 18:27:37.4485768
```

DATEADD pour l'ajout et la soustraction de périodes

Syntaxe générale:

```
DATEADD (datepart , number , datetime_expr)
```

Pour ajouter une mesure de temps, le `number` doit être positif. Pour soustraire une mesure de temps, le `number` doit être négatif.

Exemples

```
DECLARE @now DATETIME2 = GETDATE();
SELECT @now;                          --2016-07-21 14:39:46.4170000
SELECT DATEADD(YEAR, 1, @now)         --2017-07-21 14:39:46.4170000
SELECT DATEADD(QUARTER, 1, @now)      --2016-10-21 14:39:46.4170000
SELECT DATEADD(WEEK, 1, @now)         --2016-07-28 14:39:46.4170000
SELECT DATEADD(DAY, 1, @now)          --2016-07-22 14:39:46.4170000
SELECT DATEADD(HOUR, 1, @now)         --2016-07-21 15:39:46.4170000
SELECT DATEADD(MINUTE, 1, @now)       --2016-07-21 14:40:46.4170000
SELECT DATEADD(SECOND, 1, @now)       --2016-07-21 14:39:47.4170000
SELECT DATEADD(MILLISECOND, 1, @now)--2016-07-21 14:39:46.4180000
```

REMARQUE: `DATEADD` accepte également les abréviations dans le paramètre `datepart` . L'utilisation de ces abréviations est généralement déconseillée car elles peuvent être source de confusion (`m` vs `mi` , `ww` vs `w` , etc.).

Date de référence des pièces

Ce sont les `datepart` valeurs disponibles pour les fonctions de date et heure:

datepart	Abréviations
an	yyyy
trimestre	qq, q
mois	mm, m
jour de l'année	dy, y
journée	dd, d
la semaine	wk, ww
jour de la semaine	dw, w
heure	hh
minute	mi, n
seconde	ss, s
milliseconde	Mme
microseconde	mcs
nanoseconde	ns

NOTE : L'utilisation des abréviations est généralement déconseillée car elles peuvent être source de confusion (`m` vs `mi` , `ww` vs `w` , etc.). La version longue de la `datepart` représentation favorise la clarté et la lisibilité, et doit être utilisé chaque fois que possible (`month` , `minute` , `week` , `weekday` de la `weekday` , etc.).

DATEDIFF pour calculer les différences de période

Syntaxe générale:

```
DATEDIFF (datepart, datetime_expr1, datetime_expr2)
```

Il retournera un nombre positif si `datetime_expr` est dans le passé relatif à `datetime_expr2` , et un nombre négatif sinon.

Exemples

```
DECLARE @now DATETIME2 = GETDATE();
DECLARE @oneYearAgo DATETIME2 = DATEADD(YEAR, -1, @now);
SELECT @now                                 --2016-07-21 14:49:50.9800000
SELECT @oneYearAgo                          --2015-07-21 14:49:50.9800000
SELECT DATEDIFF(YEAR, @oneYearAgo, @now)    --1
SELECT DATEDIFF(QUARTER, @oneYearAgo, @now) --4
SELECT DATEDIFF(WEEK, @oneYearAgo, @now)    --52
SELECT DATEDIFF(DAY, @oneYearAgo, @now)     --366
SELECT DATEDIFF(HOUR, @oneYearAgo, @now)    --8784
SELECT DATEDIFF(MINUTE, @oneYearAgo, @now)  --527040
SELECT DATEDIFF(SECOND, @oneYearAgo, @now)  --31622400
```

REMARQUE: `DATEDIFF` accepte également les abréviations dans le paramètre `datepart` . L'utilisation de ces abréviations est généralement déconseillée car elles peuvent être source de confusion (`m` vs `mi` , `ww` vs `w` , etc.).

`DATEDIFF` peut également être utilisé pour déterminer le décalage entre l'heure UTC et l'heure locale du serveur SQL. L'instruction suivante peut être utilisée pour calculer le décalage entre l'heure UTC et l'heure locale (y compris le fuseau horaire).

```
select  DATEDIFF(hh, getutcdate(), getdate()) as 'CentralTimeOffset'
```

DATEPART & DATENAME

`DATEPART` renvoie la `datepart` spécifiée de l'expression datetime spécifiée en tant que valeur numérique.

`DATENAME` renvoie une chaîne de caractères qui représente la `datepart` l'heure spécifiées pour la date spécifiée. En pratique, `DATENAME` est surtout utile pour obtenir le nom du mois ou du jour de la semaine.

Il y a aussi quelques fonctions raccourcies pour obtenir l'année, le mois ou le jour d'une expression_dateheure, qui se comportent comme `DATEPART` avec leurs respectives `datepart` unités.

Syntaxe:

```
DATEPART ( datepart , datetime_expr )
DATENAME ( datepart , datetime_expr )
DAY ( datetime_expr )
MONTH ( datetime_expr )
YEAR ( datetime_expr )
```

Exemples:

```
DECLARE @now DATETIME2 = GETDATE();
SELECT @now                       --2016-07-21 15:05:33.8370000
SELECT DATEPART(YEAR, @now)       --2016
SELECT DATEPART(QUARTER, @now)    --3
SELECT DATEPART(WEEK, @now)       --30
SELECT DATEPART(HOUR, @now)       --15
```

```
SELECT DATEPART(MINUTE, @now)     --5
SELECT DATEPART(SECOND, @now)     --33
-- Differences between DATEPART and DATENAME:
SELECT DATEPART(MONTH, @now)      --7
SELECT DATENAME(MONTH, @now)      --July
SELECT DATEPART(WEEKDAY, @now)    --5
SELECT DATENAME(WEEKDAY, @now)    --Thursday
--shorthand functions
SELECT DAY(@now)    --21
SELECT MONTH(@now)  --7
SELECT YEAR(@now)   --2016
```

REMARQUE: `DATEPART` et `DATENAME` acceptent également les abréviations dans le paramètre `datepart` . L'utilisation de ces abréviations est généralement déconseillée car elles peuvent être source de confusion (`m` vs `mi` , `ww` vs `w` , etc.).

Obtenir le dernier jour d'un mois

En utilisant les fonctions `DATEADD` et `DATEDIFF` , il est possible de renvoyer la dernière date d'un mois.

```
SELECT DATEADD(d, -1, DATEADD(m, DATEDIFF(m, 0, '2016-09-23') + 1, 0))
-- 2016-09-30 00:00:00.000
```

SQL Server 2012

La fonction `EOMONTH` fournit un moyen plus concis de renvoyer la dernière date d'un mois et possède un paramètre facultatif pour compenser le mois.

```
SELECT EOMONTH('2016-07-21')        --2016-07-31
SELECT EOMONTH('2016-07-21', 4)     --2016-11-30
SELECT EOMONTH('2016-07-21', -5)    --2016-02-29
```

Retourner la date juste d'un DateTime

Il existe plusieurs façons de renvoyer une date à partir d'un objet DateTime

1. `SELECT CONVERT(Date, GETDATE())`
2. `SELECT DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()))` renvoie 2016-07-21 00: 00: 00.000
3. `SELECT CAST(GETDATE() AS DATE)`
4. `SELECT CONVERT(CHAR(10),GETDATE(),111)`
5. `SELECT FORMAT(GETDATE(), 'yyyy-MM-dd')`

Notez que les options 4 et 5 retournent une chaîne, pas une date.

Fonction de création pour calculer l'âge d'une personne à une date précise

Cette fonction prendra 2 paramètres datetime, le DOB et une date pour vérifier l’âge à

```
  CREATE FUNCTION [dbo].[Calc_Age]
    (
    @DOB datetime , @calcDate datetime
    )
```

```
    RETURNS int
    AS
   BEGIN
declare @age int

IF (@calcDate < @DOB  )
RETURN -1

-- If a DOB is supplied after the comparison date, then return -1
SELECT @age = YEAR(@calcDate) - YEAR(@DOB) +
  CASE WHEN DATEADD(year,YEAR(@calcDate) - YEAR(@DOB)
  ,@DOB) > @calcDate THEN -1 ELSE 0 END


RETURN @age


END
```

par exemple pour vérifier l'âge d'une personne née le 1/1/2000

```
SELECT  dbo.Calc_Age('2000-01-01',Getdate())
```

OBJET DE DATE CROSS PLATFORM

SQL Server 2012

Dans Transact SQL, vous pouvez définir un objet comme `Date` (ou `DateTime`) à l'aide de la fonction `[DATEFROMPARTS][1]` (ou `[DATETIMEFROMPARTS][1]`) comme suit:

```
 DECLARE @myDate DATE=DATEFROMPARTS(1988,11,28)
 DECLARE @someMoment DATETIME=DATEFROMPARTS(1988,11,28,10,30,50,123)
```

Les paramètres que vous fournissez sont Année, Mois, Jour pour la fonction `DATEFROMPARTS` et, pour la fonction `DATETIMEFROMPARTS` , vous devez indiquer l'année, le mois, le jour, l'heure, les minutes, les secondes et les millisecondes.

Ces méthodes sont utiles et valent la peine d'être utilisées, car l'utilisation d'une chaîne simple pour créer une date (ou une date / heure) peut échouer en fonction de la région, de l'emplacement ou du format de date de la machine hôte.

Format de date étendu

Format de date	Instruction SQL	Échantillon sortie
AA-MM-JJ	SELECT RIGHT (CONVERT (VARCHAR (10), SYSDATETIME (), 20), 8) AS [AA-MM-JJ] SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 11), '/', '-') AS [AA-MM-JJ]	11-06-08

Format de date	Instruction SQL	Échantillon sortie
AAAA-MM-JJ	SELECT CONVERT (VARCHAR (10), SYSDATETIME (), 120) AS [AAAA-MM-JJ] SELECT REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 111), '/', '-') AS [AAAA-MM-JJ]	2011-06-08
AAAA-MD	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) COMME [AAAA-MD]	2011-6-8
YY-MD	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ( )) AS VARCHAR (2)) AS [YY-MD]	11-6-8
MD-AAAA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [MD-YYYY]	6-8-2011
MD-YY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [MD-YY]	6-8-11
DM-AAAA	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [DM-YYYY]	8-6-2011
DM-YY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [DM-YY]	8-6-11
AA-MM	SELECT RIGHT (CONVERT (VARCHAR (7), SYSDATETIME (), 20), 5) AS [AA-MM] SELECT SUBSTRING (CONVERT (VARCHAR (10), SYSDATETIME (), 120), 3, 5) AS [YY-MM]	11-06
AAAA-MM	SELECT CONVERT (VARCHAR (7), SYSDATETIME (), 120) AS [AAAA-MM]	2011-06
Y-M	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YY-M]	11-6

Format de date	Instruction SQL	Échantillon sortie
AAAA-M	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY-M]	2011-6
MM-YY	SELECT RIGHT (CONVERT (VARCHAR (8), SYSDATETIME (), 5), 5) AS [MM-YY] SELECT SUBSTRING (CONVERT (VARCHAR (8), SYSDATETIME (), 5), 4, 5) AS [MM-YY]	06-11
MM-AAAA	SELECT RIGHT (CONVERT (VARCHAR (10), SYSDATETIME (), 105), 7) AS [MM-AAAA]	06-2011
M-YY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M-YY]	6-11
M-AAAA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M-YYYY]	6-2011
MM-JJ	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 10) AS [MM-DD]	06-08
DD-MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 5) AS [JJ-MM]	08-06
MARYLAND	SELECT CAST (MOIS (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [MD]	6-8
DM	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MOIS (SYSDATETIME ()) AS VARCHAR (2)) AS [DM]	8-6
M / J / AAAA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M / J / AAAA]	6/8/2011
M / J / AA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M / D / YY]	6/8/11
J / M / AAAA	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONTH (SYSDATETIME ()) AS	8/6/2011

Format de date	Instruction SQL	Échantillon sortie
	VARCHAR (2)) + '/' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [J / M / AAAA]	
D / M / YY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [D / M / YY]	8/6/11
AAAA / M / J	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) COMME [AAAA / M / J]	2011/6/8
YY / M / D	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ( )) AS VARCHAR (2)) AS [YY / M / D]	11/6/8
MM / AA	SELECT RIGHT (CONVERT (VARCHAR (8), SYSDATETIME (), 3), 5) AS [MM / YY]	06/11
MM / AAAA	SELECT RIGHT (CONVERT (VARCHAR (10), SYSDATETIME (), 103), 7) AS [MM / AAAA]	06/2011
M / YY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M / YY]	6/11
M / AAAA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M / YYYY]	6/2011
AA / MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 11) AS [AA / MM]	11/06
AAAA / MM	SELECT CONVERT (VARCHAR (7), SYSDATETIME (), 111) AS [AAAA / MM]	2011/06
YY / M	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YY / M]	11/6
AAAA / M	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY / M]	2011/6
MM / JJ	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 1)	06/08

Format de date	Instruction SQL	Échantillon sortie
	AS [MM / DD]	
JJ / MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 3) AS [JJ / MM]	08/06
MARYLAND	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [M / D]	6/8
D / M	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [D / M]	8/6
MM.DD.AAAA	SELECT REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 101), '/', '.') AS [MM.DD.YYYY]	06.08.2011
MM.DD.YY	SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 1), '/', '.') AS [MM.DD.YY]	06.08.11
MDYYYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (ANNÉE (SYSDATETIME ()) AS VARCHAR (4)) COMME [MDYYYY]	6.8.2011
MDYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + DROIT (CAST (ANNÉE (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [MDYY]	6.8.11
JJ.MM.AAAA	SELECT CONVERT (VARCHAR (10), SYSDATETIME (), 104) AS [JJ.MM.AAAA]	08.06.2011
JJ.MM.AA	SELECT CONVERT (VARCHAR (10), SYSDATETIME (), 4) AS [JJ.MM.AA]	08.06.11
DMYYYY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [DMYYYY]	8.6.2011
DMYY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + DROIT (CAST (ANNÉE (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [DMYY]	8.6.11
AAAA.MD	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '.' + CAST (MONTH (SYSDATETIME ()) AS	2011.6.8

Format de date	Instruction SQL	Échantillon sortie
	VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [AAAA.MD]	
YY.MD	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [YY.MD]	11.6.8
MM.AAAA	SELECT RIGHT (CONVERT (VARCHAR (10), SYSDATETIME (), 104), 7) AS [MM.YYYY]	06.2011
MM.YY	SELECT RIGHT (CONVERT (VARCHAR (8), SYSDATETIME (), 4), 5) AS [MM.YY]	06.11
M.YYYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M.YYYY]	6.2011
M.YY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + DROIT (CAST (ANNÉE (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M.YY]	6.11
AAAA.MM	SELECT CONVERT (VARCHAR (7), SYSDATETIME (), 102) AS [AAAA.MM]	2011.06
YY.MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 2) AS [YY.MM]	11.06
AAAA.M	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '.' + CAST (MOIS (SYSDATETIME ()) AS VARCHAR (2)) AS [AAAA.M]	2011.6
YY.M	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YY.M]	11.6
MM.DD	SELECT RIGHT (CONVERT (VARCHAR (8), SYSDATETIME (), 2), 5) AS [MM.DD]	06.08
DD.MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 4) AS [JJ.MM]	08.06
MMJJAAAA	SELECT REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 101), '/', '') AS [MMJJAAAA]	06082011
MMJJAA	SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 1), '/', '') AS [MMJJAA]	060811

Format de date	Instruction SQL	Échantillon sortie
JJMMAAAA	SELECT REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 103), '/', '') AS [JJMMAAAA]	08062011
DDMMYY	SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 3), '/', '') AS [JJMMAA]	080611
MMYYYY	SÉLECTIONNER DROITE (REMPLACER (CONVERTIR (VARCHAR (10), SYSDATETIME (), 103), '/', ''), 6) AS [MMYYYY]	062011
MMYY	SELECT RIGHT (REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 3), '/', ''), 4) AS [MMYY]	0611
YYYYMM	SELECT CONVERT (VARCHAR (6), SYSDATETIME (), 112) COMME [AAAAMM]	201106
YMM	SELECT CONVERT (VARCHAR (4), SYSDATETIME (), 12) AS [YYMM]	1106
Mois JJ, AAAA	SELECT DATENAME (MONTH, SYSDATETIME ()) + '' + RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + ',' + DATENAME (YEAR, SYSDATETIME ()) AS [Mois JJ, AAAA]	08 juin 2011
Mon AAAA	SELECT LEFT (DATENAME (MOIS, SYSDATETIME ()), 3) + '' + DATENAME (YEAR, SYSDATETIME ()) AS [lundi AAAA]	Juin 2011
Mois AAAA	SELECT DATENAME (MONTH, SYSDATETIME ()) + '' + DATENAME (YEAR, SYSDATETIME ()) AS [Mois AAAA]	Juin 2011
DD Mois	SELECT RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + '' + DATENAME (MONTH, SYSDATETIME ()) AS [JJ Mois]	08 juin
Mois DD	SELECT DATENAME (MONTH, SYSDATETIME ()) + '' + RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) AS [Mois JJ]	08 juin
JJ Mois AA	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '' + DATENAME (MM, SYSDATETIME ()) + '' + RIGHT (CAST (AN (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [JJ Mois AA]	08 juin 11
JJ Mois AAAA	SELECT RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + '' + DATENAME (MOIS, SYSDATETIME ()) + '' + DATENAME (YEAR, SYSDATETIME ()) AS [JJ Mois AAAA]	08 juin 2011

Format de date	Instruction SQL	Échantillon sortie
Mon YY	SELECT REPLACE (RIGHT (CONVERT (VARCHAR (9), SYSDATETIME (), 6), 6), '', '-') AS [Mon-YY]	Juin-08
Mon-AAAA	SELECT REPLACE (RIGHT (CONVERT (VARCHAR (11), SYSDATETIME (), 106), 8), '', '-') AS [Mon-AAAA]	Juin-2011
JJ-Mon-AA	SELECT REPLACE (CONVERT (VARCHAR (9), SYSDATETIME (), 6), '', '-') AS [JJ-Mon-AA]	08-juin-11
JJ-Mon-AAAA	SELECT REPLACE (CONVERT (VARCHAR (11), SYSDATETIME (), 106), '', '-') AS [JJ-Mon-AAAA]	08-juin-2011

Lire Rendez-vous en ligne: https://riptutorial.com/fr/sql-server/topic/1471/rendez-vous

Chapitre 90: Requêtes avec des données JSON

Examples

Utilisation de valeurs de JSON dans la requête

La fonction JSON_VALUE vous permet de prendre des données à partir du texte JSON sur le chemin spécifié en tant que second argument et d'utiliser cette valeur dans n'importe quelle partie de la requête select:

