Kostenloses eBook

LERNEN

Microsoft SQL Server

Free unaffiliated eBook created from
Stack Overflow contributors.

#sql-server

Inhaltsverzeichnis

Über

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

Kapitel 1: Erste Schritte mit Microsoft SQL Server

Bemerkungen

Dies ist eine Reihe von Beispielen, die die grundlegende Verwendung von SQL Server hervorheben.

Versionen

Ausführung	Veröffentlichungsdatum
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: Grundlagen der Datenbearbeitungssprache

D ata **M** Anipulation **L-** Sprache (kurz DML) umfasst Operationen wie `INSERT` , `UPDATE` und `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
```

In diesem Skript **erstellen** wir **eine Tabelle** , um einige grundlegende Abfragen zu veranschaulichen.

Die folgenden Beispiele zeigen, wie **Tabellen abgefragt werden:**

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

wählt die ersten 10 Datensätze der Tabelle `Customer` , sortiert nach der Spalte `CompanyName` aus der Datenbank `Northwind` (eine der Beispieldatenbanken von Microsoft, die hier heruntergeladen **werden kann**):

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

Beachten Sie, dass `Use Northwind;` ändert die Standarddatenbank für alle nachfolgenden Abfragen. Sie können weiterhin auf die Datenbank verweisen, indem Sie die vollständig qualifizierte Syntax in der Form [Datenbank]. [Schema]. [Tabelle] verwenden.

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

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

Dies ist nützlich, wenn Sie Daten aus verschiedenen Datenbanken abfragen. Beachten Sie, dass `dbo` , angegeben "zwischen", als Schema bezeichnet wird und bei der Verwendung der vollständig qualifizierten Syntax angegeben werden muss. Sie können es sich als einen Ordner in Ihrer Datenbank vorstellen. `dbo` ist das Standardschema. Das Standardschema kann weggelassen werden. Alle anderen benutzerdefinierten Schemata müssen angegeben werden.

Wenn die Datenbanktabelle Spalten enthält, die wie reservierte Wörter benannt sind, z. B. `Date` , müssen Sie den Spaltennamen wie folgt in Klammern einschließen:

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

Dasselbe gilt, wenn der Spaltenname Leerzeichen enthält (was nicht empfohlen wird). Eine alternative Syntax ist die Verwendung von doppelten Anführungszeichen anstelle von eckigen Klammern, zB:

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

ist gleichwertig, wird aber nicht so häufig verwendet. Beachten Sie den Unterschied zwischen doppelten Anführungszeichen und einfachen Anführungszeichen: Einfache Anführungszeichen werden für Zeichenfolgen verwendet

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

ist eine gültige Syntax. Beachten Sie, dass T-SQL ein `N` Präfix für NChar- und NVarchar-Datentypen hat, z

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

Gibt alle Unternehmen zurück, deren Firmenname mit `AL` beginnt (`%` ist ein Platzhalter, verwenden Sie den Stern in einer DOS-Befehlszeile, z. B. `DIR AL*`). Für `LIKE` gibt es ein paar Wildcards zur Verfügung, schauen Sie **hier** , um weitere Details zu erfahren.

Schließt sich an

Verknüpfungen sind hilfreich, wenn Sie Felder abfragen möchten, die nicht in einer einzigen Tabelle, sondern in mehreren Tabellen vorhanden sind. Beispiel: Sie möchten alle Spalten aus der `Region` in der `Northwind` Datenbank abfragen. Sie stellen jedoch fest, dass Sie auch die `RegionDescription` benötigen, die in einer anderen Tabelle, `Region` , gespeichert ist. Es gibt jedoch einen gemeinsamen Schlüssel, `RgionID` dem Sie diese Informationen in einer einzigen Abfrage wie folgt kombinieren können (`Top 5` nur die ersten 5 Zeilen zurück und lässt sie weg, um alle Zeilen abzurufen):

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

zeigt alle Spalten von `Territories` sowie die Spalte `RegionDescription` von `Region` . Das Ergebnis wird nach `TerritoryDescription` sortiert.

Tabellen-Aliase

Wenn für Ihre Abfrage ein Verweis auf zwei oder mehr Tabellen erforderlich ist, kann es hilfreich sein, einen Tabellenalias zu verwenden. Tabellenaliasnamen sind Kurzreferenzen auf Tabellen, die anstelle eines vollständigen Tabellennamens verwendet werden können und die Eingabe und Bearbeitung reduzieren. Die Syntax für die Verwendung eines Alias lautet:

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

Wo `as` ein optionales Schlüsselwort ist. Beispielsweise kann die vorherige Abfrage wie folgt umgeschrieben werden:

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

Aliase müssen für alle Tabellen in einer Abfrage eindeutig sein, auch wenn Sie dieselbe Tabelle zweimal verwenden. Wenn Ihre Employee-Tabelle beispielsweise ein SupervisorId-Feld enthält, können Sie diese Abfrage verwenden, um einen Mitarbeiter und den Namen seines Vorgesetzten zurückzugeben:

```
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
```

Gewerkschaften

Wie wir bereits gesehen haben, fügt ein Join Spalten aus verschiedenen Tabellenquellen hinzu. Was aber, wenn Sie Zeilen aus verschiedenen Quellen kombinieren möchten? In diesem Fall können Sie eine UNION verwenden. Angenommen, Sie planen eine Party und möchten nicht nur Mitarbeiter, sondern auch die Kunden einladen. Dann könnten Sie diese Abfrage ausführen, um dies zu tun:

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

Es werden Namen, Adressen und Städte der Mitarbeiter und Kunden in einer einzigen Tabelle angezeigt. Beachten Sie, dass doppelte Zeilen (falls vorhanden) automatisch entfernt werden (wenn Sie dies nicht möchten, verwenden Sie stattdessen `UNION ALL`). Die Spaltennummer, die Spaltennamen, die Reihenfolge und der Datentyp müssen in allen Select-Anweisungen der Union `LastName` Aus diesem Grund werden beim ersten SELECT `FirstName` und `LastName` aus Employee in `ContactName` .

Tabellenvariablen

Wenn Sie mit temporären Daten (insbesondere in einer gespeicherten Prozedur) arbeiten müssen, kann es nützlich sein, Tabellenvariablen zu verwenden: Der Unterschied zwischen einer "echten" Tabelle und einer Tabellenvariablen besteht darin, dass sie nur für temporäre Verarbeitung im Speicher vorhanden sind.

Beispiel:

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

erstellt eine Tabelle im Speicher. In diesem Fall ist das `@` -Präfix obligatorisch, da es sich um eine Variable handelt. Sie können alle oben genannten DML-Operationen ausführen, um Zeilen einzufügen, zu löschen und auszuwählen, z

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

Normalerweise füllt man es jedoch anhand einer echten Tabelle aus

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

`dbo.Region` würde die gefilterten Werte aus der realen Tabelle `dbo.Region` und in die Speichertabelle

`@Region` wo sie zur weiteren Verarbeitung verwendet werden kann. Zum Beispiel könnten Sie es in einem Join wie verwenden

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

die in diesem Fall alle `Northern` und `Southern` Gebiete zurückgeben würde. Weitere Informationen finden Sie hier . Temporäre Tabellen werden hier besprochen, wenn Sie mehr über dieses Thema erfahren möchten.

HINWEIS: Microsoft empfiehlt die Verwendung von Tabellenvariablen nur, wenn die Anzahl der Datenzeilen in der Tabellenvariablen weniger als 100 beträgt. Wenn Sie mit größeren Datenmengen arbeiten, verwenden Sie stattdessen eine **temporäre Tabelle** oder eine **temporäre Tabelle** .

DRUCKEN

Zeigt eine Nachricht an die Ausgabekonsole an. In SQL Server Management Studio wird dies auf der Registerkarte Nachrichten und nicht auf der Registerkarte Ergebnisse angezeigt.

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

SELECT alle Zeilen und Spalten aus einer Tabelle

Syntax:

```
SELECT *
FROM table_name
```

Mit dem Sternchen-Operator `*` Sie alle Spalten in der Tabelle auswählen. Alle Zeilen werden ebenfalls ausgewählt, da diese `SELECT` Anweisung keine `WHERE` Klausel enthält, um Filterkriterien anzugeben.

Dies würde auch auf dieselbe Weise funktionieren, wenn Sie der Tabelle einen Aliasnamen hinzufügen, z. B. `e` in diesem Fall:

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

Wenn Sie alle aus einer bestimmten Tabelle auswählen möchten, können Sie den Alias + ". *" Verwenden:

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

Auf Datenbankobjekte kann auch mit vollständig qualifizierten Namen zugegriffen werden:

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

Dies wird nicht unbedingt empfohlen, da das Ändern der Server- und / oder Datenbanknamen dazu führen würde, dass Abfragen mit vollständig qualifizierten Namen aufgrund ungültiger Objektnamen nicht mehr ausgeführt werden.

Beachten Sie, dass die Felder vor `table_name` in vielen Fällen weggelassen werden können, wenn die Abfragen auf einem einzelnen Server, einer Datenbank bzw. einem Schema ausgeführt werden. Es ist jedoch üblich, dass eine Datenbank über mehrere Schemas verfügt, und in diesen Fällen sollte der Schemaname nach Möglichkeit nicht ausgelassen werden.

Warnung: Die Verwendung von `SELECT *` in Produktionscode oder gespeicherten Prozeduren kann später zu Problemen führen (wenn neue Spalten zur Tabelle hinzugefügt werden oder wenn Spalten in der Tabelle neu angeordnet werden), insbesondere wenn Ihr Code einfache Annahmen zur Reihenfolge der Spalten macht oder Anzahl der zurückgegebenen Spalten. Daher ist es sicherer, Spaltennamen in SELECT-Anweisungen für Produktionscode immer explizit anzugeben.

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

Wählen Sie Zeilen aus, die einer Bedingung entsprechen

Im Allgemeinen lautet die Syntax:

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

Zum Beispiel:

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

Bedingungen können komplex sein:

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

UPDATE-spezifische Zeile

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

Der obige Code aktualisiert den Wert des Felds "HelloWorld" mit "HELLO WORLD !!!" für den Datensatz mit "Id = 5" in der HelloWorlds-Tabelle.

Hinweis: In einer Aktualisierungsanweisung wird die Verwendung einer "Wo" -Klausel empfohlen, um zu vermeiden, dass die gesamte Tabelle aktualisiert wird, sofern nicht Ihre Anforderung anders ist.

Alle Zeilen aktualisieren

Eine einfache Form der Aktualisierung erhöht alle Werte in einem bestimmten Feld der Tabelle. Dazu müssen wir das Feld und den Inkrementwert definieren

Im folgenden Beispiel wird das `Score` um 1 erhöht (in allen Zeilen):

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

Dies kann gefährlich sein, da Sie Ihre Daten beschädigen können, wenn Sie versehentlich ein UPDATE für eine **bestimmte Zeile** mit UPDATE für **alle Zeilen** in der Tabelle vornehmen.

Kommentare im Code

Transact-SQL unterstützt zwei Arten des Kommentarschreibens. Kommentare werden von der Datenbank-Engine ignoriert und sind für das Lesen gedacht.

Kommentaren sind vorangestellt `--` und werden ignoriert, bis eine neue Zeile gefunden wird:

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

Slash-Sternkommentare beginnen mit `/*` und enden mit `*/` . Der gesamte Text zwischen diesen Trennzeichen wird als Kommentarblock betrachtet.