```
select ProductID, Name, Color, Size, Price, JSON_VALUE(Data, '$.Type') as Type
from Product
where JSON_VALUE(Data, '$.Type') = 'part'
```

Utilisation des valeurs JSON dans les rapports

Une fois que les valeurs JSON sont extraites du texte JSON, vous pouvez les utiliser dans n'importe quelle partie de la requête. Vous pouvez créer un type de rapport sur les données JSON avec des regroupements, etc.:

```
select JSON_VALUE(Data, '$.Type') as type,
        AVG( cast(JSON_VALUE(Data, '$.ManufacturingCost') as float) ) as cost
from Product
group by JSON_VALUE(Data, '$.Type')
having JSON_VALUE(Data, '$.Type') is not null
```

Filtrage du texte JSON incorrect à partir des résultats de la requête

Si du texte JSON n'est peut-être pas correctement formaté, vous pouvez supprimer ces entrées de la requête en utilisant la fonction ISJSON.

```
select ProductID, Name, Color, Size, Price, JSON_VALUE(Data, '$.Type') as Type
from Product
where JSON_VALUE(Data, '$.Type') = 'part'
and ISJSON(Data) > 0
```

Mettre à jour la valeur dans la colonne JSON

La fonction JSON_MODIFY peut être utilisée pour mettre à jour une valeur sur un chemin. Vous pouvez utiliser cette fonction pour modifier la valeur d'origine de la cellule JSON dans l'instruction UPDATE:

```
update Product
set Data = JSON_MODIFY(Data, '$.Price', 24.99)
where ProductID = 17;
```

La fonction JSON_MODIFY mettra à jour ou créera une clé de prix (si elle n'existe pas). Si la nouvelle valeur est NULL, la clé sera supprimée. La fonction JSON_MODIFY traitera la nouvelle valeur comme une chaîne (échappez les caractères spéciaux, enveloppez-la avec des guillemets pour créer une chaîne JSON correcte). Si votre nouvelle valeur est un fragment JSON, vous devez l'emballer avec la fonction JSON_QUERY:

```
update Product
set Data = JSON_MODIFY(Data, '$.tags', JSON_QUERY('["promo","new"]'))
where ProductID = 17;
```

La fonction JSON_QUERY sans second paramètre se comporte comme un "cast to JSON". Puisque le résultat de JSON_QUERY est un fragment JSON valide (objet ou tableau), JSON_MODIFY n'échappera pas à cette valeur lorsque modifie l'entrée JSON.

Ajouter une nouvelle valeur au tableau JSON

La fonction JSON_MODIFY peut être utilisée pour ajouter une nouvelle valeur à un tableau dans JSON:

```
update Product
set Data = JSON_MODIFY(Data, 'append $.tags', "sales")
where ProductID = 17;
```

Une nouvelle valeur sera ajoutée à la fin du tableau ou un nouveau tableau avec une valeur ["sales"] sera créé. La fonction JSON_MODIFY traitera la nouvelle valeur comme une chaîne (échappez les caractères spéciaux, enveloppez-la avec des guillemets pour créer une chaîne JSON correcte). Si votre nouvelle valeur est un fragment JSON, vous devez l'emballer avec la fonction JSON_QUERY:

```
update Product
set Data = JSON_MODIFY(Data, 'append $.tags', JSON_QUERY('{"type":"new"}'))
where ProductID = 17;
```

La fonction JSON_QUERY sans second paramètre se comporte comme un "cast to JSON". Puisque le résultat de JSON_QUERY est un fragment JSON valide (objet ou tableau), JSON_MODIFY n'échappera pas à cette valeur lorsque modifie l'entrée JSON.

Table JOIN avec une collection JSON interne

Si vous avez une "table enfant" formatée en tant que collection JSON et stockée dans la ligne en tant que colonne JSON, vous pouvez décompresser cette collection, la transformer en table et la joindre à la ligne parente. Au lieu de l'opérateur JOIN standard, vous devez utiliser CROSS APPLY. Dans cet exemple, les composants du produit sont mis en forme en tant que collection d'objets JSON dans et stockés dans la colonne Données:

```
select ProductID, Name, Size, Price, Quantity, PartName, Code
from Product
    CROSS APPLY OPENJSON(Data, '$.Parts') WITH (PartName varchar(20), Code varchar(5))
```

Le résultat de la requête est équivalent à la jointure entre les tables Product et Part.

Recherche de lignes contenant une valeur dans le tableau JSON

Dans cet exemple, le tableau de balises peut contenir différents mots-clés tels que ["promo", "sales"], nous pouvons donc ouvrir ce tableau et les valeurs de filtre:

```
select ProductID, Name, Color, Size, Price, Quantity
from Product
    CROSS APPLY OPENJSON(Data, '$.Tags')
where value = 'sales'
```

OPENJSON ouvrira la collection interne de balises et la renverra sous forme de tableau. Ensuite, nous pouvons filtrer les résultats par une valeur dans la table.

Lire Requêtes avec des données JSON en ligne: https://riptutorial.com/fr/sql-server/topic/5028/requetes-avec-des-donnees-json

Chapitre 91: Sauvegarde et restauration de la base de données

Syntaxe

- *Base de données* BACKUP DATABASE TO *backup_device [, ... n]* WITH *with_options [, ... o]*
- *Base de données* RESTORE DATABASE FROM *backup_device [, ... n]* WITH *with_options [, ... o]*

Paramètres

Paramètre	Détails
base de données	Le nom de la base de données à sauvegarder ou à restaurer
backup_device	Le périphérique de sauvegarde ou de restauration de la base de données, comme {DISK ou TAPE}. Peut être séparé par des virgules (,)
avec_options	Diverses options pouvant être utilisées lors de l'exécution de l'opération. Comme le formatage du disque sur lequel la sauvegarde doit être placée ou la restauration de la base de données avec l'option replace.

Examples

Sauvegarde de base sur disque sans option

La commande suivante sauvegarde la base de données *"Utilisateurs"* dans *le* fichier *"D: \ DB_Backup"* . Il vaut mieux ne pas donner une extension.

```
BACKUP DATABASE Users TO DISK = 'D:\DB_Backup'
```

Restauration de base à partir du disque sans option

La commande suivante restaure la base de données *'Users' à* partir *du* fichier *'D: \ DB_Backup'* .

```
RESTORE DATABASE Users FROM DISK = 'D:\DB_Backup'
```

RESTORE la base de données avec REPLACE

Lorsque vous essayez de restaurer la base de données à partir d'un autre serveur, vous pouvez obtenir l'erreur suivante:

> Erreur 3154: le jeu de sauvegarde contient une sauvegarde d'une base de données autre que la base de données existante.

Dans ce cas, vous devez utiliser l'option WITH REPLACE pour remplacer la base de données par la base de données à partir de la sauvegarde:

```
RESTORE DATABASE WWIDW
FROM DISK = 'C:\Backup\WideWorldImportersDW-Full.bak'
WITH REPLACE
```

Même dans ce cas, vous pourriez avoir les erreurs en disant que les fichiers ne peuvent pas être localisés sur un chemin:

> Msg 3156, Niveau 16, État 3, Ligne 1 Le fichier 'WWI_Primary' ne peut pas être restauré dans 'D: \ Data \ WideWorldImportersDW.mdf'. Utilisez WITH MOVE pour identifier un emplacement valide pour le fichier.

Cette erreur se produit probablement parce que vos fichiers n'ont pas été placés sur le même chemin de dossier existant sur le nouveau serveur. Dans ce cas, vous devez déplacer les fichiers de base de données individuels vers un nouvel emplacement:

```
RESTORE DATABASE WWIDW
FROM DISK = 'C:\Backup\WideWorldImportersDW-Full.bak'
WITH REPLACE,
MOVE 'WWI_Primary' to 'C:\Data\WideWorldImportersDW.mdf',
MOVE 'WWI_UserData' to 'C:\Data\WideWorldImportersDW_UserData.ndf',
MOVE 'WWI_Log' to 'C:\Data\WideWorldImportersDW.ldf',
MOVE 'WWIDW_InMemory_Data_1' to 'C:\Data\WideWorldImportersDW_InMemory_Data_1'
```

Avec cette instruction, vous pouvez remplacer la base de données par tous les fichiers de base de données déplacés vers un nouvel emplacement.

Lire Sauvegarde et restauration de la base de données en ligne: https://riptutorial.com/fr/sql-server/topic/5826/sauvegarde-et-restauration-de-la-base-de-donnees

Chapitre 92: Schémas

Examples

Créer un schéma

```
CREATE SCHEMA dvr AUTHORIZATION Owner
    CREATE TABLE sat_Sales (source int, cost int, partid int)
    GRANT SELECT ON SCHEMA :: dvr TO  User1
    DENY SELECT ON SCHEMA :: dvr to User 2
GO
```

Alter Schema

```
ALTER SCHEMA dvr
    TRANSFER dbo.tbl_Staging;
GO
```

Cela transférerait la table tbl_Staging du schéma dbo au schéma dvr

Abandon des schémas

```
DROP SCHEMA dvr
```

Objectif

Le schéma fait référence à des tables de base de données spécifiques et à la manière dont elles sont liées les unes aux autres. Il fournit un schéma d'organisation de la façon dont la base de données est construite. La mise en œuvre de schémas de base de données présente d'autres avantages: les schémas peuvent être utilisés comme une méthode limitant / accordant l'accès à des tables spécifiques dans une base de données.

Lire Schémas en ligne: https://riptutorial.com/fr/sql-server/topic/5806/schemas

Chapitre 93: SCOPE_IDENTITY ()

Syntaxe

- SELECT SCOPE_IDENTITY ();
- SELECT SCOPE_IDENTITY () AS [SCOPE_IDENTITY];
- SCOPE_IDENTITY ()

Examples

Introduction avec un exemple simple

SCOPE_IDENTITY () renvoie la dernière valeur d'identité insérée dans une colonne d'identité de la même portée. Une portée est un module: une procédure stockée, un déclencheur, une fonction ou un lot. Par conséquent, deux instructions sont dans la même portée si elles se trouvent dans la même procédure stockée, la même fonction ou le même lot.

INSERT INTO ([colonne1], [colonne2]) VALUES (8,9);
ALLER
SELECT SCOPE_IDENTITY () AS [SCOPE_IDENTITY];
ALLER

Lire SCOPE_IDENTITY () en ligne: https://riptutorial.com/fr/sql-server/topic/5326/scope-identity---

Chapitre 94: SE FONDRE

Syntaxe

- COALESCE ([Column1], [Column2] .... [ColumnN]

Examples

Utilisation de COALESCE pour construire une chaîne délimitée par des virgules

Nous pouvons obtenir une chaîne délimitée par des virgules à partir de plusieurs lignes en utilisant la fusion comme indiqué ci-dessous.

Puisque la variable de table est utilisée, nous devons exécuter une requête entière une fois. Donc, pour faciliter la compréhension, j'ai ajouté BEGIN et END block.