```
/* 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';
```

Slash-Sternkommentare haben den Vorteil, dass der Kommentar verwendbar bleibt, wenn die SQL-Anweisung neue Zeilenzeichen verliert. Dies kann passieren, wenn SQL während der Problembehandlung erfasst wird.

Schrägstrich-Sternkommentare können verschachtelt sein und ein Start `/*` innerhalb eines Schrägstrich-Sternkommentars muss mit einem `*/` , um gültig zu sein. Der folgende Code führt zu einem Fehler

```
/*
```

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

Der Slash-Stern wird zwar als Anfang eines Kommentars betrachtet, obwohl er innerhalb des Zitats steht. Daher muss es mit einem weiteren schließenden Sternschrägstrich beendet werden. Der richtige Weg wäre

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

Rufen Sie grundlegende Serverinformationen ab

```
SELECT @@VERSION
```

Gibt die Version von MS SQL Server zurück, die in der Instanz ausgeführt wird.

```
SELECT @@SERVERNAME
```

Gibt den Namen der MS SQL Server-Instanz zurück.

```
SELECT @@SERVICENAME
```

Gibt den Namen des Windows-Dienstes zurück, unter dem MS SQL Server ausgeführt wird.

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

Gibt den physischen Namen des Computers zurück, auf dem SQL Server ausgeführt wird. Nützlich zum Identifizieren des Knotens in einem Failovercluster.

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

In einem Failovercluster wird jeder Knoten zurückgegeben, auf dem SQL Server ausgeführt werden kann. Es wird nichts zurückgegeben, wenn nicht ein Cluster.

Verwenden von Transaktionen zum sicheren Ändern von Daten

Wenn Sie Daten ändern, können Sie in einem DML-Befehl (Data Manipulation Language) Ihre Änderungen in eine Transaktion einbinden. DML umfasst `UPDATE` , `TRUNCATE` , `INSERT` und `DELETE` . Sie können beispielsweise sicherstellen, dass Sie die richtigen Daten ändern, indem Sie eine Transaktion verwenden.

DML-Änderungen sperren die betroffenen Zeilen. Wenn Sie eine Transaktion beginnen, müssen Sie die Transaktion beenden, oder alle Objekte, die in der DML geändert werden, bleiben von der

Person gesperrt, die mit der Transaktion begonnen hat. Sie können Ihre Transaktion entweder mit `ROLLBACK` oder `COMMIT` beenden. `ROLLBACK` bringt alles in der Transaktion in den ursprünglichen Zustand zurück. `COMMIT` versetzt die Daten in einen Endzustand, in dem Sie Ihre Änderungen nicht ohne eine andere DML-Anweisung rückgängig machen können.

Beispiel:

```
--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
```

Anmerkungen:

- Dies ist ein **vereinfachtes Beispiel,** das keine Fehlerbehandlung enthält. Jede Datenbankoperation kann jedoch fehlschlagen und daher eine Ausnahme auslösen. **Hier ein Beispiel,** wie eine solche erforderliche Fehlerbehandlung aussehen könnte. Sie sollten **niemals** Transaktionen **ohne Fehlerbehandlungsroutine verwenden** . Andernfalls **belassen** Sie die Transaktion möglicherweise in einem unbekannten Status.
- Abhängig von der **Isolationsstufe sperren** Transaktionen die abgefragten oder geänderten Daten. Sie müssen sicherstellen, dass Transaktionen lange Zeit nicht ausgeführt werden, da sie Datensätze in einer Datenbank sperren und zu **Deadlocks** mit anderen parallel laufenden Transaktionen führen können. Halten Sie die in Transaktionen gekapselten Vorgänge so kurz wie möglich und minimieren Sie die Auswirkungen der gesperrten Datenmenge.

LÖSCHEN Alle Zeilen

```
DELETE
FROM Helloworlds
```

Dadurch werden alle Daten aus der Tabelle gelöscht. Die Tabelle enthält keine Zeilen, nachdem Sie diesen Code ausgeführt haben. Im Gegensatz zu `DROP TABLE` bleiben die Tabelle und ihre Struktur erhalten, und Sie können weiterhin neue Zeilen in diese Tabelle einfügen.

Eine andere Methode zum Löschen aller Zeilen in der Tabelle besteht darin, sie wie folgt zu kürzen:

```
TRUNCATE TABLE HelloWords
```

Der Unterschied beim DELETE-Betrieb ist mehrere:

1. Der Abschneidevorgang wird nicht in der Transaktionslogdatei gespeichert
2. Wenn das Feld `IDENTITY` ist, wird dieses zurückgesetzt
3. TRUNCATE kann auf die gesamte Tabelle und nicht auf einen Teil der Tabelle angewendet werden (stattdessen können Sie mit dem Befehl `DELETE` eine `WHERE` Klausel verknüpfen.)

Einschränkungen von TRUNCATE

1. TRUNCATE kann nicht ausgeführt werden, wenn ein `FOREIGN KEY` Verweis vorhanden ist
2. Wenn die Tabelle an einer `INDEXED VIEW`
3. Wenn die Tabelle mit `TRANSACTIONAL REPLICATION` oder `MERGE REPLICATION`
4. Es werden keine in der Tabelle definierten TRIGGER ausgelöst

[sic]

TRUNCATE TABLE

```
TRUNCATE TABLE Helloworlds
```

Dieser Code löscht alle Daten aus der Tabelle Helloworlds. Tabelle abschneiden ist fast ähnlich wie Code `Delete from Table` löschen. Der Unterschied ist, dass Sie keine where-Klauseln mit Truncate verwenden können. Das Abschneiden von Tabellen gilt als besser als Löschen, da weniger Transaktionsprotokollbereiche verwendet werden.

Wenn eine Identitätsspalte vorhanden ist, wird sie auf den ursprünglichen Startwert zurückgesetzt (z. B. wird die automatisch inkrementierte ID von 1 neu gestartet). Dies kann zu Inkonsistenzen führen, wenn die Identitätsspalten als Fremdschlüssel in einer anderen Tabelle verwendet werden.

Erstellen Sie eine neue Tabelle und fügen Sie Datensätze aus der alten Tabelle ein

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

Erstellt eine neue Tabelle mit der Struktur der alten Tabelle und fügt alle Zeilen in die neue Tabelle ein.

Einige Einschränkungen

1. Sie können keine Tabellenvariable oder einen Tabellenwertparameter als neue Tabelle angeben.
2. Sie können SELECT… INTO nicht verwenden, um eine partitionierte Tabelle zu erstellen, auch wenn die Quellentabelle partitioniert ist. SELECT ... INTO verwendet das Partitionsschema der Quelltabelle nicht. Stattdessen wird die neue Tabelle in der Standarddateigruppe erstellt. Um Zeilen in eine partitionierte Tabelle einzufügen, müssen Sie zuerst die partitionierte Tabelle erstellen und dann die Anweisung INSERT INTO ... SELECT FROM verwenden.
3. In der Quelltabelle definierte Indizes, Integritätsbedingungen und Auslöser werden weder in die neue Tabelle übertragen noch können sie in der SELECT ... INTO-Anweisung angegeben werden. Wenn diese Objekte erforderlich sind, können Sie sie erstellen, nachdem Sie die SELECT ... INTO-Anweisung ausgeführt haben.
4. Durch die Angabe einer ORDER BY-Klausel kann nicht garantiert werden, dass die Zeilen in der angegebenen Reihenfolge eingefügt werden. Wenn eine spärliche Spalte in der Auswahlliste enthalten ist, wird die Eigenschaft der spärlichen Spalte nicht in die Spalte in der neuen Tabelle übertragen. Wenn diese Eigenschaft in der neuen Tabelle erforderlich ist, ändern Sie die Spaltendefinition nach der Ausführung der Anweisung SELECT ... INTO, um diese Eigenschaft einzuschließen.
5. Wenn eine berechnete Spalte in der Auswahlliste enthalten ist, ist die entsprechende Spalte in der neuen Tabelle keine berechnete Spalte. Die Werte in der neuen Spalte sind die Werte, die zum Zeitpunkt der Ausführung von SELECT ... INTO berechnet wurden.

[*sic*]

Tabellenzeilenanzahl abrufen

Mit dem folgenden Beispiel können Sie die Gesamtzahl der Zeilen für eine bestimmte Tabelle in einer Datenbank ermitteln, wenn `table_name` durch die Tabelle ersetzt wird, die Sie abfragen möchten:

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

Es ist auch möglich, die Zeilenanzahl für alle Tabellen abzurufen, indem Sie sich mit der Tabellenpartition basierend auf dem HEAP der Tabelle (index_id = 0) oder dem Clustered Clustered-Index (index_id = 1) mit dem folgenden Skript verbinden:

```
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;
```

Dies ist möglich, da jede Tabelle im Wesentlichen eine einzelne Partitionstabelle ist, es sei denn, zusätzliche Partitionen werden hinzugefügt. Dieses Skript hat auch den Vorteil, dass Lese- und Schreibvorgänge in den Tabellenzeilen nicht beeinträchtigt werden.

Erste Schritte mit Microsoft SQL Server online lesen: https://riptutorial.com/de/sql-server/topic/236/erste-schritte-mit-microsoft-sql-server

Kapitel 2: Abfragehinweise

Examples

JOIN Hinweise

Wenn Sie zwei Tabellen verbinden, kann der SQL Server-Abfrageoptimierer (QO) verschiedene Arten von Joins auswählen, die in der Abfrage verwendet werden:

- HASH beitreten
- LOOP beitreten
- MERGE beitreten

QO untersucht Pläne und wählt den optimalen Operator für das Verbinden von Tabellen aus. Wenn Sie jedoch sicher sind, dass Sie den optimalen Join-Operator kennen, können Sie angeben, welche Art von JOIN verwendet werden soll. Innerer LOOP-Join zwingt QO zur Auswahl des Nested-Loop-Joins, während er zwei Tabellen verbindet:

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

Inner Merge Join erzwingt den MERGE Join-Operator:

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

inneres Hash-Join zwingt den HASH-Join-Operator:

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

GROUP BY Hinweise

Wenn Sie die GROUP BY-Klausel verwenden, kann der SQL Server-Abfrageoptimierer (QO) verschiedene Typen von Gruppierungsoperatoren auswählen:

- HASH Aggregat, das eine Hash-Map zum Gruppieren von Einträgen erstellt
- Stream Aggregate, das gut mit vorbestellten Eingaben funktioniert

Sie können explizit verlangen, dass QO den einen oder anderen Aggregatoperator auswählt, wenn Sie wissen, was optimal ist. Mit OPTION (ORDER GROUP) wählt QO immer Stream-

Aggregat und fügt den Sortieroperator vor Stream-Aggregat hinzu, wenn die Eingabe nicht sortiert ist:

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

Bei OPTION (HASH GROUP) wählt QO immer das Hash-Aggregat:

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

SCHNELLE Zeilen Hinweis

Gibt an, dass die Abfrage für das schnelle Abrufen der ersten number_rows optimiert ist. Dies ist eine nicht negative Ganzzahl. Nachdem die ersten number_rows zurückgegeben wurden, setzt die Abfrage die Ausführung fort und erzeugt die vollständige Ergebnismenge.

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

UNION Hinweise

Wenn Sie den UNION-Operator für zwei Abfrageergebnisse verwenden, kann der Abfrageoptimierer (QO) die folgenden Operatoren verwenden, um eine Vereinigung von zwei Ergebnissätzen zu erstellen:

- Zusammenführen (Union)
- Concat (Union)
- Hash Match (Union)

Sie können explizit angeben, welcher Operator mit dem OPTION () - Hinweis verwendet werden soll:

```
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)
```

MAXDOP-Option

Gibt den maximalen Parallelitätsgrad für die Abfrage an, die diese Option angibt.

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

Diese Option überschreibt die MAXDOP-Konfigurationsoption von sp_configure und Resource Governor. Wenn MAXDOP auf Null gesetzt ist, wählt der Server den maximalen Parallelitätsgrad.

INDEX-Hinweise

Indexhinweise werden verwendet, um zu erzwingen, dass eine Abfrage einen bestimmten Index verwendet, anstatt zuzulassen, dass der Abfrageoptimierer von SQL Server den besten Index auswählt. In einigen Fällen können Sie Vorteile haben, indem Sie den Index angeben, den eine Abfrage verwenden muss. Normalerweise wählt der Abfrageoptimierer von SQL Server den besten für die Abfrage geeigneten Index aus. Aufgrund fehlender / veralteter Statistiken oder bestimmter Anforderungen können Sie dies erzwingen.

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

Abfragehinweise online lesen: https://riptutorial.com/de/sql-server/topic/6881/abfragehinweise

Kapitel 3: Abfragen der Ergebnisse nach Seite

Examples

Zeilennummer()

```
SELECT Row_Number() OVER(ORDER BY UserName) As RowID, UserFirstName, UserLastName
FROM Users
```

Daraus ergibt sich eine Ergebnismenge mit einem RowID-Feld, das Sie zum Seitenwechsel verwenden können.

```
SELECT *
FROM
    ( SELECT Row_Number() OVER(ORDER BY UserName) As RowID, UserFirstName, UserLastName
      FROM Users
    ) As RowResults
WHERE RowID Between 5 AND 10
```

Abfragen der Ergebnisse nach Seite online lesen: https://riptutorial.com/de/sql-server/topic/5803/abfragen-der-ergebnisse-nach-seite

Kapitel 4: Abfragen mit JSON-Daten

Examples

Werte aus JSON in Abfrage verwenden

Mit der Funktion JSON_VALUE können Sie Daten aus dem JSON-Text in dem als zweiten Argument angegebenen Pfad abrufen und diesen Wert in einem beliebigen Teil der Auswahlabfrage verwenden:

```
select ProductID, Name, Color, Size, Price, JSON_VALUE(Data, '$.Type') as Type
from Product
where JSON_VALUE(Data, '$.Type') = 'part'
```

JSON-Werte in Berichten verwenden

Sobald JSON-Werte aus JSON-Text extrahiert wurden, können Sie sie in einem beliebigen Teil der Abfrage verwenden. Sie können einen Bericht über JSON-Daten mit Gruppierungsaggregaten usw. erstellen:

```
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
```

Ungültigen JSON-Text aus Abfrageergebnissen herausfiltern

Wenn ein JSON-Text möglicherweise nicht richtig formatiert ist, können Sie diese Einträge mithilfe der ISJSON-Funktion aus der Abfrage entfernen.

```
select ProductID, Name, Color, Size, Price, JSON_VALUE(Data, '$.Type') as Type
from Product
where JSON_VALUE(Data, '$.Type') = 'part'
and ISJSON(Data) > 0
```

Aktualisieren Sie den Wert in der JSON-Spalte

Die JSON_MODIFY-Funktion kann verwendet werden, um den Wert in einem bestimmten Pfad zu aktualisieren. Sie können diese Funktion verwenden, um den ursprünglichen Wert der JSON-Zelle in der UPDATE-Anweisung zu ändern:

```
update Product
set Data = JSON_MODIFY(Data, '$.Price', 24.99)
where ProductID = 17;
```

Die JSON_MODIFY-Funktion aktualisiert oder erstellt einen Preisschlüssel (falls nicht vorhanden).

Wenn der neue Wert NULL ist, wird der Schlüssel entfernt. Die JSON_MODIFY-Funktion behandelt den neuen Wert als Zeichenfolge (Escape-Sonderzeichen, mit doppelten Anführungszeichen umschließen, um die richtige JSON-Zeichenfolge zu erstellen). Wenn Ihr neuer Wert ein JSON-Fragment ist, sollten Sie ihn mit der JSON_QUERY-Funktion einschließen:

```
update Product
set Data = JSON_MODIFY(Data, '$.tags', JSON_QUERY('["promo","new"]'))
where ProductID = 17;
```

Die JSON_QUERY-Funktion ohne zweiten Parameter verhält sich wie eine Umwandlung in JSON. Da das Ergebnis von JSON_QUERY ein gültiges JSON-Fragment (Objekt oder Array) ist, wird JSON_MODIFY diesen Wert nicht ändern, wenn die Eingabe-JSON geändert wird.

Neuen Wert an JSON-Array anhängen

Die JSON_MODIFY-Funktion kann verwendet werden, um einen neuen Wert an ein Array in JSON anzuhängen:

```
update Product
set Data = JSON_MODIFY(Data, 'append $.tags', "sales")
where ProductID = 17;
```

Ein neuer Wert wird am Ende des Arrays angehängt oder ein neues Array mit dem Wert ["sales"] wird erstellt. Die JSON_MODIFY-Funktion behandelt den neuen Wert als Zeichenfolge (Escape-Sonderzeichen, mit doppelten Anführungszeichen umschließen, um die richtige JSON-Zeichenfolge zu erstellen). Wenn Ihr neuer Wert ein JSON-Fragment ist, sollten Sie ihn mit der JSON_QUERY-Funktion einschließen:

```
update Product
set Data = JSON_MODIFY(Data, 'append $.tags', JSON_QUERY('{"type":"new"}'))
where ProductID = 17;
```

Die JSON_QUERY-Funktion ohne zweiten Parameter verhält sich wie eine Umwandlung in JSON. Da das Ergebnis von JSON_QUERY ein gültiges JSON-Fragment (Objekt oder Array) ist, wird JSON_MODIFY diesen Wert nicht ändern, wenn die Eingabe-JSON geändert wird.

JOIN-Tabelle mit innerer JSON-Kollektion

Wenn Sie eine "untergeordnete Tabelle" haben, die als JSON-Auflistung formatiert und in Reihe als JSON-Spalte gespeichert ist, können Sie diese Auflistung entpacken, in eine Tabelle umwandeln und sie mit der übergeordneten Zeile verbinden. Anstelle des Standardoperators JOIN sollten Sie CROSS APPLY verwenden. In diesem Beispiel werden Produktteile als Sammlung von JSON-Objekten in der Datenspalte formatiert und in dieser gespeichert:

```
select ProductID, Name, Size, Price, Quantity, PartName, Code
from Product
    CROSS APPLY OPENJSON(Data, '$.Parts') WITH (PartName varchar(20), Code varchar(5))
```

Das Ergebnis der Abfrage entspricht dem Join zwischen Produkt- und Teiletabellen.

Zeilen finden, die Werte im JSON-Array enthalten

In diesem Beispiel kann das Tags-Array verschiedene Schlüsselwörter wie ["promo", "sales"] enthalten. Daher können wir dieses Array öffnen und Werte filtern:

```
select ProductID, Name, Color, Size, Price, Quantity
from Product
    CROSS APPLY OPENJSON(Data, '$.Tags')
where value = 'sales'
```

OPENJSON öffnet die innere Sammlung von Tags und gibt sie als Tabelle zurück. Dann können wir die Ergebnisse nach einem Wert in der Tabelle filtern.

Abfragen mit JSON-Daten online lesen: https://riptutorial.com/de/sql-server/topic/5028/abfragen-mit-json-daten

Kapitel 5: Abfragespeicher

Examples

Aktivieren Sie den Abfragespeicher in der Datenbank

Der Abfragespeicher kann für die Datenbank mit dem folgenden Befehl aktiviert werden:

```
ALTER DATABASE tpch SET QUERY_STORE = ON
```

SQL Server / Azure SQL-Datenbank erfasst Informationen zu ausgeführten Abfragen und stellt Informationen in Ansichten von sys.query_store bereit:

- 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

Rufen Sie Ausführungsstatistiken für SQL-Abfragen / -pläne ab

Die folgende Abfrage gibt Informationen zu Fragen, ihren Plänen und Durchschnittsstatistiken bezüglich Dauer, CPU-Zeit, physischen und logischen Lesevorgängen zurück.