```
BEGIN

    --Table variable declaration to store sample records
    DECLARE @Table TABLE (FirstName varchar(256), LastName varchar(256))

    --Inserting sample records into table variable @Table
    INSERT INTO @Table (FirstName, LastName)
    VALUES
    ('John','Smith'),
    ('Jane','Doe')

    --Creating variable to store result
    DECLARE @Names varchar(4000)

    --Used COLESCE function, so it will concatenate comma seperated FirstName into @Names
varible
    SELECT @Names = COALESCE(@Names + ',', '') + FirstName
    FROM @Table

    --Now selecting actual result
    SELECT @Names
    END
```

Exemple de base de coalescence

`COALESCE()` renvoie la première `NON NULL` dans une liste d'arguments. Supposons que nous ayons une table contenant des numéros de téléphone et des numéros de téléphone portable et que nous voulions en retourner une pour chaque utilisateur. Pour n'en obtenir qu'un, nous pouvons obtenir la première `NON NULL` .

```
DECLARE @Table TABLE (UserID int, PhoneNumber varchar(12), CellNumber varchar(12))
INSERT INTO @Table (UserID, PhoneNumber, CellNumber)
VALUES
```

```
(1,'555-869-1123',NULL),
(2,'555-123-7415','555-846-7786'),
(3,NULL,'555-456-8521')


SELECT
    UserID,
    COALESCE(PhoneNumber, CellNumber)
FROM
    @Table
```

Obtenir le premier non nul à partir d'une liste de valeurs de colonne

```
SELECT COALESCE(NULL, NULL, 'TechOnTheNet.com', NULL, 'CheckYourMath.com');
Result: 'TechOnTheNet.com'

SELECT COALESCE(NULL, 'TechOnTheNet.com', 'CheckYourMath.com');
Result: 'TechOnTheNet.com'

SELECT COALESCE(NULL, NULL, 1, 2, 3, NULL, 4);
Result: 1
```

Lire SE FONDRE en ligne: https://riptutorial.com/fr/sql-server/topic/3234/se-fondre

Chapitre 95: Sécurité au niveau des lignes

Examples

Prédicat de filtre RLS

La base de données SQL Server 2016+ et Azure Sql vous permet de filtrer automatiquement les lignes renvoyées dans l'instruction select à l'aide d'un prédicat. Cette fonctionnalité est appelée **sécurité au niveau de la ligne** .

Tout d'abord, vous avez besoin d'une fonction table contenant un prédicat décrivant la condition qui permettra aux utilisateurs de lire les données d'une table:

```
DROP FUNCTION IF EXISTS dbo.pUserCanAccessCompany
GO
CREATE FUNCTION

dbo.pUserCanAccessCompany(@CompanyID int)

    RETURNS TABLE
    WITH SCHEMABINDING
AS RETURN (
    SELECT 1 as canAccess WHERE

    CAST(SESSION_CONTEXT(N'CompanyID') as int) = @CompanyID

)
```

Dans cet exemple, le prédicat indique que seuls les utilisateurs ayant une valeur dans SESSION_CONTEXT correspondant à un argument d'entrée peuvent accéder à la société. Vous pouvez mettre toute autre condition, par exemple, qui vérifie le rôle de la base de données ou l'identifiant de base de données de l'utilisateur actuel, etc.

> La plupart du code ci-dessus est un modèle que vous allez copier-coller. La seule chose qui changera ici est le nom et les arguments du prédicat et de la condition dans la clause WHERE. Vous créez maintenant une stratégie de sécurité qui appliquera ce prédicat sur certaines tables.

Vous pouvez maintenant créer une politique de sécurité qui appliquera le prédicat sur certaines tables:

```
CREATE SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany(CompanyID) ON dbo.Company
    WITH (State=ON)
```

Cette stratégie de sécurité affecte des prédicats à la table de l'entreprise. Chaque fois que quelqu'un essaie de lire des données de la table Société, la stratégie de sécurité applique un prédicat sur chaque ligne, transmet la colonne CompanyID comme paramètre du prédicat et le prédicat évalue si cette ligne est renvoyée dans le résultat de la requête SELECT.

Modification de la politique de sécurité RLS

La stratégie de sécurité est un groupe de prédicats associés aux tables pouvant être gérées ensemble. Vous pouvez ajouter ou supprimer des prédicats ou activer / désactiver toute la stratégie.

Vous pouvez ajouter d'autres prédicats sur les tables de la stratégie de sécurité existante.

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany(CompanyID) ON dbo.Company
```

Vous pouvez supprimer des prédicats de la politique de sécurité:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    DROP FILTER PREDICATE ON dbo.Company
```

Vous pouvez désactiver la politique de sécurité

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = OFF );
```

Vous pouvez activer la stratégie de sécurité désactivée:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = ON );
```

Empêcher la mise à jour à l'aide du prédicat de bloc RLS

La sécurité au niveau de la ligne vous permet de définir des prédicats qui contrôlent qui peut mettre à jour les lignes de la table. Vous devez d'abord définir une fonction de valeur de table qui représente le prédicat qui contrôlera la stratégie d'accès.

CREER UNE FONCTION

dbo.pUserCanAccessProduct (@CompanyID int)

```
RETURNS TABLE
WITH SCHEMABINDING
```

AS RETURN (SELECT 1 comme canAccess WHERE

CAST (SESSION_CONTEXT (N'CompanyID ') en tant qu'int) = @CompanyID

) Dans cet exemple, le prédicat indique que seuls les utilisateurs ayant une valeur dans SESSION_CONTEXT correspondant à l’argument en entrée peuvent accéder à la société. Vous pouvez mettre toute autre condition, par exemple, qui vérifie le rôle de la base de données ou l'identifiant de base de données de l'utilisateur actuel, etc.

> La plupart du code ci-dessus est un modèle que vous allez copier-coller. La seule chose qui changera ici est le nom et les arguments du prédicat et de la condition dans

la clause WHERE. Vous créez maintenant une stratégie de sécurité qui appliquera ce prédicat sur certaines tables.

Maintenant, nous pouvons créer une politique de sécurité avec le prédicat qui bloquera les mises à jour sur la table de produit si la colonne CompanyID de la table ne satisfait pas aux prédicats.

CREATE SECURITY POLICY dbo.ProductAccessPolicy ADD BLOC PREDICATE dbo.pUserCanAccessProduct (CompanyID) ON dbo.Product

Ce prédicat sera appliqué à toutes les opérations. Si vous voulez appliquer des prédicats sur certaines opérations, vous pouvez écrire quelque chose comme:

CREATE SECURITY POLICY dbo.ProductAccessPolicy ADD BLOC PREDICATE dbo.pUserCanAccessProduct (ID de l'entreprise) ON dbo.Product AFTER INSERT

Les options possibles que vous pouvez ajouter après la définition du prédicat de bloc sont les suivantes:

[{APRÈS {INSERT | METTRE À JOUR } }
| {AVANT {MISE À JOUR | EFFACER } }]

Lire Sécurité au niveau des lignes en ligne: https://riptutorial.com/fr/sql-server/topic/7045/securite-au-niveau-des-lignes

Chapitre 96: Séquences

Examples

Créer une séquence

```
CREATE SEQUENCE [dbo].[CustomersSeq]
AS INT
START WITH 10001
INCREMENT BY 1
MINVALUE -1;
```

Utiliser la séquence dans la table

```
CREATE TABLE [dbo].[Customers]
(
    CustomerID INT DEFAULT (NEXT VALUE FOR [dbo].[CustomersSeq]) NOT NULL,
    CustomerName VARCHAR(100),
);
```

Insérer dans la table avec séquence

```
INSERT INTO [dbo].[Customers]
       ([CustomerName])
 VALUES
       ('Jerry'),
       ('Gorge')

SELECT * FROM [dbo].[Customers]
```

Résultats

N ° de client	Nom du client
10001	Jerry
10002	Gorge

Supprimer de et insérer un nouveau

```
DELETE FROM [dbo].[Customers]
WHERE CustomerName = 'Gorge';

INSERT INTO [dbo].[Customers]
       ([CustomerName])
 VALUES ('George')

SELECT * FROM [dbo].[Customers]
```

Résultats

N ° de client	Nom du client
10001	Jerry
10003	George

Lire Séquences en ligne: https://riptutorial.com/fr/sql-server/topic/5324/sequences

Chapitre 97: SINON

Examples

Déclaration IF unique

Comme la plupart des autres langages de programmation, T-SQL prend également en charge les instructions IF..ELSE.

Par exemple, dans l'exemple ci-dessous, `1 = 1` est l'expression qui correspond à True et le contrôle entre le bloc `BEGIN..END` et l'instruction Print imprime la chaîne `'One is equal to One'`

```
IF ( 1 = 1)  --<-- Some Expression
 BEGIN
   PRINT 'One is equal to One'
 END
```

Déclarations IF multiples

Nous pouvons utiliser plusieurs instructions IF pour vérifier plusieurs expressions totalement indépendantes les unes des autres.

Dans l'exemple ci-dessous, l'expression de chaque instruction `IF` est évaluée et, si elle est vraie, le code `BEGIN...END` dans le bloc `BEGIN...END` est exécuté. Dans cet exemple particulier, les expressions First et Third sont true et seules ces instructions print seront exécutées.

```
IF (1 = 1)  --<-- Some Expression      --<-- This is true
BEGIN
    PRINT 'First IF is True'           --<-- this will be executed
END

IF (1 = 2)  --<-- Some Expression
BEGIN
    PRINT 'Second IF is True'
END

IF (3 = 3)  --<-- Some Expression        --<-- This true
BEGIN
    PRINT 'Thrid IF is True'             --<-- this will be executed
END
```

Instruction IF..ELSE unique

Dans une seule instruction `IF..ELSE` , si l'expression est `IF..ELSE` sur True dans l'instruction `IF` , le contrôle entre le premier bloc `BEGIN..END` et seul le code `BEGIN..END` dans ce bloc est exécuté. Sinon, le bloc Else est simplement ignoré.

D'autre part, si l'expression est évaluée à `False` le bloc `ELSE BEGIN..END` est exécuté et le contrôle n'entre jamais dans le premier bloc `BEGIN..END` .

Dans l'exemple ci-dessous, l'expression sera évaluée à false et le bloc Else sera exécuté en imprimant la chaîne `'First expression was not true'`

```
IF ( 1 <> 1)  --<-- Some Expression
 BEGIN
     PRINT 'One is equal to One'
 END
ELSE
 BEGIN
     PRINT 'First expression was not true'
 END
```

Multiple IF ... ELSE avec les instructions ELSE finales

Si nous avons plusieurs instructions `IF...ELSE IF` mais que nous voulons également exécuter un morceau de code si aucune des expressions n'est évaluée à True, nous pouvons simplement ajouter un bloc `ELSE` final qui ne sera exécuté que si aucun des `IF` ou `ELSE IF` expressions sont évaluées à true.

Dans l'exemple ci-dessous, aucune des expressions `IF` ou `ELSE IF` n'est vraie. Par conséquent, seul le bloc `ELSE` est exécuté et affiche `'No other expression is true'`

```
IF ( 1 = 1 + 1 )
    BEGIN
       PRINT 'First If Condition'
    END
ELSE IF (1 = 2)
    BEGIN
        PRINT 'Second If Else Block'
    END
ELSE IF (1 = 3)
    BEGIN
        PRINT 'Third If Else Block'
    END
ELSE
    BEGIN
        PRINT 'No other expression is true'  --<-- Only this statement will be printed
    END
```

Instructions IF multiples ... ELSE

Le plus souvent, nous devons vérifier plusieurs expressions et prendre des mesures spécifiques en fonction de ces expressions. Cette situation est gérée à l'aide de plusieurs instructions `IF...ELSE IF` .

Dans cet exemple, toutes les expressions sont évaluées de haut en bas. Dès qu'une expression est évaluée à true, le code à l'intérieur de ce bloc est exécuté. Si aucune expression n'est évaluée à true, rien n'est exécuté.

```
IF (1 = 1 + 1)
BEGIN
    PRINT 'First If Condition'
END
```

```
ELSE IF (1 = 2)
BEGIN
    PRINT 'Second If Else Block'
END
ELSE IF (1 = 3)
BEGIN
    PRINT 'Third If Else Block'
END
ELSE IF (1 = 1)      --<-- This is True
BEGIN
    PRINT 'Last Else Block'  --<-- Only this statement will be printed
END
```

Lire SINON en ligne: https://riptutorial.com/fr/sql-server/topic/5186/sinon

Chapitre 98: Sous-requêtes

Examples

Sous-requêtes

Une sous-requête est une requête dans une autre requête SQL. Une sous-requête est également appelée requête interne ou sélection interne et l'instruction contenant une sous-requête est appelée requête externe ou sélection externe.

Remarque

1. Les sous-requêtes doivent être placées entre parenthèses,
2. Un ORDER BY ne peut pas être utilisé dans une sous-requête.
3. Le type d'image, par exemple BLOB, tableau, types de données texte, n'est pas autorisé dans les sous-requêtes.

Les sous-requêtes peuvent être utilisées avec les instructions select, insert, update et delete dans la clause where, from, select avec IN, les opérateurs de comparaison, etc.

Nous avons une table nommée ITCompanyInNepal sur laquelle nous allons effectuer des requêtes pour montrer des exemples de sous-requêtes:

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyOne	Kathmandu	USA	350
2	CompanyTwo	Kathmandu	USA	310
3	CompanyThree	Kathmandu	Nepal	300
4	CompanyFour	Kathmandu	Nepal	180
5	CompanyFive	Birgunj	Denmark	150
6	CompanySix	Janakpur	USA	100
7	CompanySeven	Janakpur	Australia	100
8	CompanyEight	Birganj	Australia	150
9	CompanyNine	Biratnagar	Canada	200
10	CompanyTen	Pokhara	India	85

Exemples: **SubQueries With Select Statement**

avec l'opérateur **In** et la clause **where** :

```
SELECT *
FROM ITCompanyInNepal
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'USA');
```

avec **opérateur de comparaison** et **où** clause

```
SELECT *
FROM ITCompanyInNepal
```

```
WHERE NumberOfEmployee < (SELECT AVG(NumberOfEmployee)
                          FROM ITCompanyInNepal
                        )
```

avec la clause **select**

```
SELECT   CompanyName,
         CompanyAddress,
         Headquarter,
         (Select SUM(NumberOfEmployee)
         FROM ITCompanyInNepal
         Where Headquarter = 'USA') AS TotalEmployeeHiredByUSAInKathmandu
FROM     ITCompanyInNepal
WHERE    CompanyAddress = 'Kathmandu' AND Headquarter = 'USA'
```

Sous-requêtes avec instruction insert

Nous devons insérer des données de la table IndianCompany dans ITCompanyInNepal. Le tableau pour IndianCompany est présenté ci-dessous:

ID	CompanyName	CompanyAddress	Headquarter	NumberOfEmployee
1	CompanyA	Banglore	USA	450
2	CompanyB	Banglore	USA	500
3	CompanyC	Hyderabad	Denmark	480
4	CompanyD	Hyderabad	Australia	780
5	CompanyE	Delhi	Canada	790

```
INSERT INTO ITCompanyInNepal
SELECT *
FROM IndianCompany
```

Sous-requêtes avec instruction de mise à jour

Supposons que toutes les entreprises dont le siège est aux États-Unis ont décidé de licencier 50 employés de toutes les sociétés américaines du Népal en raison d'un changement de politique des sociétés américaines.