```
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
```

Daten aus dem Abfragespeicher entfernen

Wenn Sie eine Abfrage oder einen Abfrageplan aus dem Abfragespeicher entfernen möchten, können Sie die folgenden Befehle verwenden:

```
EXEC sp_query_store_remove_query 4;
EXEC sp_query_store_remove_plan 3;
```

Parameter für diese gespeicherten Prozeduren sind Abfrage- / Plan-ID, die aus Systemansichten abgerufen wird.

Sie können auch einfach Ausführungsstatistiken für einen bestimmten Plan entfernen, ohne den Plan aus dem Geschäft zu entfernen:

```
EXEC sp_query_store_reset_exec_stats 3;
```

Parameter für diese Prozedurplan-ID.

Plan für die Abfrage erzwingen

Der SQL-Abfrageoptimierer wählt den möglichen Plan aus, den er für eine Abfrage finden kann. Wenn Sie einen Plan finden, der für einige Abfragen optimal funktioniert, können Sie QO zwingen, diesen Plan immer mit der folgenden gespeicherten Prozedur zu verwenden:

```
EXEC sp_query_store_unforce_plan @query_id, @plan_id
```

Ab diesem Zeitpunkt verwendet QO immer den für die Abfrage bereitgestellten Plan.

Wenn Sie diese Bindung entfernen möchten, können Sie die folgende gespeicherte Prozedur verwenden:

```
EXEC sp_query_store_force_plan @query_id, @plan_id
```

Ab diesem Zeitpunkt wird QO erneut versuchen, den besten Plan zu finden.

Abfragespeicher online lesen: https://riptutorial.com/de/sql-server/topic/7349/abfragespeicher

Kapitel 6: Aggregatfunktionen

Einführung

Aggregatfunktionen in SQL Server führen Berechnungen mit Wertesätzen durch und geben einen einzelnen Wert zurück.

Syntax

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

Examples

SUMME()

Gibt die Summe der numerischen Werte in einer bestimmten Spalte zurück.

Wir haben eine Tabelle wie in der Abbildung gezeigt, in der verschiedene Aggregatfunktionen ausgeführt werden. Der Tabellenname lautet *Marksheet* .

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

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

Die `sum` berücksichtigt Zeilen mit NULL-Wert in dem als Parameter verwendeten Feld nicht

Im obigen Beispiel, wenn wir eine andere Zeile wie folgt haben:

```
106    Italian    NULL
```

Diese Zeile wird bei der Summenberechnung nicht berücksichtigt

AVG ()

Gibt den Durchschnitt der numerischen Werte in einer bestimmten Spalte zurück.

Wir haben eine Tabelle wie in der Abbildung gezeigt, in der verschiedene Aggregatfunktionen ausgeführt werden. Der Tabellenname lautet *Marksheet* .

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

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

Die `average` berücksichtigt Zeilen mit NULL-Wert in dem als Parameter verwendeten Feld nicht

Im obigen Beispiel, wenn wir eine andere Zeile wie folgt haben:

```
106    Italian    NULL
```

Diese Zeile wird bei der Durchschnittsberechnung nicht berücksichtigt

MAX ()

Gibt den größten Wert in einer bestimmten Spalte zurück.

Wir haben eine Tabelle wie in der Abbildung gezeigt, in der verschiedene Aggregatfunktionen ausgeführt werden. Der Tabellenname lautet *Marksheet* .

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

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

MINDEST()

Gibt den kleinsten Wert in einer bestimmten Spalte zurück.

Wir haben eine Tabelle wie in der Abbildung gezeigt, in der verschiedene Aggregatfunktionen ausgeführt werden. Der Tabellenname lautet *Marksheet* .

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

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

ANZAHL()

Gibt die Gesamtzahl der Werte in einer bestimmten Spalte zurück.

Wir haben eine Tabelle wie in der Abbildung gezeigt, in der verschiedene Aggregatfunktionen ausgeführt werden. Der Tabellenname lautet *Marksheet* .

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

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

Die `count` Funktion berücksichtigt keine Zeilen mit NULL-Wert in dem als Parameter verwendeten Feld. Normalerweise ist der count-Parameter * (alle Felder), dh, nur wenn alle Felder der Zeile NULL sind, wird diese Zeile nicht berücksichtigt

Im obigen Beispiel, wenn wir eine andere Zeile wie folgt haben:

```
106    Italian    NULL
```

Diese Zeile wird bei der Zählungsberechnung nicht berücksichtigt

HINWEIS

Die Funktion `COUNT(*)` gibt die Anzahl der Zeilen in einer Tabelle zurück. Dieser Wert kann auch durch die Verwendung eines konstanten Nicht-Null - Ausdrucks erhalten werden, der keine Spalt Referenzen, wie beispielsweise enthält `COUNT(1)` .

Beispiel

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

COUNT (Column_Name) mit GROUP BY Column_Name

Meistens möchten wir beispielsweise die Gesamtzahl des Auftretens eines Spaltenwerts in einer

Tabelle ermitteln:

TABELLENAME: BERICHTE

ReportName	ReportPrice
Prüfung	10,00 $
Prüfung	10,00 $
Prüfung	10,00 $
Test 2	11,00 $
Prüfung	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
```

NAME DES BERICHTS	ANZAHL
Prüfung	4
Test 2	1
Test 3	2
Test 4	1

Aggregatfunktionen online lesen: https://riptutorial.com/de/sql-server/topic/5802/aggregatfunktionen

Kapitel 7: Aliasnamen in SQL Server

Einführung

Es gibt einige Möglichkeiten, Aliasnamen für Spalten in SQL Server bereitzustellen

Examples

Verwendung von AS

Diese ANSI-SQL-Methode funktioniert in allen RDBMS. Weit verbreiteter Ansatz.

```
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
```

Mit =

Dies ist mein bevorzugter Ansatz. Nichts im Zusammenhang mit der Leistung, nur eine persönliche Entscheidung. Es macht den Code sauber aussehen. Sie können die resultierenden Spaltennamen leicht anzeigen, anstatt den Code zu scrollen, wenn Sie einen großen Ausdruck haben.

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FullName = FirstName +' '+ LastName
FROM   AliasNameDemo
```

Geben Sie einen Alias nach abgeleiteter Tabellenname

Dies ist eine seltsame Herangehensweise, von der die meisten Menschen nicht wissen, dass es sie überhaupt gibt.

```
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)
```

- **Demo**

Ohne Verwendung von AS

Diese Syntax ähnelt der Verwendung von `AS` Schlüsselwörtern. Nur müssen wir kein `AS` Schlüsselwort verwenden

```
CREATE TABLE AliasNameDemo (id INT,firstname VARCHAR(20),lastname VARCHAR(20))

INSERT INTO AliasNameDemo
VALUES      (1,'MyFirstName','MyLastName')

SELECT FirstName +' '+ LastName FullName
FROM   AliasNameDemo
```

Aliasnamen in SQL Server online lesen: https://riptutorial.com/de/sql-server/topic/10784/aliasnamen-in-sql-server

Kapitel 8: Allgemeine Tabellenausdrücke

Syntax

- WITH *cte_name* [(*spaltenname_1* , *spaltenname_2* , ...)] AS (*cte_expression*)

Bemerkungen

Es ist notwendig, einen CTE von der vorherigen Anweisung mit einem Semikolon (;) zu trennen.

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

Der Geltungsbereich eines CTE ist eine einzelne Charge und nur hinter seiner Definition. Ein Stapel kann mehrere CTEs enthalten, und ein CTE kann auf einen anderen CTE verweisen, der zuvor im Stapel definiert wurde. Ein CTE kann jedoch keinen anderen CTE angeben, der später im Stapel definiert wird.

Examples

Mitarbeiterhierarchie

Tabelleneinrichtung

```
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)
```

Allgemeiner Tabellenausdruck

```
;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
```

Ausgabe:

Vollständiger Name	EmpLevel	Der Name des Managers
Ken Sánchez	1	*Null*
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
Chloe Samuels	3	Žydre Klybe
George Weasley	3	Žydre Klybe
Michael Kensington	4	Sam Jackson

Finden Sie das n-te höchste Gehalt mit CTE

Mitarbeiter-Tabelle:

```
|  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 (Common Table Expression):

```
 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
```

Um das zweithöchste Gehalt zu finden, ersetzen Sie einfach N durch 2. Um das dritthöchste Gehalt zu ermitteln, ersetzen Sie einfach N durch 3.

Löschen Sie doppelte Zeilen mit CTE

Mitarbeiter-Tabelle:

```
|  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 (Common Table Expression):

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

Ausführungsergebnis:

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

Generieren Sie eine Datumstabelle mit 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)
```

In diesem Beispiel wird eine einspaltige Tabelle mit Datumsangaben zurückgegeben, die mit dem in der Variablen @startdate angegebenen Datum beginnen und den nächsten Datumswert von @numberDays zurückgeben.