```
UPDATE ITCompanyInNepal
SET NumberOfEmployee = NumberOfEmployee - 50
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'USA')
```

Sous-requêtes avec instruction de suppression

Supposons que toutes les entreprises dont le siège est au Danemark ont décidé de fermer leurs entreprises au Népal.

```
DELETE FROM ITCompanyInNepal
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'Denmark')
```

Lire Sous-requêtes en ligne: https://riptutorial.com/fr/sql-server/topic/5629/sous-requetes

Chapitre 99: SQL dynamique

Examples

Exécuter l'instruction SQL fournie sous forme de chaîne

Dans certains cas, vous devrez exécuter une requête SQL placée dans une chaîne. EXEC, EXECUTE ou la procédure système sp_executesql peut exécuter toute requête SQL fournie sous forme de chaîne:

```
sp_executesql N'SELECT * FROM sys.objects'
-- or
sp_executesql @stmt = N'SELECT * FROM sys.objects'
-- or
EXEC sp_executesql N'SELECT * FROM sys.objects'
-- or
EXEC('SELECT * FROM sys.columns')
-- or
EXECUTE('SELECT * FROM sys.tables')
```

Cette procédure renvoie le même jeu de résultats que la requête SQL fournie en tant que texte d'instruction. sp_executesql peut exécuter une requête SQL fournie sous forme de chaîne, de variable / paramètre ou même d'expression:

```
declare @table nvarchar(40) = N'product items'
EXEC(N'SELECT * FROM ' + @table)
declare @sql nvarchar(40) = N'SELECT * FROM ' + QUOTENAME(@table);
EXEC sp_executesql @sql
```

Vous avez besoin de la fonction QUOTENAME pour échapper les caractères spéciaux dans la variable @table. Sans cette fonction, vous obtiendrez une erreur de syntaxe si la variable @table contient quelque chose comme des espaces, des crochets ou tout autre caractère spécial.

SQL dynamique exécuté en tant qu'utilisateur différent

Vous pouvez exécuter une requête SQL en tant qu'utilisateur différent en utilisant AS USER = 'nom de l'utilisateur de la base de données'

```
EXEC(N'SELECT * FROM product') AS USER = 'dbo'
```

La requête SQL sera exécutée sous l'utilisateur de la base de données dbo. Toutes les vérifications d'autorisation applicables à l'utilisateur dbo seront vérifiées sur la requête SQL.

Injection SQL avec SQL dynamique

Les requêtes dynamiques sont

```
SET @sql = N'SELECT COUNT(*) FROM AppUsers WHERE Username = ''' + @user + ''' AND Password =
''' + @pass + ''''
EXEC(@sql)
```

Si la valeur de la variable utilisateur est ***myusername '' OR 1 = 1 -*** la requête suivante sera exécutée:

```
SELECT COUNT(*)
FROM AppUsers
WHERE Username = 'myusername' OR 1=1 --' AND Password = ''
```

Le commentaire à la fin de la valeur de la variable @username fera un commentaire hors partie de la requête et la condition 1 = 1 sera évaluée. L'application qui la contrôle au moins un utilisateur renvoyé par cette requête renvoie un nombre supérieur à 0 et la connexion réussira.

En utilisant cette approche, l'attaquant peut se connecter à l'application même s'il ne connaît pas un nom d'utilisateur et un mot de passe valides.

SQL dynamique avec paramètres

Pour éviter les problèmes d’injection et d’échappement, les requêtes SQL dynamiques doivent être exécutées avec des paramètres, par exemple:

```
SET @sql = N'SELECT COUNT(*) FROM AppUsers WHERE Username = @user AND Password = @pass
EXEC sp_executesql @sql, '@user nvarchar(50), @pass nvarchar(50)', @username, @password
```

Le second paramètre est une liste de paramètres utilisés dans les requêtes avec leurs types, une fois que cette liste contient des variables qui seront utilisées comme valeurs de paramètre.

sp_executesql échappera aux caractères spéciaux et exécutera la requête SQL.

Lire SQL dynamique en ligne: https://riptutorial.com/fr/sql-server/topic/6871/sql-dynamique

Chapitre 100: SQL Server Evolution à travers différentes versions (2000 - 2016)

Introduction

J'utilise SQL Server depuis 2004. J'ai commencé avec 2000 et maintenant je vais utiliser SQL Server 2016. J'ai créé des tables, des vues, des fonctions, des déclencheurs, des procédures stockées et écrit de nombreuses requêtes SQL mais je n'ai pas utilisé beaucoup de nouvelles fonctionnalités. versions. Je l'ai googlé mais malheureusement, je n'ai pas trouvé toutes les fonctionnalités dans un seul endroit. J'ai donc rassemblé et validé ces informations à partir de différentes sources et les ai mises ici. J'ajoute juste les informations de haut niveau pour toutes les versions à partir de 2000 à 20

Examples

SQL Server Version 2000 - 2016

Les fonctionnalités suivantes ont été ajoutées à SQL Server 2000 à partir de sa version précédente:

1. De nouveaux types de données ont été ajoutés (BIGINT, SQL_VARIANT, TABLE)
2. Au lieu de et pour les déclencheurs ont été introduits comme avancement à la DDL.
3. Intégrité référentielle en cascade.
4. Support XML
5. Fonctions définies par l'utilisateur et vues de partition.
6. Vues indexées (autoriser l'index sur les vues avec colonnes calculées).

Les fonctionnalités suivantes ont été ajoutées à la version 2005 de sa version précédente:

1. Amélioration de la clause TOP avec l'option “WITH TIES”.
2. Commandes de manipulation de données (DML) et clause OUTPUT pour obtenir des valeurs INSERTED et DELETED
3. Les opérateurs PIVOT et UNPIVOT.
4. Gestion des exceptions avec le bloc TRY / CATCH
5. Fonctions de classement
6. Expressions de table communes (CTE)
7. Common Language Runtime (Intégration des langages .NET pour construire des objets tels que des procédures stockées, des déclencheurs, des fonctions, etc.)
8. Service Broker (Gestion du message entre un expéditeur et un destinataire de manière souple)
9. Chiffrement des données (fonctionnalités natives pour prendre en charge le chiffrement des données stockées dans des bases de données définies par l'utilisateur)
10. SMTP mail
11. Terminaux HTTP (création de points de terminaison à l'aide d'une simple instruction T-SQL

exposant un objet à accéder via Internet)
12. Plusieurs ensembles de résultats actifs (MARS). Cela permet à une connexion de base de données persistante à partir d'un seul client d'avoir plusieurs requêtes actives par connexion.
13. SQL Server Integration Services (Sera utilisé comme outil ETL (Extraction, transformation et chargement) principal)
14. Améliorations dans Analysis Services et Reporting Services.
15. Partitionnement de table et d'index. Permet le partitionnement des tables et des index en fonction des limites de la partition, comme spécifié par une FONCTION PARTITION avec des partitions individuelles mappées à des groupes de fichiers via un PARTITION SCHEME.

Les fonctionnalités suivantes ont été ajoutées à la version 2008 de sa version précédente:

1. Amélioration des types de données DATE et TIME existants
2. Nouvelles fonctions comme - SYSUTCDATETIME () et SYSDATETIMEOFFSET ()
3. Colonnes de rechange - Permet d'économiser une quantité importante d'espace disque.
4. Grands types définis par l'utilisateur (jusqu'à 2 Go)
5. Introduit une nouvelle fonctionnalité pour passer un type de données de table dans les procédures stockées et les fonctions
6. Nouvelle commande MERGE pour les opérations INSERT, UPDATE et DELETE
7. Nouveau type de données HierarchyID
8. Types de données spatiales - Représentent l'emplacement physique et la forme de tout objet géométrique.
9. Requêtes et rapports plus rapides avec GROUPING SETS - Extension de la clause GROUP BY.
10. Amélioration de l'option de stockage FILESTREAM

Les fonctionnalités suivantes ont été ajoutées à la version 2008 R2 de sa version précédente:

1. PowerPivot - Pour traiter de grands ensembles de données.
2. Générateur de rapports version 3.0
3. Cloud prêt
4. StreamInsight
5. Master Data Services
6. Intégration SharePoint
7. DACPAC (ensembles de composants d'application de niveau données)
8. Amélioration des autres fonctionnalités de SQL Server 2008

Les fonctionnalités suivantes ont été ajoutées à la version 2012 de sa version précédente:

1. Index de magasin de colonnes: réduit l'utilisation des E / S et de la mémoire dans les requêtes volumineuses.
2. Pagination - La pagination peut être effectuée en utilisant les commandes «OFFSET» et «FETCH».
3. Base de données contenue - Excellente fonctionnalité pour les migrations de données périodiques.
4. Groupes de disponibilité AlwaysOn

5. Prise en charge de Windows Server Core
6. Rôles de serveur définis par l'utilisateur
7. Big Data Support
8. PowerView
9. Améliorations de SQL Azure
10. Modèle tabulaire (SSAS)
11. Services de qualité des données DQS
12. File Table - une amélioration de la fonctionnalité FILESTREAM introduite en 2008.
13. Amélioration de la gestion des erreurs, y compris l'instruction THROW
14. Amélioration du débogage de SQL Server Management Studio a. SQL Server 2012 introduit davantage d'options pour contrôler les points d'arrêt. b. Améliorations apportées aux fenêtres en mode débogage
    c. Amélioration d'IntelliSense - comme l'insertion d'extraits de code.

Les fonctionnalités suivantes ont été ajoutées dans la version 2014 de sa version précédente:

1. Moteur OLTP en mémoire - Améliore les performances jusqu'à 20 fois.
2. Améliorations AlwaysOn
3. Extension du pool de mémoire tampon
4. Caractéristiques du cloud hybride
5. Amélioration des index de magasin de colonnes (comme les index de magasin de colonnes pouvant être mis à jour)
6. Améliorations du traitement des requêtes (comme SELECT INTO parallèle)
7. Intégration de Power BI pour Office 365
8. Durabilité différée
9. Améliorations pour les sauvegardes de base de données

Les fonctionnalités suivantes ont été ajoutées à la version 2016 de sa version précédente:

1. Always Encrypted - Always Encrypted est conçu pour protéger les données au repos ou en mouvement.
2. Analyse opérationnelle en temps réel
3. PolyBase dans SQL Server
4. Prise en charge JSON native
5. Magasin de requêtes
6. Améliorations apportées à AlwaysOn
7. OLTP amélioré en mémoire
8. Plusieurs fichiers de base de données TempDB
9. Stretch Database
10. Sécurité au niveau des lignes
11. Améliorations en mémoire

    Améliorations de T-SQL ou ajouts dans SQL Server 2016

1. TABLEAU TRUNCATE avec PARTITION

2. DROP SI EXISTE

3. Fonctions STRING_SPLIT et STRING_ESCAPE
4. ALTER TABLE peut maintenant modifier plusieurs colonnes alors que la table reste en ligne, en utilisant WITH (ONLINE = ON | OFF).
5. MAXDOP pour DBCC CHECKDB, DBCC CHECKTABLE et DBCC CHECKFILEGROUP
6. ALTER DATABASE SET AUTOGROW_SINGLE_FILE
7. ALTER DATABASE SET AUTOGROW_ALL_FILES
8. Fonctions COMPRESSE et DECOMPRESS
9. Déclaration FORMATMESSAGE
10. 2016 introduit 8 propriétés supplémentaires avec SERVERPROPERTY

une. InstanceDefaultDataPath

b. InstanceDefaultLogPath

c. ProductBuild

ré. ProductBuildType

e. ProductMajorVersion

F. ProductMinorVersion

g. ProductUpdateLevel

h. ProductUpdateReference

Lire SQL Server Evolution à travers différentes versions (2000 - 2016) en ligne: https://riptutorial.com/fr/sql-server/topic/10129/sql-server-evolution-a-travers-differentes-versions--2000---2016-

Chapitre 101: SQL Server Management Studio (SSMS)

Introduction

SQL Server Management Studio (SSMS) est un outil permettant de gérer et d'administrer SQL Server et la base de données SQL.

SSMS est offert gratuitement par Microsoft.

La documentation SSMS est disponible.

Examples

Actualiser le cache IntelliSense

Lorsque des objets sont créés ou modifiés, ils ne sont pas automatiquement disponibles pour IntelliSense. Pour les mettre à la disposition d'IntelliSense, le cache local doit être actualisé.

Dans une fenêtre d'éditeur de requêtes, appuyez sur `Ctrl + Shift + R` ou sélectionnez `Edit | IntelliSense | Refresh Local Cache` dans le menu.

Après cela, tous les changements depuis la dernière actualisation seront disponibles pour IntelliSense.

Lire SQL Server Management Studio (SSMS) en ligne: https://riptutorial.com/fr/sql-server/topic/10642/sql-server-management-studio--ssms-

Chapitre 102: SQLCMD

Remarques

Vous devez soit être dans le chemin où existe SQLCMD.exe ou l'ajouter à votre variable d'environnement PATH.

Examples

SQLCMD.exe appelé à partir d'un fichier de commandes ou d'une ligne de commande

```
echo off

cls

sqlcmd.exe -S "your server name" -U "sql user name" -P "sql password" -d "name of databse" -Q
"here you may write your query/stored procedure"
```

Les fichiers batch tels que ceux-ci peuvent être utilisés pour automatiser des tâches, par exemple pour effectuer des sauvegardes de bases de données à une heure spécifiée (peut être planifiée avec le Planificateur de tâches) pour une version SQL Server Express où les jobs d'agent ne peuvent pas être utilisés.

Lire SQLCMD en ligne: https://riptutorial.com/fr/sql-server/topic/5396/sqlcmd

Chapitre 103: Stocker JSON dans les tables SQL

Examples

JSON stocké en colonne de texte

JSON est un format textuel, il est donc stocké dans des colonnes NVARCHAR standard. La collection NoSQL est équivalente à la table de valeurs de clé à deux colonnes:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max)
)
```

Utilisez `nvarchar(max)` car vous ne savez pas quelle serait la taille de vos documents JSON. `nvarchar(4000)` et `varchar(8000)` ont de meilleures performances mais avec une taille limite à 8 Ko.

Assurez-vous que JSON est correctement formaté avec ISJSON

Puisque JSON est stocké dans la colonne textuelle, vous pouvez vous assurer qu'il est correctement formaté. Vous pouvez ajouter la contrainte CHECK sur la colonne JSON qui vérifie que le texte est correctement formaté JSON:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max)
       CONSTRAINT [Data should be formatted as JSON]
       CHECK (ISJSON(Data) > 0)
)
```

Si vous avez déjà une table, vous pouvez ajouter une contrainte de vérification à l'aide de l'instruction ALTER TABLE:

```
ALTER TABLE ProductCollection
    ADD CONSTRAINT [Data should be formatted as JSON]
        CHECK (ISJSON(Data) > 0)
```

Exposer les valeurs du texte JSON en tant que colonnes calculées

Vous pouvez exposer les valeurs de la colonne JSON en tant que colonnes calculées:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max),
  Price AS JSON_VALUE(Data, '$.Price'),
  Color JSON_VALUE(Data, '$.Color') PERSISTED
```

```
)
```

Si vous ajoutez une colonne calculée PERSISTED, la valeur du texte JSON sera matérialisée dans cette colonne. De cette façon, vos requêtes peuvent lire plus rapidement la valeur du texte JSON car aucune analyse n'est nécessaire. Chaque fois que JSON dans cette ligne change, la valeur sera recalculée.

Ajout d'index sur le chemin JSON

Les requêtes qui filtrent ou trient les données avec une certaine valeur dans la colonne JSON utilisent généralement l'analyse complète de la table.

```
SELECT * FROM ProductCollection
WHERE JSON_VALUE(Data, '$.Color') = 'Black'
```

Pour optimiser ce type de requêtes, vous pouvez ajouter une colonne calculée non persistante qui expose une expression JSON utilisée dans un filtre ou un tri (dans cet exemple, JSON_VALUE (Data, '$ .Color')) et créez un index sur cette colonne:

```
ALTER TABLE ProductCollection
ADD vColor as JSON_VALUE(Data, '$.Color')

CREATE INDEX idx_JsonColor
ON ProductCollection(vColor)
```

Les requêtes utiliseront l'index au lieu de l'analyse de table simple.

JSON stocké dans des tables en mémoire

Si vous pouvez utiliser des tables optimisées pour la mémoire, vous pouvez stocker JSON sous forme de texte:

```
CREATE TABLE ProductCollection (
  Id int identity primary key nonclustered,
  Data nvarchar(max)
) WITH (MEMORY_OPTIMIZED=ON)
```

Avantages de JSON en mémoire:

- Les données JSON sont toujours en mémoire, il n'y a donc pas d'accès au disque
- Il n'y a pas de verrou ni de verrou lors de l'utilisation de JSON

Lire Stocker JSON dans les tables SQL en ligne: https://riptutorial.com/fr/sql-server/topic/5029/stocker-json-dans-les-tables-sql

Chapitre 104: SYNDICAT

Examples

Union et syndicat tous

L' opération d' **union** combine les résultats de deux requêtes ou plus en un seul ensemble de résultats qui inclut toutes les lignes appartenant à toutes les requêtes de l'union et ignore les doublons existants. **Union fait** aussi la même chose mais inclut même les valeurs en double. Le concept d'opération syndicale sera clair à partir de l'exemple ci-dessous. Peu de choses à considérer en utilisant l'union sont:

1.Le nombre et l'ordre des colonnes doivent être identiques dans toutes les requêtes.

2. Les types de données doivent être compatibles.

Exemple:

Nous avons trois tableaux: Marksheet1, Marksheet2 et Marksheet3. Marksheet3 est la table en double de Marksheet2 qui contient les mêmes valeurs que celle de Marksheet2.

Tableau 1 : Feuille de marquage1

SubjectCode	SubjectName	MarksObtained
101	Physics	87
102	Chemistry	75
103	Maths	85
104	English	89
105	Computer	95

Tableau 2: Marksheet2

CourseCode	CourseName	MarksObtained
201	PhysicsII	82
202	ChemistryII	86
203	MathsII	95
204	EnglishII	70
205	ComputerII	86

Tableau 3 : Feuille de calcul3

SubjectCode	SubjectName	MarksObtained
201	PhysicsII	82
202	ChemistryII	86
203	MathsII	95
204	EnglishII	70
205	ComputerII	86

Union sur les tables Marksheet1 et Marksheet2

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
```

Remarque: La sortie pour l'union des trois tables sera également identique à celle de l'union sur Marksheet1 et Marksheet2 car l'opération d'union ne prend pas les valeurs en double.

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

SORTIE

	SubjectCode	SubjectName	MarksObtained
1	101	Physics	87
2	102	Chemistry	75
3	103	Maths	85
4	104	English	89
5	105	Computer	95
6	201	PhysicsII	82
7	202	ChemistryII	86
8	203	MathsII	95
9	204	EnglishII	70
10	205	ComputerII	86

Union Tous

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION ALL
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION ALL
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

SORTIE

	SubjectCode	SubjectName	MarksObtained
1	101	Physics	87
2	102	Chemistry	75
3	103	Maths	85
4	104	English	89
5	105	Computer	95
6	201	PhysicsII	82
7	202	ChemistryII	86
8	203	MathsII	95
9	204	EnglishII	70
10	205	ComputerII	86
11	201	PhysicsII	82
12	202	ChemistryII	86
13	203	MathsII	95
14	204	EnglishII	70
15	205	ComputerII	86

Vous remarquerez ici que les doublons de Marksheet3 sont également affichés en utilisant tous les union.

Lire SYNDICAT en ligne: https://riptutorial.com/fr/sql-server/topic/5590/syndicat

Chapitre 105: Tables Temporelles

Remarques

SQL Server 2016 prend en charge les tables temporelles versionnées par le système en tant que fonctionnalité de base de données qui prend en charge la fourniture d'informations sur les données stockées dans la table à tout moment et non uniquement aux données correctes au moment présent.

Une table temporelle versionnée par le système est un nouveau type de table utilisateur dans SQL Server 2016, conçue pour conserver un historique complet des modifications de données et permettre une analyse ponctuelle facile. Ce type de table temporelle est appelé table temporelle versionnée par le système car la période de validité de chaque ligne est gérée par le système (c.-à-d. Le moteur de base de données). Chaque table temporelle a deux colonnes explicitement définies, chacune avec un type de données datetime2. Ces colonnes sont appelées colonnes de période. Ces colonnes de période sont utilisées exclusivement par le système pour enregistrer la période de validité de chaque ligne chaque fois qu'une ligne est modifiée.

Examples

CREATE Tables Temporelles

```
CREATE TABLE dbo.Employee
(
  [EmployeeID] int NOT NULL PRIMARY KEY CLUSTERED
  , [Name] nvarchar(100) NOT NULL
  , [Position] varchar(100) NOT NULL
  , [Department] varchar(100) NOT NULL
  , [Address] nvarchar(1024) NOT NULL
  , [AnnualSalary] decimal (10,2) NOT NULL
  , [ValidFrom] datetime2 (2) GENERATED ALWAYS AS ROW START
  , [ValidTo] datetime2 (2) GENERATED ALWAYS AS ROW END
  , PERIOD FOR SYSTEM_TIME (ValidFrom, ValidTo)
 )
 WITH (SYSTEM_VERSIONING = ON (HISTORY_TABLE = dbo.EmployeeHistory));
```

INSERTS: sur un **INSERT** , le système définit la valeur de la colonne **ValidFrom** sur l'heure de début de la transaction en cours (dans le fuseau horaire UTC) en fonction de l'horloge système et affecte la valeur de la colonne **ValidTo** à la valeur maximale de 9999- 12-31. Cela marque la ligne comme ouverte.

MISES À **JOUR:** Sur un **UPDATE,** le système stocke la valeur précédente de la ligne dans la table d'historique et définit la valeur de la colonne **validTo** au temps de début de la transaction en cours (dans le temps UTC zone) en fonction de l'horloge système. Cela marque la ligne comme fermée, avec une période enregistrée pour laquelle la ligne était valide. Dans la table en cours, la ligne est mise à jour avec sa nouvelle valeur et le système définit la valeur de la colonne **ValidFrom** sur l'heure de début de la transaction (dans le fuseau horaire UTC) en fonction de

l'horloge système. La valeur de la ligne mise à jour dans la table en cours pour la colonne **ValidTo** reste la valeur maximale de 9999-12-31.

DELETES : sur un **DELETE** , le système stocke la valeur précédente de la ligne dans la table d'historique et définit la valeur de la colonne **ValidTo** sur l'heure de début de la transaction en cours (dans le fuseau horaire UTC) en fonction de l'horloge système. Cela marque la ligne comme fermée, avec une période enregistrée pour laquelle la ligne précédente était valide. Dans la table en cours, la ligne est supprimée. Les requêtes de la table en cours ne renverront pas cette ligne. Seules les requêtes qui traitent des données d'historique renvoient des données pour lesquelles une ligne est fermée.

FUSION : sur un **MERGE** , l'opération se comporte exactement comme si jusqu'à trois instructions (un **INSERT** , un **UPDATE** et / ou un **DELETE**) **étaient** exécutées, en fonction des actions spécifiées dans l'instruction **MERGE** .

Conseil: Les heures enregistrées dans les colonnes système datetime2 sont basées sur l'heure de début de la transaction elle-même. Par exemple, toutes les lignes insérées dans une seule transaction auront la même heure UTC enregistrée dans la colonne correspondant au début de la période **SYSTEM_TIME** .

Comment interroger des données temporelles?

```
SELECT * FROM Employee
    FOR SYSTEM_TIME
        BETWEEN '2014-01-01 00:00:00.0000000' AND '2015-01-01 00:00:00.0000000'
            WHERE EmployeeID = 1000 ORDER BY ValidFrom;
```

Renvoie la valeur réelle spécifiée dans le temps (FOR SYSTEM_TIME AS OF)

Renvoie une table avec des lignes contenant les valeurs réelles (actuelles) au moment spécifié dans le passé.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME AS  OF '2016-08-06 08:32:37.91'
```

POUR SYSTEM_TIME ENTRE ET

Comme ci-dessus dans la description FOR SYSTEM_TIME FROM <start_date_time> TO <end_date_time>, sauf que la table de lignes renvoyée comprend des lignes devenues actives sur la limite supérieure définie par le noeud final <end_date_time>.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME BETWEEN  '2015-01-01' AND '2015-12-31'
```

POUR SYSTEM_TIME DE À

Renvoie une table avec les valeurs de toutes les versions de ligne actives dans la plage de temps spécifiée, qu'elles aient ou non commencé avant la valeur du paramètre <start_date_time> pour

l'argument FROM ou qu'elles aient cessé d'être actives après la valeur du paramètre <end_date_time> TO argument. En interne, une union est effectuée entre la table temporelle et sa table d'historique et les résultats sont filtrés pour renvoyer les valeurs de toutes les versions de lignes actives à tout moment au cours de la période spécifiée. Les lignes devenues actives exactement sur la limite inférieure définie par le noeud final FROM sont incluses et les enregistrements devenus actifs exactement sur la limite supérieure définie par le noeud final TO ne sont pas inclus.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME FROM '2015-01-01' TO '2015-12-31'
```

POUR SYSTEM_TIME CONTENU DANS (,)

Renvoie une table avec les valeurs de toutes les versions de lignes ouvertes et fermées dans la plage de temps spécifiée définie par les deux valeurs datetime pour l'argument CONTAINED IN. Les lignes devenues actives exactement à la limite inférieure ou cessant d'être actives exactement à la limite supérieure sont incluses.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME CONTAINED IN ('2015-04-01', '2015-09-25')
```

POUR SYSTEM_TIME ALL

Renvoie l'union des lignes qui appartiennent à la table courante et à la table d'historique.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME ALL
```

Création d'une table temporelle versionnée par le système et optimisée pour la mémoire et nettoyage de la table d'historique SQL Server

La création d'une table temporelle avec une table d'historique par défaut est une option pratique lorsque vous souhaitez contrôler la dénomination et que vous comptez toujours sur le système pour créer la table d'historique avec la configuration par défaut. Dans l'exemple ci-dessous, une nouvelle table temporelle optimisée en fonction de la mémoire du système est associée à une nouvelle table d'historique sur disque.

```
CREATE SCHEMA History
GO
CREATE TABLE dbo.Department
(
    DepartmentNumber char(10) NOT NULL PRIMARY KEY NONCLUSTERED,
    DepartmentName varchar(50) NOT NULL,
    ManagerID int  NULL,
    ParentDepartmentNumber char(10) NULL,
    SysStartTime datetime2 GENERATED ALWAYS AS ROW START HIDDEN NOT NULL,
    SysEndTime datetime2 GENERATED ALWAYS AS ROW END HIDDEN NOT NULL,
    PERIOD FOR SYSTEM_TIME (SysStartTime,SysEndTime)
)
```

```
WITH
    (
        MEMORY_OPTIMIZED = ON, DURABILITY = SCHEMA_AND_DATA,
        SYSTEM_VERSIONING = ON ( HISTORY_TABLE = History.DepartmentHistory )
    );
```

Nettoyage de la table d'historique SQL Server Au fil du temps, la table d'historique peut augmenter considérablement. Étant donné que l'insertion, la mise à jour ou la suppression des données de la table d'historique ne sont pas autorisées, la seule façon de nettoyer la table d'historique est de désactiver le contrôle de version du système:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = OFF); ALLER

Supprimez les données inutiles de la table d'historique:

```
    DELETE FROM dbo.EmployeeHistory
```

WHERE EndTime <= '2017-01-26 14:00:29';

puis réactivez le contrôle de version du système:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = [dbo]. [EmployeeHistory], DATA_CONSISTENCY_CHECK = ON));

Le nettoyage de la table d'historique dans les bases de données SQL Azure est un peu différent, car les bases de données SQL Azure ont une prise en charge intégrée pour le nettoyage de la table d'historique. Tout d'abord, le nettoyage de la conservation de l'historique temporel doit être activé au niveau de la base de données:

```
ALTER DATABASE CURRENT
```

SET TEMPORAL_HISTORY_RETENTION SUR GO

Ensuite, définissez la période de rétention par table:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = ON (HISTORY_RETENTION_PERIOD = 90 JOURS));

Cela supprimera toutes les données de la table d'historique de plus de 90 jours. Les bases de données SQL Server 2016 sur site ne prennent pas en charge TEMPORAL_HISTORY_RETENTION et HISTORY_RETENTION_PERIOD et l'une des deux requêtes ci-dessus est exécutée sur les bases de données SQL Server 2016 sur site. Les erreurs suivantes se produisent.