Rekursiver CTE

Dieses Beispiel zeigt, wie Sie jedes Jahr von diesem Jahr bis 2011 (2012 - 1) erhalten.

```
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
```

mein Jahr
2016
2015
2014

mein Jahr
2013
2012
2011

Sie können die Rekursion (Think Stack-Überlauf im Code) mit MAXRECURSION als Abfrageoption steuern, um die Anzahl der rekursiven Aufrufe zu begrenzen.

```
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);
```

> Meldung 530, Ebene 16, Status 1, Zeile 2Die Anweisung wurde abgebrochen. Die maximale Rekursion 10 ist vor Abschluss der Anweisung erschöpft.

CTE mit mehreren AS-Anweisungen

```
;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
)
```

Mit allgemeinen Tabellenausdrücken können mehrere Abfragen mit durch Kommas getrennten AS-Anweisungen erstellt werden. Eine Abfrage kann dann auf viele verschiedene Arten auf eine

oder alle dieser Fragen verweisen und sie sogar verbinden.

Allgemeine Tabellenausdrücke online lesen: https://riptutorial.com/de/sql-server/topic/1343/allgemeine-tabellenausdrucke

Kapitel 9: Ändern Sie den JSON-Text

Examples

Ändern Sie den Wert in JSON-Text im angegebenen Pfad

Die Funktion JSON_MODIFY verwendet JSON-Text als Eingabeparameter und ändert einen Wert im angegebenen Pfad mit dem dritten 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}
```

Als Ergebnis erhalten wir neuen JSON-Text mit "Preis": 39,99 und andere Werte werden nicht geändert. Wenn das Objekt im angegebenen Pfad nicht vorhanden ist, fügt JSON_MODIFY ein Schlüssel / Wert-Paar ein.

Um das key: value-Paar zu löschen, setzen Sie NULL als neuen Wert:

```
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 löscht standardmäßig den Schlüssel, wenn er keinen Wert hat. Daher können Sie ihn zum Löschen eines Schlüssels verwenden.

Einen Skalarwert an ein JSON-Array anhängen

JSON_MODIFY verfügt über einen Anfügemodus, der den Wert an das Array anfügt.

```
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"]}
```

Wenn das Array im angegebenen Pfad nicht vorhanden ist, erstellt JSON_MODIFY (append) ein neues Array mit einem einzelnen Element:

```
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"]}
```

Ein neues JSON-Objekt in den JSON-Text einfügen

Mit der Funktion JSON_MODIFY können Sie JSON-Objekte in JSON-Text einfügen:

```
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}}
```

Da der dritte Parameter Text ist, müssen Sie ihn mit der JSON_QUERY-Funktion umschließen, um Text in JSON "umzuwandeln". Ohne diese "Umwandlung" behandelt JSON_MODIFY den dritten Parameter als Nur-Text- und Escape-Zeichen, bevor er als Zeichenfolgenwert eingefügt wird. Ohne JSON_QUERY werden folgende Ergebnisse angezeigt:

```
{"Id":1,"Name":"Toy Car","Price":'{\"Min\":34.99,\"Recommended\":45.49}'}
```

JSON_MODIFY fügt dieses Objekt ein, wenn es nicht vorhanden ist, oder löscht es, wenn der Wert des dritten Parameters NULL ist.

Fügen Sie ein neues JSON-Array ein, das mit der FOR JSON-Abfrage erstellt wurde

Sie können ein JSON-Objekt mithilfe der standardmäßigen SELECT-Abfrage mit der FOR JSON-Klausel generieren und als dritten Parameter in JSON-Text einfügen:

```
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 weiß, dass die Auswahlabfrage mit der FOR JSON-Klausel ein gültiges JSON-Array generiert und es einfach in den JSON-Text einfügt.

> Sie können alle FOR JSON-Optionen in der SELECT-Abfrage verwenden, **außer WITHOUT_ARRAY_WRAPPER** , das ein einzelnes Objekt anstelle eines JSON-Arrays generiert. In dem anderen Beispiel in diesem Thema erfahren Sie, wie ein einzelnes JSON-Objekt eingefügt wird.

Fügen Sie ein einzelnes JSON-Objekt ein, das mit der FOR JSON-Klausel generiert wurde

Sie können ein JSON-Objekt mithilfe der standardmäßigen SELECT-Abfrage mit der FOR JSON-Klausel und der Option WITHOUT_ARRAY_WRAPPER generieren und als dritten Parameter in JSON-Text einfügen:

```
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 mit der Option WITHOUT_ARRAY_WRAPPER generiert möglicherweise ungültigen JSON-Text, wenn die SELECT-Abfrage mehr als ein Ergebnis zurückgibt (in diesem Fall sollten Sie TOP 1 verwenden oder nach Primärschlüssel filtern). Daher nimmt JSON_MODIFY an, dass das zurückgegebene Ergebnis nur ein einfacher Text ist, und escape wie bei jedem anderen Text, wenn Sie es nicht mit der JSON_QUERY-Funktion umschließen.

> Sie sollten die Abfrage **FOR JSON, WITHOUT_ARRAY_WRAPPER** mit der Funktion **JSON_QUERY umschließen** , um das Ergebnis in JSON **umzuwandeln** .

Ändern Sie den JSON-Text online lesen: https://riptutorial.com/de/sql-server/topic/6883/andern-sie-den-json-text

Kapitel 10: ANSICHT ERSTELLEN

Examples

ANSICHT ERSTELLEN

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

Zeilen aus Ansichten können wie Tabellen ausgewählt werden:

```
SELECT FirstName
FROM view_EmployeeInfo
```

Sie können auch eine Ansicht mit einer berechneten Spalte erstellen. Wir können die Ansicht oben wie folgt ändern, indem Sie eine berechnete Spalte hinzufügen:

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

Diese Ansicht fügt eine zusätzliche Spalte hinzu, die `SELECT` wird, wenn Sie Zeilen daraus `SELECT` . Die Werte in dieser zusätzlichen Spalte hängen von den Feldern `FirstName` und `LastName` in der Tabelle `Employee` und werden automatisch aktualisiert, wenn diese Felder aktualisiert werden.

VIEW CREATE mit Verschlüsselung

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

ANSICHT ERSTELLEN mit 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
```

Ansichten können Joins verwenden, um Daten aus zahlreichen Quellen wie Tabellen, Tabellenfunktionen oder sogar anderen Ansichten auszuwählen. In diesem Beispiel werden die Spalten FirstName und LastName aus der Person-Tabelle und die Spalte JobTitle aus der Employee-Tabelle verwendet.

Diese Ansicht kann jetzt verwendet werden, um alle entsprechenden Zeilen für Manager in der Datenbank anzuzeigen:

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

ANZEIGE ERSTELLEN

Um eine Sicht mit einem Index zu erstellen, muss die Sicht mit den Schlüsselwörtern `WITH SCHEMABINDING` erstellt werden:

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

Jetzt können beliebige gruppierte oder nicht gruppierte Indizes erstellt werden:

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

Es gibt einige Einschränkungen für indizierte Sichten:

- Die Sichtdefinition kann auf eine oder mehrere Tabellen in derselben Datenbank verweisen.
- Nachdem der eindeutige gruppierte Index erstellt wurde, können zusätzliche nicht gruppierte Indizes für die Ansicht erstellt werden.
- Sie können die Daten in den zugrunde liegenden Tabellen aktualisieren - einschließlich

Einfügungen, Aktualisierungen, Löschungen und sogar Abschneiden.

- Sie können die zugrunde liegenden Tabellen und Spalten nicht ändern. Die Ansicht wird mit der Option WITH SCHEMABINDING erstellt.
- Es kann nicht COUNT, MIN, MAX, TOP, Outer Joins oder einige andere Schlüsselwörter oder Elemente enthalten.

Weitere Informationen zum Erstellen von indizierten Ansichten finden Sie in diesem MSDN-Artikel

Gruppierte Ansichten

Eine gruppierte Ansicht basiert auf einer Abfrage mit einer GROUP BY-Klausel. Da jede der Gruppen mehr als eine Zeile in der Basis haben kann, aus der sie erstellt wurde, sind diese notwendigerweise schreibgeschützt. Solche VIEWs verfügen normalerweise über eine oder mehrere Aggregatfunktionen und werden zu Berichtszwecken verwendet. Sie sind auch nützlich, um Schwachstellen in SQL zu umgehen. Betrachten Sie eine Ansicht, die den größten Verkauf in jedem Bundesstaat zeigt. Die Abfrage ist einfach:

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;
```

UNION-ed ANSICHTEN

VIEWs, die auf einer UNION- oder UNION ALL-Operation basieren, sind schreibgeschützt, da es keine Möglichkeit gibt, eine Änderung auf nur eine Zeile in einer der Basistabellen abzubilden. Der UNION-Operator entfernt doppelte Zeilen aus den Ergebnissen. Sowohl die UNION- als auch die UNION ALL-Operatoren verbergen, aus welcher Tabelle die Zeilen stammen. Solche VIEWs müssen ein verwenden, da die Spalten in einer UNION [ALL] keine eigenen Namen haben. Theoretisch sollte eine UNION aus zwei getrennten Tabellen, von denen keine doppelte Zeilen in sich hat, aktualisierbar sein.

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));
```

ANSICHT ERSTELLEN online lesen: https://riptutorial.com/de/sql-server/topic/3815/ansicht-

erstellen

Kapitel 11: Ansichten

Bemerkungen

Views sind gespeicherte Abfragen, die wie reguläre Tabellen abgefragt werden können. Ansichten sind nicht Teil des physischen Modells der Datenbank. Alle Änderungen, die auf die Datenquelle einer Ansicht angewendet werden, beispielsweise eine Tabelle, werden auch in der Ansicht angezeigt.

Examples

Ansicht erstellen

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

Ansicht erstellen oder ersetzen

Diese Abfrage löscht die Ansicht - sofern bereits vorhanden - und erstellt eine neue.

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

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

Erstellen Sie eine Ansicht mit Schemabindung

Wenn eine Ansicht WITH SCHEMABINDING erstellt wird, können die zugrunde liegenden Tabellen nicht gelöscht oder so geändert werden, dass sie die Ansicht zerstören. Beispielsweise kann eine Tabellenspalte, auf die in einer Ansicht verwiesen wird, nicht entfernt werden.

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

Ansichten *ohne* Schemabindung können brechen, wenn sich die zugrunde liegenden Tabellen ändern oder gelöscht werden. Das Abfragen einer unterbrochenen Ansicht führt zu einer Fehlermeldung. sp_refreshview kann verwendet werden, um sicherzustellen, dass vorhandene Ansichten ohne Schemabindung nicht beschädigt werden.

Ansichten online lesen: https://riptutorial.com/de/sql-server/topic/5327/ansichten

Kapitel 12: ANSONSTEN

Examples

Einzelne IF-Anweisung

Wie die meisten anderen Programmiersprachen unterstützt T-SQL auch IF..ELSE-Anweisungen.

Im folgenden Beispiel ist `1 = 1` der Ausdruck, der den `BEGIN..END` True `BEGIN..END` und das Steuerelement in den `BEGIN..END` Block eintritt und die Print-Anweisung die Zeichenfolge `'One is equal to One' BEGIN..END`

```
IF ( 1 = 1)  --<-- Some Expression
 BEGIN
   PRINT 'One is equal to One'
 END
```

Mehrere IF-Anweisungen

Wir können mehrere IF-Anweisungen verwenden, um mehrere Ausdrücke völlig unabhängig voneinander zu prüfen.

Im folgenden Beispiel wird der Ausdruck jeder `IF` Anweisung ausgewertet, und wenn sie wahr ist, wird der Code innerhalb des `BEGIN...END` Blocks ausgeführt. In diesem speziellen Beispiel sind der erste und der dritte Ausdruck wahr und nur diese Druckanweisungen werden ausgeführt.

```
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
```

Einzelne IF..ELSE-Anweisung

Wenn der Ausdruck in einer einzigen `IF..ELSE` Anweisung in der `IF` Anweisung als "True" ausgewertet wird, tritt die Steuerung in den ersten `BEGIN..END` Block ein und wird nur der Code innerhalb dieses Blocks ausgeführt, der Else-Block wird einfach ignoriert.

Wenn der Ausdruck dagegen als " `False` ausgewertet wird, wird der `ELSE BEGIN..END` Block ausgeführt und die Steuerung tritt niemals in den ersten `BEGIN..END` Block ein.

Im nachstehenden Beispiel wird der Ausdruck als "false" ausgewertet und der Else-Block wird ausgeführt, wobei die Zeichenfolge `'First expression was not true'` gedruckt `'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
```

Mehrere IF ... ELSE mit endgültigen ELSE-Anweisungen

Wenn wir Multiple `IF...ELSE IF` Anweisungen haben, aber auch etwas Code ausführen möchten, wenn keine der Ausdrücke als "True" ausgewertet wird, können wir einfach einen abschließenden `ELSE` Block hinzufügen, der nur ausgeführt wird, wenn keine der `IF` oder `ELSE IF` Ausdrücke werden als wahr ausgewertet.

Im folgenden Beispiel ist `'No other expression is true'` der `IF` noch der `ELSE IF` Ausdruck "True", daher wird nur der `ELSE` Block ausgeführt und `'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
```

Mehrere IF ... ELSE-Anweisungen

In den meisten Fällen müssen wir mehrere Ausdrücke überprüfen und auf der Grundlage dieser Ausdrücke spezifische Aktionen ausführen. Diese Situation wird mit mehreren `IF...ELSE IF` Anweisungen behandelt.

In diesem Beispiel werden alle Ausdrücke von oben nach unten ausgewertet. Sobald ein Ausdruck als wahr ausgewertet wird, wird der Code in diesem Block ausgeführt. Wenn kein Ausdruck als wahr ausgewertet wird, wird nichts ausgeführt.

```
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
```

ANSONSTEN online lesen: https://riptutorial.com/de/sql-server/topic/5186/ansonsten

Kapitel 13: Auslösen

Einführung

Ein Trigger ist ein spezieller Typ einer gespeicherten Prozedur, die automatisch ausgeführt wird, nachdem ein Ereignis aufgetreten ist. Es gibt zwei Arten von Auslösern: Datendefinitionssprachenauslöser und Datenmanipulationssprachauslöser.

Es ist normalerweise an einen Tisch gebunden und wird automatisch ausgelöst. Sie können keinen Auslöser explizit aufrufen.

Examples

Typen und Klassifizierungen des Auslösers

In SQL Server gibt es zwei Kategorien von Triggern: DDL-Trigger und DML-Trigger.

DDL-Trigger werden als Reaktion auf DDL-Ereignisse (Data Definition Language) ausgelöst. Diese Ereignisse entsprechen im Wesentlichen den Transact-SQL-Anweisungen, die mit den Schlüsselwörtern `CREATE` , `ALTER` und `DROP` .

DML-Trigger werden als Reaktion auf DML-Ereignisse (Data Manipulation Language) ausgelöst. Diese Ereignisse entsprechen Transact-SQL-Anweisungen, die mit den Schlüsselwörtern `INSERT` , `UPDATE` und `DELETE` .

DML-Trigger werden in zwei Haupttypen unterteilt:

1. After Trigger (für Trigger)

    - After Insert-Auslöser.
    - NACH UPDATE-Auslöser.
    - NACH DELETE-Auslöser.

2. Anstelle von Auslösern

    - INSTEAD OF INSERT-Auslöser.
    - STATT UPDATE Auslöser.
    - Anstelle von DELETE Trigger.

DML-Trigger

DML-Trigger werden als Antwort auf dml-Anweisungen ausgelöst (`insert` , `update` oder `delete`). Ein dml-Auslöser kann erstellt werden, um ein oder mehrere dml-Ereignisse für eine einzelne Tabelle oder Ansicht zu behandeln. Dies bedeutet, dass ein einzelner dml-Auslöser das Einfügen, Aktualisieren und Löschen von Datensätzen aus einer bestimmten Tabelle oder Ansicht übernehmen kann. In können jedoch nur Daten behandelt werden, die in dieser einzelnen Tabelle

oder Ansicht geändert werden.

DML-Trigger bieten Zugriff auf `inserted` und `deleted` Tabellen, die Informationen zu den Daten enthalten, die von der Einfüge-, Aktualisierungs- oder Löschanweisung, die den Trigger ausgelöst hat, betroffen waren / werden.

Beachten Sie, dass DML-Trigger auf Anweisungen und nicht auf Zeilen basieren. Das bedeutet, dass, wenn die Anweisung mehr als eine Zeile ausgeführt hat, die eingefügten oder gelöschten Tabellen mehr als eine Zeile enthalten.

Beispiele:

```
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
```

In allen obigen Beispielen werden Datensätze zu tblAudit hinzugefügt, wenn ein Datensatz in tblSomething hinzugefügt, gelöscht oder aktualisiert wird.

Auslösen online lesen: https://riptutorial.com/de/sql-server/topic/5032/auslosen

Kapitel 14: bcp (Massenkopierprogramm) Dienstprogramm

Einführung

Das Massenkopierprogramm (bcp) kopiert Daten zwischen einer Instanz von Microsoft SQL Server und einer Datendatei in einem benutzerdefinierten Format. Mit dem Dienstprogramm bcp können Sie eine große Anzahl neuer Zeilen in SQL Server-Tabellen importieren oder Daten aus Tabellen in Datendateien exportieren.

Examples

Beispiel zum Importieren von Daten ohne Formatdatei (im nativen Format)

```
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;"
```

bcp (Massenkopierprogramm) Dienstprogramm online lesen: https://riptutorial.com/de/sql-server/topic/10942/bcp--massenkopierprogramm--dienstprogramm

Kapitel 15: Begrenzung von Sonderzeichen und reservierten Wörtern

Bemerkungen

Im Allgemeinen ist es am besten, T-SQL Reserved Words nicht als Tabellennamen, Spaltennamen, Namen von Programmierobjekten, Aliasnamen usw. zu verwenden. Daher sollte die Methode zum Fluchen dieser Schlüsselwörter nur angewendet werden, wenn Sie einen Datenbankentwurf erben, der nicht geändert werden kann .

Bei reservierten Wörtern ist die Verwendung der eckigen Klammern nicht obligatorisch. Bei Verwendung eines Tools wie SQL Server Management Studio werden die reservierten Wörter hervorgehoben, um darauf aufmerksam zu machen, dass sie reserviert sind.

Examples

Grundmethode

Die grundlegende Methode, um reservierte Wörter für SQL Server zu schützen, ist die Verwendung der eckigen Klammern (`[` und `]`). Zum Beispiel sind *Beschreibung* und *Name* reservierte Wörter. Wenn jedoch ein Objekt beide als Namen verwendet, wird folgende Syntax verwendet:

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

Das einzige Sonderzeichen für SQL Server ist das Apostroph `'` und es wird entkam durch seine Verwendung zu verdoppeln. Um beispielsweise den Namen *O'Shea* in derselben Tabelle zu finden, wird folgende Syntax verwendet:

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

Begrenzung von Sonderzeichen und reservierten Wörtern online lesen: https://riptutorial.com/de/sql-server/topic/7156/begrenzung-von-sonderzeichen-und-reservierten-wortern

Kapitel 16: Beitreten

Einführung

In SQL (Structured Query Language) ist ein JOIN eine Methode zum Verknüpfen zweier Datentabellen in einer einzigen Abfrage, die es der Datenbank ermöglicht, einen Satz mit Daten aus beiden Tabellen gleichzeitig zurückzugeben, oder Daten aus einer Tabelle als Tabelle verwenden Filter auf der zweiten Tabelle. Im ANSI SQL-Standard sind verschiedene Arten von JOINs definiert.

Examples

Inner Join

`Inner join` gibt nur die Datensätze / Zeilen zurück, die basierend auf einer oder mehreren Bedingungen (angegeben mit dem Schlüsselwort `ON`) in beiden Tabellen übereinstimmen / existieren. Dies ist die häufigste Art der Verknüpfung. Die allgemeine Syntax für `inner join` lautet:

```
SELECT *
FROM table_1
INNER JOIN table_2
  ON table_1.column_name = table_2.column_name
```

Es kann auch als `JOIN` vereinfacht werden:

```
SELECT *
FROM table_1
JOIN table_2
  ON table_1.column_name = table_2.column_name
```

Beispiel

```
/* 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
```

Inner Join mit linkem Outer Join verwenden (Ersatz für Nicht vorhanden)

Diese Abfrage gibt Daten aus Tabelle 1 zurück, wobei Felder, die mit Tabelle2 mit einem Schlüssel übereinstimmen, und Daten, die nicht in Tabelle 1 enthalten sind, wenn mit Tabelle2 mit einer Bedingung und einem Schlüssel verglichen werden

```
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` ist eine kartesische Verknüpfung, dh ein kartesisches Produkt aus beiden Tabellen. Dieser Join benötigt keine Bedingung, um zwei Tabellen zu verbinden. Jede Zeile in der linken Tabelle wird mit jeder Zeile der rechten Tabelle verbunden. Syntax für einen Cross Join:

```
SELECT * FROM table_1
CROSS JOIN table_2
```

Beispiel:

```
/* 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;
```

Ergebnisse:

```
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
```

Beachten Sie, dass ein CROSS JOIN auf andere Arten angewendet werden kann. Dies ist ein *"alter Stil"* -Join (veraltet seit ANSI SQL-92) ohne Bedingung, was zu einem Kreuz / kartesischen Join führt:

```
SELECT *
FROM @Animal, @AnimalSound;
```

Diese Syntax funktioniert auch aufgrund einer Join-Bedingung "Always True", wird jedoch nicht empfohlen und sollte aus Gründen der Lesbarkeit zugunsten der expliziten `CROSS JOIN` Syntax vermieden werden.

```
SELECT *
FROM
    @Animal
    JOIN @AnimalSound
        ON 1=1
```

Äußere Join

Linke äußere Verbindung

`LEFT JOIN` gibt alle Zeilen der linken Tabelle zurück, die mit den Zeilen der rechten Tabelle übereinstimmen, in denen die Bedingungen der `ON` Klausel erfüllt sind. Zeilen, in denen die `ON` Klausel nicht erfüllt ist, weisen in allen Spalten der rechten Tabelle `NULL` auf. Die Syntax eines `LEFT JOIN` lautet:

```
SELECT * FROM table_1 AS t1
LEFT JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Rechter äußerer Join

`RIGHT JOIN` gibt alle Zeilen der rechten Tabelle zurück, die mit den Zeilen der linken Tabelle übereinstimmen, in denen die Bedingungen der `ON` Klausel erfüllt sind. Zeilen, in denen die `ON` Klausel nicht erfüllt ist, weisen in allen Spalten der linken Tabelle `NULL` auf. Die Syntax eines `RIGHT JOIN` lautet:

```
SELECT * FROM table_1 AS t1
RIGHT JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Voller äußerer Join

`FULL JOIN` kombiniert `LEFT JOIN` und `RIGHT JOIN` . Alle Zeilen werden aus beiden Tabellen zurückgegeben, unabhängig davon, ob die Bedingungen in der `ON` Klausel erfüllt sind. Zeilen, die die `ON` Klausel nicht erfüllen, werden in allen Spalten der gegenüberliegenden Tabelle mit `NULL` zurückgegeben (`NULL` für eine Zeile in der linken Tabelle enthalten alle Spalten in der rechten Tabelle `NULL` und umgekehrt). Die Syntax eines `FULL JOIN` lautet:

```
SELECT * FROM table_1 AS t1
FULL JOIN table_2 AS t2 ON t1.ID_Column = t2.ID_Column
```

Beispiele

```
/* 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. */
```

LINKE ÄUSSERE VERBINDUNG

```
SELECT *
FROM @Animal As t1
```

```
LEFT JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Ergebnisse für `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
```

RECHTE AUSSEN VERBINDEN

```
SELECT *
FROM @Animal As t1
RIGHT JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Ergebnisse für `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
```

VOLL AUSSEN MITGLIED

```
SELECT *
FROM @Animal As t1
FULL JOIN @AnimalSound As t2 ON t1.AnimalId = t2.AnimalId;
```

Ergebnisse für `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
```

Mitmachen bei einem Update

Joins können auch in einer `UPDATE` Anweisung verwendet werden:

```
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
)
```

Aktualisieren Sie die `SomeSetting` Spalte der Tabelle " `Preferences` mit einem Prädikat in der Tabelle `Users` , wie folgt:

```
UPDATE p
SET p.SomeSetting = 1
FROM Users u
JOIN Preferences p ON u.UserId = p.UserId
WHERE u.AccountId = 1234
```

`p` ist ein Alias für `Preferences` die in der `FROM` Klausel der Anweisung definiert sind. Nur Zeilen mit einer übereinstimmenden `AccountId` aus der Tabelle `Users` werden aktualisiert.

Aktualisieren Sie mit linken äußeren Join-Anweisungen

```
Update t
SET  t.Column1=100
FROM Table1 t LEFT JOIN Table12 t2
ON t2.ID=t.ID
```

Tabellen mit Inner Join- und Aggregatfunktion aktualisieren

```
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
```

Treten Sie einer Unterabfrage bei

Das Verknüpfen bei einer Unterabfrage wird häufig verwendet, wenn Sie aggregierte Daten (wie z. B. Count, Avg, Max oder Min) aus einer untergeordneten Tabelle / Details abrufen und diese zusammen mit Datensätzen aus der übergeordneten / Header-Tabelle anzeigen möchten. Beispielsweise möchten Sie möglicherweise die oberste / erste untergeordnete Zeile basierend auf Datum oder ID abrufen, oder Sie möchten vielleicht eine Zählung aller untergeordneten Zeilen oder einen Durchschnitt.

In diesem Beispiel werden Aliase verwendet, sodass Abfragen leichter lesbar sind, wenn mehrere Tabellen beteiligt sind. In diesem Fall rufen wir alle Zeilen aus der übergeordneten Tabelle "Bestellungen" und nur die letzte (oder letzte) untergeordnete Zeile aus der untergeordneten Tabelle "PurchaseOrderLineItems" ab. In diesem Beispiel wird davon ausgegangen, dass die untergeordnete Tabelle inkrementelle numerische IDs verwendet.

```
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
```

Selbst beitreten

Eine Tabelle kann in einem sogenannten Self-Join mit sich selbst verbunden werden, wobei Datensätze in der Tabelle mit anderen Datensätzen in derselben Tabelle kombiniert werden. Self-Joins werden normalerweise in Abfragen verwendet, in denen eine Hierarchie in den Tabellenspalten definiert ist.

Betrachten Sie die Beispieldaten in einer Tabelle mit dem Namen `Employees` :

ICH WÜRDE	Name	Boss_ID
1	Bob	3
2	Jim	1
3	Sam	2

Jeder Mitarbeiter `Boss_ID` Karten auf einem anderen Mitarbeiter - `ID` . Um eine Liste von Mitarbeitern mit dem Namen ihres jeweiligen Chefs abzurufen, kann die Tabelle mithilfe dieser Zuordnung in sich zusammengefügt werden. Beachten Sie, dass das Verknüpfen einer Tabelle auf diese Weise die Verwendung eines Alias (`Bosses`) in der zweiten Referenz der Tabelle erfordert, um sich von der Originaltabelle zu unterscheiden.

```
SELECT Employees.Name,
    Bosses.Name AS Boss
FROM Employees
INNER JOIN Employees AS Bosses
    ON Employees.Boss_ID = Bosses.ID
```

Beim Ausführen dieser Abfrage werden folgende Ergebnisse ausgegeben:

Name	Boss
Bob	Sam
Jim	Bob
Sam	Jim

Mit Join löschen

Joins können auch in einer `DELETE` Anweisung verwendet werden. Ein Schema wie folgt gegeben:

```
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
)
```

Wir können Zeilen aus der `Preferences` Tabelle löschen und nach einem Prädikat in der `Users` Tabelle wie folgt filtern:

```
DELETE p
FROM Users u
INNER JOIN Preferences p ON u.UserId = p.UserId
WHERE u.AccountId = 1234
```

Hier ist `p` ein Alias für `Preferences` die in der `FROM` Klausel der Anweisung definiert sind. Wir löschen nur Zeilen mit einer übereinstimmenden `AccountId` aus der Tabelle `Users` .

Versehentliches Verwandeln einer äußeren Verbindung in eine innere Verbindung

Äußere Joins geben alle Zeilen aus einer oder beiden Tabellen plus übereinstimmende Zeilen zurück.