Pour TEMPORAL_HISTORY_RETENTION, l'erreur sera:

```
Msg 102, Level 15, State 6, Line 34
```

Syntaxe incorrecte près de 'TEMPORAL_HISTORY_RETENTION'.

Pour HISTORY_RETENTION_PERIOD, l'erreur sera:

```
Msg 102, Level 15, State 1, Line 39
```

Syntaxe incorrecte près de 'HISTORY_RETENTION_PERIOD'.

Lire Tables Temporelles en ligne: https://riptutorial.com/fr/sql-server/topic/5296/tables-temporelles

Chapitre 106: Tâche ou tâche programmée

Introduction

SQL Server Agent utilise SQL Server pour stocker les informations de travail. Les travaux contiennent une ou plusieurs étapes du travail. Chaque étape contient sa propre tâche, à savoir: sauvegarder une base de données. SQL Server Agent peut exécuter un travail selon un calendrier, en réponse à un événement spécifique ou à la demande.

Examples

Créer un travail planifié

Créer un emploi

- Pour ajouter un travail, nous devons d'abord utiliser une procédure stockée nommée sp_add_job

```
USE msdb ;
GO
EXEC dbo.sp_add_job
@job_name = N'Weekly Job' ;  -- the job name
```

- Ensuite, nous devons ajouter une étape de travail en utilisant une procédure stockée nommée sp_add_jobStep

```
EXEC sp_add_jobstep
@job_name = N'Weekly Job',  -- Job name to add a step
@step_name = N'Set database to read only',  -- step name
@subsystem = N'TSQL',  -- Step type
@command = N'ALTER DATABASE SALES SET READ_ONLY',   -- Command
@retry_attempts = 5,  --Number of attempts
@retry_interval = 5 ; -- in minutes
```

- Ciblez le travail sur un serveur

```
EXEC dbo.sp_add_jobserver
@job_name = N'Weekly Sales Data Backup',
@server_name = 'MyPC\data;   -- Default is LOCAL
GO
```

Créer un planning à l'aide de SQL

Pour créer une planification, nous devons utiliser une procédure stockée système appelée sp_add_schedule

```
USE msdb
GO
```

```
EXEC sp_add_schedule
    @schedule_name = N'NightlyJobs' ,  -- specify the schedule name
    @freq_type = 4,   -- A value indicating when a job is to be executed (4) means Daily
    @freq_interval = 1,  -- The days that a job is executed and depends on the value of
`freq_type`.
    @active_start_time = 010000 ;   -- The time on which execution of a job can begin
GO
```

Il y a plus de paramètres qui peuvent être utilisés avec `sp_add_schedule` vous pouvez en savoir plus sur le lien ci-dessus.

Joindre un planning à un job

Pour associer une planification à un travail d'agent SQL, vous devez utiliser une procédure stockée appelée sp_attach_schedule

```
-- attaches the schedule to the job BackupDatabase
EXEC sp_attach_schedule
   @job_name = N'BackupDatabase',  -- The job name to attach with
   @schedule_name = N'NightlyJobs' ;  -- The schedule name
GO
```

Lire Tâche ou tâche programmée en ligne: https://riptutorial.com/fr/sql-server/topic/5329/tache-ou-tache-programmee

Chapitre 107: Touches de raccourci Microsoft SQL Server Management Studio

Examples

Exemples de raccourcis

1. Ouvrir une nouvelle fenêtre de requête avec la connexion en cours (`Ctrl` + `N`)
2. Basculer entre les onglets ouverts (`Ctrl` + `Tab`)
3. Afficher / Masquer le volet des résultats (`Ctrl` + `R`)
4. Exécuter la requête en surbrillance (`Ctrl` + `E`)
5. Rendre le texte sélectionné en majuscule ou en minuscule (`Ctrl` + `Maj` + `U` , `Ctrl` + `Maj` + `L`)
6. Membre de la liste Intellisense et mot complet (`Ctrl` + `Espace` , `Tab`)
7. Aller à la ligne (`Ctrl` + `G`)
8. fermer un onglet dans SQL Server Managament Studio (`Ctrl` + `F4`)

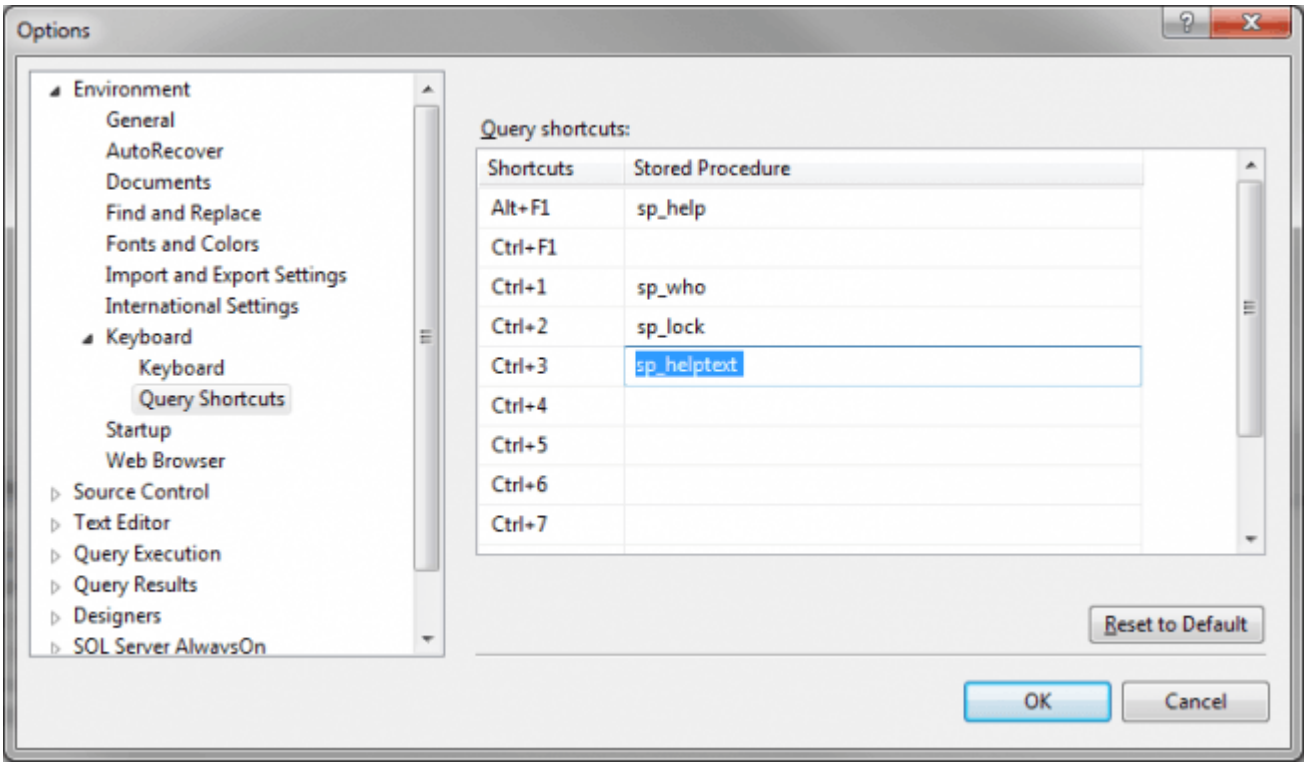
Raccourcis clavier d'activation de menu

1. Déplacer vers la barre de menus de SQL Server Management Studio (`ALT`)
2. Activer le menu pour un composant outil (`ALT` + `HYPHEN`)
3. Afficher le menu contextuel (`MAJ` + `F`)
4. Afficher la boîte de dialogue Nouveau fichier pour créer un fichier (`CTRL` + `N`)
5. Afficher la boîte de dialogue Ouvrir un projet pour ouvrir un projet existant (`CTRL` + `MAJ` + `O`)
6. Afficher la boîte de dialogue Ajouter un nouvel élément pour ajouter un nouveau fichier au projet en cours (`CTRL` + `MAJ` + `A`)
7. Afficher la boîte de dialogue Ajouter un élément existant pour ajouter un fichier existant au projet en cours (`CTRL` + `MAJ` + `A`)
8. Afficher le concepteur de requêtes (`CTRL` + `MAJ` + `Q`)
9. Fermer un menu ou une boîte de dialogue, annulant l'action (`ESC`)

Raccourcis clavier personnalisés

Allez dans Outils -> Options. Allez dans Environnement -> Clavier -> Raccourcis de requête

Sur le côté droit, vous pouvez voir des raccourcis par défaut dans SSMS. Maintenant, si vous devez en ajouter un nouveau, cliquez simplement sur une colonne sous la colonne Procédure stockée.



Cliquez sur OK. Maintenant, s'il vous plaît aller à une fenêtre de requête et sélectionnez la procédure stockée puis appuyez sur CTRL + 3, il affichera le résultat de la procédure stockée.

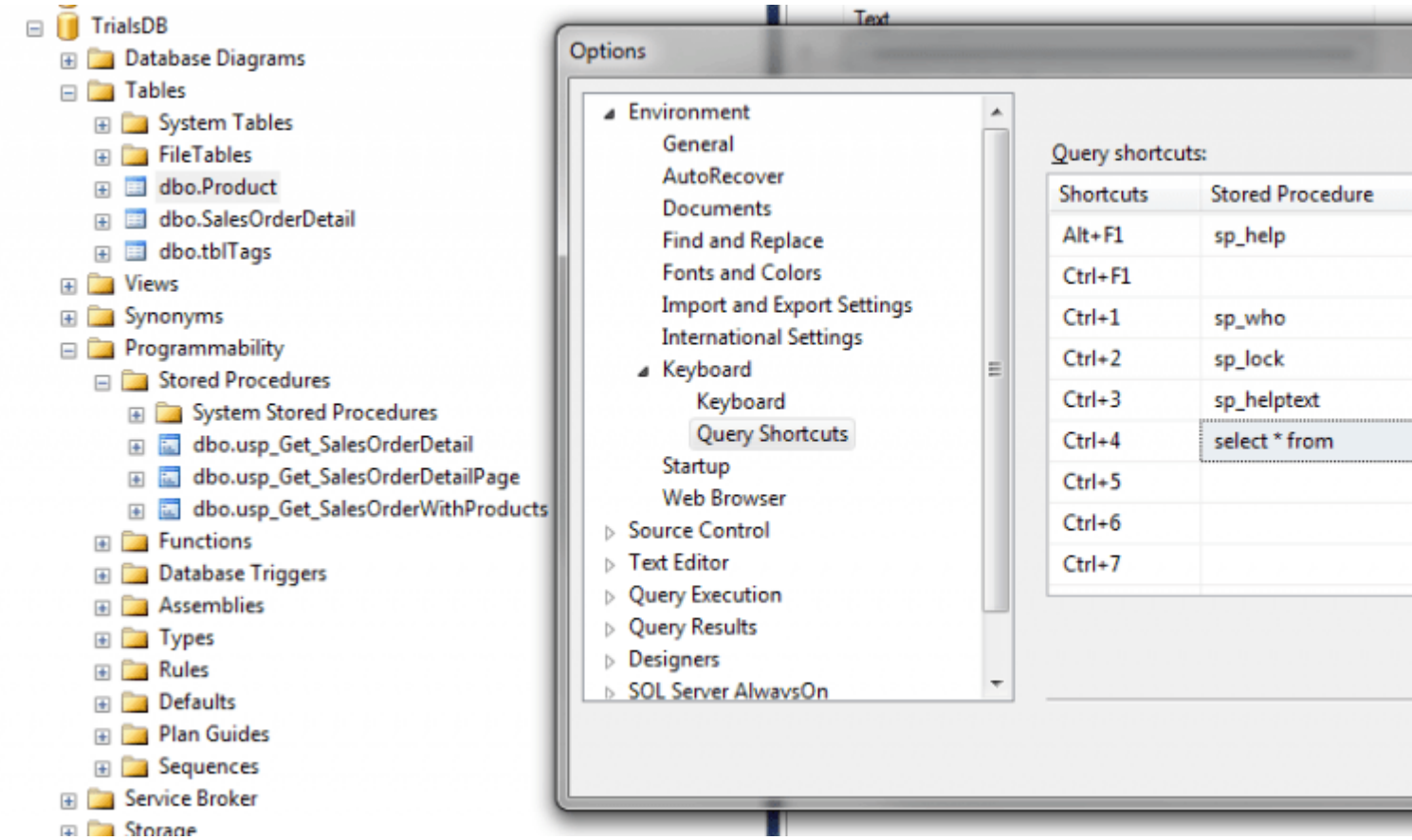
```
usp_Get_SalesOrderDetail
```

100 %

Results | Messages

	Text
1	-- ==
2	-- Author: <Author,Sibeesh Venu>
3	-- Create date: <Create Date, 18-Feb-2016>
4	-- Description: <Description,To fetch SalesOrderDetail>
5	-- ==
6	CREATE PROCEDURE [dbo].[usp_Get_SalesOrderDetail]
7	AS
8	BEGIN
9	-- SET NOCOUNT ON added to prevent extra result sets ...
10	-- interfering with SELECT statements.
11	SET NOCOUNT ON;
12	
13	-- Select statements for procedure here
14	SELECT top(100) SalesOrderID,SalesOrderDetailID,Carri...
15	END

Maintenant, si vous devez sélectionner tous les enregistrements d'une table lorsque vous sélectionnez la table et appuyez sur CTRL + 5 (vous pouvez sélectionner n'importe quelle touche). Vous pouvez créer le raccourci comme suit.



Maintenant, allez-y et sélectionnez le nom de la table dans la fenêtre de requête et appuyez sur CTRL + 4 (la clé que nous avons sélectionnée), cela vous donnera le résultat.

Lire Touches de raccourci Microsoft SQL Server Management Studio en ligne: https://riptutorial.com/fr/sql-server/topic/7749/touches-de-raccourci-microsoft-sql-server-management-studio

Chapitre 108: Traitement des transactions

Paramètres

Paramètre	Détails
transaction_name	pour nommer votre transaction - utile avec le paramètre [*avec marque*] qui permettra une journalisation significative - sensible à la casse (!)
avec marque ['description']	peut être ajouté à [*transaction_name*] et stockera une marque dans le journal

Examples

squelette de transaction de base avec gestion des erreurs

```
BEGIN TRY -- start error handling
    BEGIN TRANSACTION; -- from here on transactions (modifictions) are not final
        -- start your statement(s)
        select 42/0 as ANSWER  -- simple SQL Query with an error
        -- end your statement(s)
    COMMIT TRANSACTION; -- finalize all transactions (modifications)
END TRY   -- end error handling -- jump to end
BEGIN CATCH -- execute this IF an error occured
        ROLLBACK TRANSACTION; -- undo any transactions (modifications)
-- put together some information as a query
    SELECT
        ERROR_NUMBER() AS ErrorNumber
        ,ERROR_SEVERITY() AS ErrorSeverity
        ,ERROR_STATE() AS ErrorState
        ,ERROR_PROCEDURE() AS ErrorProcedure
        ,ERROR_LINE() AS ErrorLine
        ,ERROR_MESSAGE() AS ErrorMessage;

END CATCH;  -- final line of error handling
GO -- execute previous code
```

Lire Traitement des transactions en ligne: https://riptutorial.com/fr/sql-server/topic/5859/traitement-des-transactions

Chapitre 109: Tri / classement des lignes

Examples

Les bases

Tout d'abord, configurons le tableau d'exemple.

```
-- Create a table as an example
CREATE TABLE SortOrder
(
    ID INT IDENTITY PRIMARY KEY,
    [Text] VARCHAR(256)
)
GO

-- Insert rows into the table
INSERT INTO SortOrder ([Text])
SELECT ('Lorem ipsum dolor sit amet, consectetur adipiscing elit')
UNION ALL SELECT ('Pellentesque eu dapibus libero')
UNION ALL SELECT ('Vestibulum et consequat est, ut hendrerit ligula')
UNION ALL SELECT ('Suspendisse sodales est congue lorem euismod, vel facilisis libero
pulvinar')
UNION ALL SELECT ('Suspendisse lacus est, aliquam at varius a, fermentum nec mi')
UNION ALL SELECT ('Praesent tincidunt tortor est, nec consequat dolor malesuada quis')
UNION ALL SELECT ('Quisque at tempus arcu')
GO
```

Rappelez-vous que lors de la récupération de données, si vous ne spécifiez pas de clause de classement des lignes (ORDER BY), le serveur SQL ne garantit pas le tri (ordre des colonnes) **à tout moment** . Vraiment, à tout moment. Et il est inutile de discuter de cela, il a été montré littéralement des milliers de fois et sur Internet.

No ORDER BY == pas de tri. Fin de l'histoire.

```
-- It may seem the rows are sorted by identifiers,
-- but there is really no way of knowing if it will always work.
-- And if you leave it like this in production, Murphy gives you a 100% that it wont.
SELECT * FROM SortOrder
GO
```

Il y a deux directions les données peuvent être commandées par:

- ascendant (en remontant), en utilisant ASC
- en descendant (en descendant), en utilisant DESC

```
-- Ascending - upwards
SELECT * FROM SortOrder ORDER BY ID ASC
GO

-- Ascending is default
SELECT * FROM SortOrder ORDER BY ID
```

```
GO

-- Descending - downwards
SELECT * FROM SortOrder ORDER BY ID DESC
GO
```

Lors de la commande par la colonne textuelle ((n) char ou (n) varchar), faites attention à ce que l'ordre respecte le classement. Pour plus d'informations sur le classement, recherchez le sujet.

La commande et le tri des données peuvent consommer des ressources. C'est là que les index correctement créés sont utiles. Pour plus d'informations sur les index, recherchez le sujet.

Il est possible de pseudo-randomiser l'ordre des lignes dans votre jeu de résultats. Il suffit de forcer la commande à paraître non déterministe.

```
SELECT * FROM SortOrder ORDER BY CHECKSUM(NEWID())
GO
```

La commande peut être mémorisée dans une procédure stockée, et c'est ainsi que vous devez le faire s'il s'agit de la dernière étape de manipulation de l'ensemble de lignes avant de le montrer à l'utilisateur final.

```
CREATE PROCEDURE GetSortOrder
AS
    SELECT *
    FROM SortOrder
    ORDER BY ID DESC
GO

EXEC GetSortOrder
GO
```

Il existe également un support limité (et novateur) pour la commande dans les vues SQL Server, mais soyez encouragé à ne PAS l'utiliser.

```
/* This may or may not work, and it depends on the way
   your SQL Server and updates are installed */
CREATE VIEW VwSortOrder1
AS
    SELECT TOP 100 PERCENT *
    FROM SortOrder
    ORDER BY ID DESC
GO

SELECT * FROM VwSortOrder1
GO

-- This will work, but hey... should you really use it?
CREATE VIEW VwSortOrder2
AS
    SELECT TOP 99999999 *
    FROM SortOrder
    ORDER BY ID DESC
GO
```

```
SELECT * FROM VwSortOrder2
GO
```

Pour commander, vous pouvez utiliser des noms de colonne, des alias ou des numéros de colonne dans votre BY BY ORDER.

```
SELECT *
FROM SortOrder
ORDER BY [Text]

-- New resultset column aliased as 'Msg', feel free to use it for ordering
SELECT ID, [Text] + ' (' + CAST(ID AS nvarchar(10)) + ')' AS Msg
FROM SortOrder
ORDER BY Msg

-- Can be handy if you know your tables, but really NOT GOOD for production
SELECT *
FROM SortOrder
ORDER BY 2
```

Je déconseille d'utiliser les numéros dans votre code, sauf si vous souhaitez l'oublier au moment où vous l'exécutez.

Commande par cas

Si vous souhaitez trier vos données numériquement ou alphabétiquement, vous pouvez simplement utiliser l' `order by [column]` . Si vous souhaitez trier à l'aide d'une hiérarchie personnalisée, utilisez une instruction de casse.

```
Group
-----
Total
Young
MiddleAge
Old
Male
Female
```

Utiliser un `order by` base `order by` :

```
Select * from MyTable
Order by Group
```

renvoie un tri alphabétique, ce qui n'est pas toujours souhaitable:

```
Group
-----
Female
Male
MiddleAge
Old
Total
Young
```

L'ajout d'une instruction 'case', en attribuant des valeurs numériques ascendantes dans l'ordre dans lequel vous voulez classer vos données:

```
Select * from MyTable
Order by case Group
    when 'Total' then 10
    when 'Male' then 20
    when 'Female' then 30
    when 'Young' then 40
    when 'MiddleAge' then 50
    when 'Old' then 60
    end
```

renvoie les données dans l'ordre spécifié:

```
Group
-----
Total
Male
Female
Young
MiddleAge
Old
```

Lire Tri / classement des lignes en ligne: https://riptutorial.com/fr/sql-server/topic/5332/tri---classement-des-lignes

Chapitre 110: Types de données

Introduction

Cette section présente les types de données que SQL Server peut utiliser, notamment leur plage de données, leur longueur et leurs limites (le cas échéant).

Examples

Numérique exacte

Il existe deux classes de base de types de données numériques exactes: **Integer** et **Fixed Precision and Scale** .

Types de données entiers

- bit
- tinyint
- smallint
- int
- bigint

Les entiers sont des valeurs numériques qui ne contiennent jamais de fraction, et utilisent toujours une quantité de stockage fixe. La plage et les tailles de stockage des types de données entiers sont présentées dans ce tableau:

Type de données	Gamme	Espace de rangement
bit	0 ou 1	1 bit **
tinyint	0 à 255	1 octet
smallint	-2 ^ 15 (-32 768) à 2 ^ 15-1 (32 767)	2 octets
int	-2 ^ 31 (-2 147 443 648) à 2 ^ 31-1 (2 147 483 647)	4 octets
bigint	-2 ^ 63 (-9 223 372 036 854 775 808) à 2 ^ 63-1 (9 223 372 036 854 775 807)	8 octets

Précision fixe et types de données d'échelle

- numérique
- décimal
- petit argent
- argent

Ces types de données sont utiles pour représenter les nombres avec précision. Tant que les valeurs peuvent tenir dans la plage des valeurs pouvant être stockées dans le type de données, la valeur n'aura pas de problèmes d'arrondi. Ceci est utile pour tout calcul financier, où les erreurs d'arrondi entraîneront une folie clinique pour les comptables.

Notez que **decimal** et **numeric** sont des synonymes pour le même type de données.

Type de données	Gamme	Espace de rangement
Decimal [(p [, s])] ou Numeric [(p [, s])]	-10 ^ 38 + 1 à 10 ^ 38 - 1	Voir tableau de **précision**

Lors de la définition d'un type de données *décimal* ou *numérique* , vous devrez peut-être spécifier la précision [p] et l'échelle [s].

La précision est le nombre de chiffres pouvant être stockés. Par exemple, si vous deviez stocker des valeurs entre 1 et 999, vous auriez besoin d'une précision de 3 (pour contenir les trois chiffres de 100). Si vous ne spécifiez pas de précision, la précision par défaut est 18.

L'échelle est le nombre de chiffres après le point décimal. Si vous avez besoin de stocker un nombre compris entre 0,00 et 999,99, vous devez spécifier une précision de 5 (cinq chiffres) et une échelle de 2 (deux chiffres après le point décimal). Vous devez spécifier une précision pour spécifier une échelle. L'échelle par défaut est zéro.

La précision d'un type de données *décimal* ou *numérique* définit le nombre d'octets requis pour stocker la valeur, comme indiqué ci-dessous:

Tableau de précision

Précision	Octets de stockage
1 - 9	5
10-19	9
20-28	13
29-38	17

Types de données fixes monétaires

Ces types de données sont destinés spécifiquement à la comptabilité et à d'autres données monétaires. Ces types ont une échelle fixe de 4 - vous verrez toujours quatre chiffres après la décimale. Pour la plupart des systèmes fonctionnant avec la plupart des devises, utiliser une valeur *numérique* avec une échelle de 2 suffira. Notez qu'aucune information sur le type de devise représenté n'est enregistrée avec la valeur.

Type de données	Gamme	Espace de rangement
argent	-922 337 203 685 477,5808 à 922 337 203 685 477,5807	8 octets
petit argent	-214.748.3648 à 214.748.3647	4 octets

Numérique approximative

- float [(**n**)]
- réal

Ces types de données sont utilisés pour stocker des nombres à virgule flottante. Étant donné que ces types ne sont destinés qu'à contenir des valeurs numériques approximatives, celles-ci ne doivent pas être utilisées dans les cas où une erreur d'arrondi est inacceptable. Cependant, si vous avez besoin de traiter des nombres très importants ou des nombres avec un nombre indéterminé de chiffres après la décimale, cela peut être votre meilleure option.

Type de données	Gamme	Taille
flotte	-1,79E + 308 à -2,23E-308, 0 et 2,23E-308 à 1,79E + 308	dépend de **n** dans le tableau ci-dessous
réal	-3,40E + 38 à -1,18E - 38, 0 et 1,18E - 38 à 3,40E + 38	4 octets

n tableau des valeurs pour les nombres *flottants* . Si aucune valeur n'est spécifiée dans la déclaration du flottant, la valeur par défaut de 53 sera utilisée. Notez que *float (24)* est l'équivalent d'une valeur *réelle* .

n valeur	Précision	Taille
1-24	7 chiffres	4 octets
25-53	15 chiffres	8 octets

Date et l'heure

Ces types sont dans toutes les versions de SQL Server

- datetime
- petit temps

Ces types sont dans toutes les versions de SQL Server après SQL Server 2012

- rendez-vous amoureux

- datetimeoffset
- datetime2
- temps

Chaînes de caractères

- carboniser
- varchar
- texte

Chaînes de caractères Unicode

- nchar
- nvarchar
- ntext

Cordes binaires

- binaire
- varbinary
- image

Autres types de données

- le curseur
- horodatage
- hierarchyid
- identifiant unique
- sql_variant
- xml
- table
- Types spatiaux

Lire Types de données en ligne: https://riptutorial.com/fr/sql-server/topic/5260/types-de-donnees

Chapitre 111: Types de table définis par l'utilisateur

Introduction

Les types de table définis par l'utilisateur (UDT en abrégé) sont des types de données permettant à l'utilisateur de définir une structure de table. Les types de table définis par l'utilisateur prennent en charge les clés primaires, les contraintes uniques et les valeurs par défaut.

Remarques

Les UDT ont les restrictions suivantes -

- ne peut pas être utilisé comme une colonne dans une table ou un champ dans un type structuré défini par l'utilisateur
- un index non clusterisé ne peut pas être créé dans un UDT sauf si l'index est le résultat de la création d'une contrainte PRIMARY KEY ou UNIQUE sur l'UDT
- La définition UDT NE PEUT PAS être modifiée après sa création

Examples

créer un UDT avec une seule colonne int qui est également une clé primaire

```
CREATE TYPE dbo.Ids as TABLE
(
    Id int PRIMARY KEY
)
```

Créer un UDT avec plusieurs colonnes

```
CREATE TYPE MyComplexType as TABLE
(
    Id int,
    Name varchar(10)
)
```

Créer un UDT avec une contrainte unique:

```
CREATE TYPE MyUniqueNamesType as TABLE
(
    FirstName varchar(10),
    LastName varchar(10),
    UNIQUE (FirstName,LastName)
)
```

Remarque: les contraintes dans les types de table définis par l'utilisateur ne peuvent pas être nommées.

Créer un UDT avec une clé primaire et une colonne avec une valeur par défaut:

```
CREATE TYPE MyUniqueNamesType as TABLE
(
    FirstName varchar(10),
    LastName varchar(10),
    CreateDate datetime default GETDATE()
    PRIMARY KEY (FirstName,LastName)
)
```

Lire Types de table définis par l'utilisateur en ligne: https://riptutorial.com/fr/sql-server/topic/5280/types-de-table-definis-par-l-utilisateur

Chapitre 112: Utilisation de la table TEMP

Remarques

Les tables temporaires sont vraiment très utiles.

La table peut être créée à l'exécution et peut effectuer toutes les opérations effectuées dans une table normale.

Ces tables sont créées dans une base de données tempdb.

Utilisé quand?

1. Nous devons faire des opérations de jointure complexes.
2. Nous effectuons un grand nombre de manipulations de lignes dans les procédures stockées.
3. Peut remplacer l'utilisation du curseur.

Augmente ainsi la performance.

Examples

Table Temp locale

- Sera disponible jusqu'à ce que la connexion en cours persiste pour l'utilisateur.

  Supprimé automatiquement lorsque l'utilisateur se déconnecte.

  Le nom devrait commencer par # (#temp)

  ```
  CREATE TABLE #LocalTempTable(
                  StudentID      int,
                  StudentName    varchar(50),
                  StudentAddress varchar(150))
  ```

```
insert into #LocalTempTable values ( 1, 'Ram','India');

select * from #LocalTempTable
```

Après avoir exécuté toutes ces instructions si nous fermons la fenêtre de requête et l'ouvrons à nouveau et que nous essayons d'insérer et de sélectionner, la fenêtre affiche un message d'erreur

```
“Invalid object name #LocalTempTable”
```

Table de température globale

- Commencera par ## (## temp).

  Sera supprimé uniquement si l'utilisateur déconnecte toutes les connexions.

  Il se comporte comme une table permanente.

```
CREATE TABLE ##NewGlobalTempTable(
               StudentID      int,
               StudentName    varchar(50),
               StudentAddress varchar(150))

Insert Into ##NewGlobalTempTable values ( 1,'Ram','India');
Select * from ##NewGlobalTempTable
```

Remarque: ils sont visibles par tous les utilisateurs de la base de données, quel que soit leur niveau d'autorisation.

Suppression de tables temporaires

Les tables temporaires doivent avoir des ID uniques (dans la session, pour les tables temporaires locales ou dans le serveur, pour les tables temporaires globales). Essayer de créer une table en utilisant un nom existant renverra l'erreur suivante:

```
There is already an object named '#tempTable' in the database.
```

Si votre requête produit des tables temporaires et que vous souhaitez l'exécuter plusieurs fois, vous devrez supprimer les tables avant d'essayer de les générer à nouveau. La syntaxe de base pour ceci est:

```
drop table #tempTable
```

Essayer d'exécuter cette syntaxe avant que la table n'existe (par exemple lors de la première exécution de votre syntaxe) provoquera une autre erreur:

```
Cannot drop the table '#tempTable', because it does not exist or you do not have permission.
```

Pour éviter cela, vous pouvez vérifier si la table existe déjà avant de la supprimer, comme ceci:

```
IF OBJECT_ID ('tempdb..#tempTable', 'U') is not null DROP TABLE #tempTable
```

Lire Utilisation de la table TEMP en ligne: https://riptutorial.com/fr/sql-server/topic/5328/utilisation-de-la-table-temp

Chapitre 113: WHILE boucle

Remarques

L'utilisation d'une boucle `WHILE` ou d'un autre processus itératif n'est généralement pas le moyen le plus efficace de traiter des données dans SQL Server.

Vous devriez préférer utiliser une requête basée sur un ensemble sur les données pour obtenir les mêmes résultats, si possible

Examples

Utiliser la boucle While

La boucle `WHILE` peut être utilisée comme alternative aux `CURSORS` . L'exemple suivant imprimera les nombres de 0 à 99.

```
DECLARE @i int = 0;
WHILE(@i < 100)
BEGIN
   PRINT @i;
   SET @i = @i+1
END
```

Pendant la boucle avec l'utilisation de la fonction d'agrégat min

```
DECLARE @ID AS INT;

SET @ID = (SELECT MIN(ID) from TABLE);

WHILE @ID IS NOT NULL
BEGIN
    PRINT @ID;
    SET @ID = (SELECT MIN(ID) FROM TABLE WHERE ID > @ID);
END
```

Lire WHILE boucle en ligne: https://riptutorial.com/fr/sql-server/topic/4249/while-boucle

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec Microsoft SQL Server	Abhilash R Vankayala, Abhishek Jain, Ahmad Aghazadeh, Ahmar, Akshay Anand, alalp, Almir Vuk, Arthur D, ATC, Athafoud, BeaglesEnd, Bhanu, Biju jose, Blachshma, bluefeet, ChrisM, Christos, Community, cteski, D M, Darshak, Gidil, Gordon Bell, Greg Bray, Iztoksson, Jared Hooper, JerryOL, Job AJ, Joe Taras, John Odom, John Slegers, JonasCz, K48, kafka , Lamak, Laughing Vergil, Mahesh Dahal, Malt, Martin Smith, Matt, Matt, Max, Mihai-Daniel Virna, Mudassir Hasan, n00b, Nick, Nikolay Kostov, onupdatecascade, OzrenTkalcecKrznaric, Peter Tirrell, Phrancis, Prateek, Sam, Shaneis, Thuta Aung, Tony L., Tot Zam, Uberzen1, Umachandar - Microsoft, user_0, user2314737, VoidDemon, Zsuzsa
2	Analyser une requête	DForck42
3	Autorisations de base de données	Ben Thul
4	Autorisations et sécurité	5arx
5	Avec option liens	TheGameiswar
6	Base de données système - TempDb	Anuj Tripathi, RamenChef
7	bcp (programme de copie en bloc) Utilitaire	MarmiK
8	BULK Import	Jovan MSFT
9	Clause OVER	Athafoud, bluefeet, Brandon, DVT, gofr1, Lamak, Paul Bambury , RamenChef, Rowland Shaw, Sam
10	Clés étrangères	Jovan MSFT
11	Clés primaires	Kritner
12	Colonnes calculées	cnayak, Kannan Kandasamy

13	COLUMNSTORE CLUSTERED	Jovan MSFT
14	COMMANDÉ PAR	APH, beercohol, cteski, Gidil, RamenChef
15	Conseils de requête	cteski, DARKOCEAN, Jovan MSFT, user_0
16	Conversion de types de données	Ben O, Edathadan Chief aka Arun
17	Courtier de service	Ken S., Matej, RamenChef
18	CRÉER UNE VUE	Almir Vuk, cteski, Edathadan Chief aka Arun, Hadi, Josh B, Robert Columbia, Tot Zam
19	croix appliquer	Hamza Rabah, Jovan MSFT, Tom V
20	Cryptage	Rubenisme
21	DBCC	Jovan MSFT
22	DBMAIL	Phrancis
23	Déclaration CASE	Laughing Vergil, RamenChef, Vikas Vaidya
24	Déclencheur	Oluwafemi, The_Outsider, Zohar Peled
25	Délimiter des caractères spéciaux et des mots réservés	bassrek
26	Déplacer et copier des données autour des tables	Nick.McDermaid
27	Dernière identité insérée	Jeffrey Van Laethem, sqluser, Tot Zam
28	Des curseurs	Kane, Phrancis
29	Des vues	Benjamin Hodgson, Daniel Lemke, Max
30	Données spatiales	cteski, Neil Kennedy, RamenChef, Vladimir Oselsky
31	Drop mot-clé	Ignas, Jakub Ojmucianski, Justin Rohr, Max, scsimon
32	Dynamic SQL Pivot	Jesse
33	Ensemble de résultats de limite	alalp, chrisb, cteski, ErikE

34	ESSAYEZ / CATCH	Jovan MSFT, ravindra, Uberzen1
35	Exporter des données dans un fichier txt à l'aide de SQLCMD	sheraz mirza
36	Expressions de table communes	Arif, bbrown, cteski, DForck42, Jeffrey Van Laethem, Jovan MSFT, kafka, Keith Hall, Monty Wild, SQLMason
37	Filestream	Raghu Ariga
38	Fonction Split String dans Sql Server	Jibin Balachandran, Jovan MSFT, MasterBob, Pரதீ ப், RamenChef
39	Fonctions d'agrégat	Akshay Anand, cnayak, cteski, Jeffrey L Whitledge, Joe Taras, Vexator
40	Fonctions d'agrégation de chaînes dans SQL Server	Kannan Kandasamy
41	Fonctions de chaîne	A_Arnold, anon, cteski, FoxyBOA, Hadi, hatchet, Igor Micev, Jibin Balachandran, Jovan MSFT, mtb, Phrancis, Raidri, Ricardo C, Ross Presser, takrl, Zohar Peled
42	Fonctions de classement	cteski, kolunar, New
43	Fonctions de fenêtre	andyabel, feetwet, MarmiK
44	Fonctions logiques	dd4711
45	FUSIONNER	Abhilash R Vankayala, Abubakar Riaz, Alex, David Kaminski, dd4711, Hari K M, Moshiour, Rogerio Soares, Serg
46	Générer une plage de dates	James, Siyual
47	Gestion de la base de données SQL Azure	Jovan MSFT
48	Gouverneur des ressources	Ako, RamenChef
49	Groupe de fichiers	Behzad
50	Indexation de texte	Edathadan Chief aka Arun

	intégral	
51	Indice	Ahmad Aghazadeh, Akshay Anand, cteski, Henrik Staun Poulsen, Martin Smith, Tom V
52	Insérer	Randall
53	INSÉRER DANS	Abubakar Riaz, barcanoj, DVJex, Hari K M, intox, martinshort, Matas Vaitkevicius, Max, Michael Stum, n00b, Piotr Nawrot, Robert Columbia, Tot Zam, woony
54	Installation de SQL Server sur Windows	Luis Bosquez
55	Instantanés de base de données	Akash, Daryl, Jovan MSFT, Wolfgang
56	Instruction SELECT	cteski, Jovan MSFT
57	Intégration du Common Language Runtime	Jovan MSFT
58	Interrogation des résultats par page	Pat
59	Joindre	4444, Akshay Anand, Andy, APH, Bino Mathew Varghese, cteski, Dean Ward, DhruvJoshi, Dileep, Gajendra, HK1, Iztoksson, Jeffrey Van Laethem, Joao Araujo, JonH, L J, Lamak , Laughing Vergil, LowlyDBA, mtb, Nikolay Kostov, OzrenTkalcecKrznaric, Phrancis, Ram Grandhi, SqlZim
60	JSON dans Sql Server	Jovan MSFT, Mono
61	La fonction STUFF	Arthur D, bluefeet, Chetan Sanghani, dacohenii, Kiran Ukande, Luis Bosquez, MrE, user1690166
62	Les variables	APH, Keith Hall, Phrancis
63	Magasin de requêtes	bakedpatato, Jovan MSFT
64	Masquage dynamique des données	Jovan MSFT
65	Migration	Matas Vaitkevicius
66	Modifier le texte JSON	Jovan MSFT

67	Modules compilés nativement (Hekaton)	bakedpatato, Jovan MSFT
68	Niveaux d'isolation des transactions	Phrancis
69	Niveaux d'isolement et verrouillage	RamenChef, sqlandmore.com
70	Noms d'alias dans Sql Server	Pரதீ ப
71	NULL	Amir Pourmand, Hadi, Kannan Kandasamy, Kritner, Laughing Vergil, podiluska, Sean Branchaw, Zohar Peled
72	OLTP en mémoire (Hekaton)	Akshay Anand, Behzad, Brandon, Jovan MSFT, Martijn Pieters
73	OPENJSON	James, Jovan MSFT
74	Opérations de base DDL dans MS SQL Server	Matt
75	Options avancées	Ahmad Aghazadeh
76	Pagination	cteski, Jovan MSFT, Sender
77	PAR GROUPE	Andy, Edathadan Chief aka Arun, Jenism, juergen d, Julien Vavasseur, Kiran Ukande, Matas Vaitkevicius, ShlomiR
78	Paramètres de la table	Zohar Peled
79	Parsename	Mani
80	Partitionnement	Dan Guzman, James Anderson, John Odom, Susang
81	PHANTOM lu	Max
82	PIVOT / UNPIVOT	Athafoud, bluefeet, DhruvJoshi, kolunar
83	POUR JSON	James, Jovan MSFT
84	POUR XML PATH	bluefeet, gotqn, Keith Hall, Wolfgang
85	Privilèges ou autorisations	Oluwafemi
86	Procédures stockées	Bino Mathew Varghese, cnayak, cteski, Erik Oppedijk, Eugene

		Niemand, Hari K M, Jayasurya Satheesh, Matas Vaitkevicius, Nathan Skerl, Pirate X, scsimon
87	Récupérer des informations sur la base de données	Andrea, Anuj Tripathi, Baodad, Brent Ozar, dario, feetwet, James Anderson, JamieA, Jasmin Solanki, Jeffrey Van Laethem, jyao, Kritner, Laughing Vergil, LowlyDBA, Mahendra, Moshiour, Phrancis, Rhumborl, scsimon, Shiva, spaghettidba, Tot Zam, TZHX, Umachandar - Microsoft
88	Récupérer des informations sur votre instance	Bino Mathew Varghese, feetwet, James Anderson, Kritner, LowlyDBA, S.Karras, scsimon
89	Rendez-vous	A_Arnold, Adam Porad, Akshay Anand, Bellash, cteski, Edathadan Chief aka Arun, JamieA, Jared Hooper, Kritner, Lamak, Mert Gülsoy, Nick, Phrancis, SHD, Siyual, Soukai, UnhandledExcepSean, Zohar Peled
90	Requêtes avec des données JSON	bakedpatato, James, Jovan MSFT
91	Sauvegarde et restauration de la base de données	Jones Joseph, Jovan MSFT
92	Schémas	Merenix
93	SCOPE_IDENTITY ()	Dheeraj Kumar
94	SE FONDRE	Bharat Prasad Satyal, Edathadan Chief aka Arun, Karthikeyan, Matej, scsimon, Tab Alleman
95	Sécurité au niveau des lignes	Carsten Hynne, Jovan MSFT
96	Séquences	Josh Morel
97	SINON	cteski, M.Ali, RamenChef
98	Sous-requêtes	cnayak
99	SQL dynamique	Jovan MSFT
100	SQL Server Evolution à travers différentes versions (2000 - 2016)	Dan Guzman, M.Ali
101	SQL Server	dd4711

	Management Studio (SSMS)	
102	SQLCMD	Eugene Niemand, Techie
103	Stocker JSON dans les tables SQL	Ed Harper, Jovan MSFT, RamenChef
104	SYNDICAT	cnayak
105	Tables Temporelles	Ahmad Aghazadeh, Akshay Anand, Ben O, Mspaja
106	Tâche ou tâche programmée	Edathadan Chief aka Arun, Hadi
107	Touches de raccourci Microsoft SQL Server Management Studio	Bino Mathew Varghese, cteski, Sibeesh Venu
108	Traitement des transactions	Metanormal
109	Tri / classement des lignes	APH, OzrenTkalcecKrznaric
110	Types de données	Laughing Vergil, Matas Vaitkevicius
111	Types de table définis par l'utilisateur	Jivan, Zohar Peled
112	Utilisation de la table TEMP	APH, New
113	WHILE boucle	lord5et, Matas Vaitkevicius, podiluska, RamenChef, Wojciech Kazior