```
Table People
PersonID FirstName
       1 Alice
       2 Bob
       3 Eve

Table Scores
PersonID Subject Score
       1 Math    100
       2 Math     54
       2 Science  98
```

Links bei den Tischen:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
```

Kehrt zurück:

```
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
```

Wenn Sie alle Personen mit den zutreffenden mathematischen Ergebnissen zurückgeben möchten, lautet ein häufiger Fehler:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
where Subject = 'Math'
```

Dies würde Eve aus Ihren Ergebnissen entfernen und zusätzlich Bobs Wissenschaftswert entfernen, da `Subject` für sie `NULL` .

Die korrekte Syntax zum Entfernen von Nicht-Mathematikdatensätzen unter Beibehaltung aller Personen in der Tabelle `People` lautet:

```
Select * from People a
left join Scores b
on a.PersonID = b.PersonID
and b.Subject = 'Math'
```

Beitreten online lesen: https://riptutorial.com/de/sql-server/topic/1008/beitreten

Kapitel 17: Benutzerdefinierte Tabellentypen

Einführung

Benutzerdefinierte Tabellentypen (kurz UDT) sind Datentypen, mit denen der Benutzer eine Tabellenstruktur definieren kann. Benutzerdefinierte Tabellentypen unterstützen Primärschlüssel, eindeutige Einschränkungen und Standardwerte.

Bemerkungen

UDTs haben folgende Einschränkungen -

- kann nicht als Spalte in einer Tabelle oder als Feld in einem strukturierten benutzerdefinierten Typ verwendet werden
- Ein nicht gruppierter Index kann nicht in einem UDT erstellt werden, es sei denn, der Index ist das Ergebnis der Erstellung einer PRIMARY KEY- oder UNIQUE-Einschränkung für den UDT
- Die UDT-Definition kann nach ihrer Erstellung NICHT geändert werden

Examples

Erstellen einer UDT mit einer einzelnen int-Spalte, die auch ein Primärschlüssel ist

```
CREATE TYPE dbo.Ids as TABLE
(
    Id int PRIMARY KEY
)
```

Erstellen eines UDT mit mehreren Spalten

```
CREATE TYPE MyComplexType as TABLE
(
    Id int,
    Name varchar(10)
)
```

Erstellen eines UDT mit einer eindeutigen Einschränkung:

```
CREATE TYPE MyUniqueNamesType as TABLE
(
    FirstName varchar(10),
    LastName varchar(10),
    UNIQUE (FirstName,LastName)
)
```

Hinweis: Einschränkungen in benutzerdefinierten Tabellentypen können nicht benannt werden.

Erstellen eines UDT mit einem Primärschlüssel und einer Spalte mit einem Standardwert:

```
CREATE TYPE MyUniqueNamesType as TABLE
(
    FirstName varchar(10),
    LastName varchar(10),
    CreateDate datetime default GETDATE()
    PRIMARY KEY (FirstName,LastName)
)
```

Benutzerdefinierte Tabellentypen online lesen: https://riptutorial.com/de/sql-server/topic/5280/benutzerdefinierte-tabellentypen

Kapitel 18: Berechnete Spalten

Examples

Eine Spalte wird aus einem Ausdruck berechnet

Eine berechnete Spalte wird aus einem Ausdruck berechnet, der andere Spalten in derselben Tabelle verwenden kann. Der Ausdruck kann ein nicht berechneter Spaltenname, eine Konstante, eine Funktion und eine beliebige Kombination davon sein, die mit einem oder mehreren Operatoren verbunden ist.

Erstellen Sie eine Tabelle mit einer berechneten Spalte

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

)
```

Der Wert wird automatisch berechnet und in der berechneten Spalte gespeichert, wenn andere Werte eingefügt werden.

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

Ein einfaches Beispiel, das wir normalerweise in Log-Tabellen verwenden

```
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())))
```

Dadurch ergibt sich eine Laufzeitdifferenz in Minuten, die sehr praktisch ist.

Berechnete Spalten online lesen: https://riptutorial.com/de/sql-server/topic/5561/berechnete-spalten

Kapitel 19: Berechtigungen und Sicherheit

Examples

Weisen Sie einem Benutzer Objektberechtigungen zu

In der Produktion ist es eine bewährte Methode, Ihre Daten zu schützen und es nur zuzulassen, dass Vorgänge daran über gespeicherte Verfahren durchgeführt werden. Dies bedeutet, dass Ihre Anwendung CRUD-Vorgänge für Ihre Daten nicht direkt ausführen kann und möglicherweise Probleme verursachen kann. Das Zuweisen von Berechtigungen ist eine zeitaufwändige, fummelige und generell aufwendige Aufgabe. Aus diesem Grund ist es oft einfacher, einen Teil der (erheblichen) Leistung des INFORMATION_SCHEMA er-Schemas zu nutzen, das in jeder SQL Server-Datenbank enthalten ist.

Statt einem Benutzer einzelne Berechtigungen einzeln zuzuweisen, führen Sie einfach das folgende Skript aus, kopieren Sie die Ausgabe und führen Sie sie dann in einem Abfragefenster aus.

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

Berechtigungen und Sicherheit online lesen: https://riptutorial.com/de/sql-server/topic/7929/berechtigungen-und-sicherheit

Kapitel 20: BULK Import

Examples

BULK INSERT mit Optionen

Sie können die Parsing-Regeln mit verschiedenen Optionen in WITH-Klausel anpassen:

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

In diesem Beispiel gibt CODEPAGE an, dass eine Quelldatei in der UTF-8-Datei und TERMINATORS eine Koma und eine neue Zeile sind.

BULK INSERT

Der Befehl BULK INSERT kann zum Importieren von Dateien in SQL Server verwendet werden:

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

Der Befehl BULK INSERT ordnet Spalten in Dateien mit Spalten in der Zieltabelle zu.

Lesen des gesamten Dateiinhalts mit OPENROWSET (BULK)

Sie können den Inhalt einer Datei mit der Funktion OPENROWSET (BULK) lesen und den Inhalt in einer Tabelle speichern:

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

Die Option SINGLE_BLOB liest den gesamten Inhalt aus einer Datei als einzelne Zelle.

Datei mit OPENROWSET (BULK) lesen und Datei formatieren

Sie können das Format der Datei definieren, die mit der Option FORMATFILE importiert wird:

```
INSERT INTO mytable
SELECT a.*
FROM OPENROWSET(BULK 'c:\test\values.txt',
   FORMATFILE = 'c:\test\values.fmt') AS a;
```

Die Formatdatei format_file.fmt beschreibt die Spalten in 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
```

Json-Datei mit OPENROWSET lesen (BULK)

Sie können OPENROWSET verwenden, um den Inhalt einer Datei zu lesen und an eine andere Funktion zu übergeben, die die Ergebnisse analysiert.

Das folgende Beispiel zeigt, wie der gesamte Inhalt der JSON-Datei mithilfe von OPENROWSET (BULK) gelesen werden kann. Anschließend können Sie der OPENJSON-Funktion BulkColumn zur Verfügung stellen, die JSON analysiert und Spalten zurückgibt:

```
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
```

BULK Import online lesen: https://riptutorial.com/de/sql-server/topic/7330/bulk-import

Kapitel 21: CASE-Erklärung

Bemerkungen

Das obige Beispiel dient nur zur Darstellung der Syntax für die Verwendung von case-Anweisungen in SQL Server mit Beispiel für Wochentage. Die gleiche Ausgabe kann auch mit "SELECT DATENAME (WEEKDAY, GETDATE ())" erzielt werden.

Examples

Einfache CASE-Anweisung

In einer einfachen Fallanweisung wird ein Wert oder eine Variable auf mehrere mögliche Antworten geprüft. Der folgende Code ist ein Beispiel für eine einfache Fallanweisung:

```
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
```

Gesuchte CASE-Anweisung

In einer gesuchten Case-Anweisung, kann jede Option prüfen, einen oder mehr unabhängig voneinander Werte. Der folgende Code ist ein Beispiel für eine gesuchte Case-Anweisung:

```
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
```

CASE-Erklärung online lesen: https://riptutorial.com/de/sql-server/topic/7238/case-erklarung

Kapitel 22: CLUSTERED COLUMNSTORE

Examples

Tabelle mit dem CLUSTERED COLUMNSTORE-Index

Wenn Sie eine Tabelle im Spaltenspeicherformat anstelle des Zeilenspeichers organisieren möchten, fügen Sie INDEX cci CLUSTERED COLUMNSTORE in der Definition der Tabelle hinzu:

```
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
)
```

COLUMSTORE-Tabellen eignen sich besser für Tabellen, bei denen vollständige Scans und Berichte erwartet werden, während Zeilenspeichertabellen besser für Tabellen geeignet sind, in denen Sie kleinere Zeilenmengen lesen oder aktualisieren.

Clustered Columnstore-Index für vorhandene Tabelle hinzufügen

Mit CREATE CLUSTERED COLUMNSTORE INDEX können Sie eine Tabelle im Spaltenformat organisieren:

```
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
```

Erstellen Sie den CLUSTERED COLUMNSTORE-Index neu

Wenn Sie viele gelöschte Zeilen haben, können Sie den Clusterindex für Spaltenspeicher neu erstellen:

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

Durch das Wiederherstellen von CLUSTERED COLUMNSTORE werden Daten aus der aktuellen Tabelle in die neue Tabelle "neu geladen", die Komprimierung wird erneut angewendet, gelöschte Zeilen werden entfernt usw.

Sie können eine oder mehrere Partitionen neu erstellen.

CLUSTERED COLUMNSTORE online lesen: https://riptutorial.com/de/sql-server/topic/5774/clustered-columnstore

Kapitel 23: Common Language Runtime Integration

Examples

Aktivieren Sie CLR für die Datenbank

CLR-Prozeduren sind standardmäßig nicht aktiviert. Sie müssen die folgenden Abfragen ausführen, um CLR zu aktivieren:

```
sp_configure 'show advanced options', 1;
GO
RECONFIGURE;
GO
sp_configure 'clr enabled', 1;
GO
RECONFIGURE;
GO
```

Wenn für ein CLR-Modul ein externer Zugriff erforderlich ist, sollten Sie die TRUSTWORTHY-Eigenschaft in Ihrer Datenbank auf ON setzen:

```
ALTER DATABASE MyDbWithClr SET TRUSTWORTHY ON
```

Hinzufügen von DLL-Dateien, die SQL-CLR-Module enthalten

In .Net-Sprachen geschriebene Prozeduren, Funktionen, Trigger und Typen werden in DLL-Dateien gespeichert. Nachdem Sie eine DLL-Datei mit CLR-Prozeduren erstellt haben, sollten Sie sie in SQL Server importieren:

```
CREATE ASSEMBLY MyLibrary
FROM 'C:\lib\MyStoredProcedures.dll'
    WITH PERMISSION_SET = EXTERNAL_ACCESS
```

PERMISSION_SET ist standardmäßig sicher, was bedeutet, dass Code in .dll keine Berechtigung für den Zugriff auf externe Ressourcen (z. B. Dateien, Websites, andere Server) benötigt und dass kein native Code verwendet wird, der auf den Speicher zugreifen kann.

PERMISSION_SET = EXTERNAL_ACCESS wird zum Markieren von Assemblys verwendet, die Code enthalten, der auf externe Ressourcen zugreift.

Informationen zu den aktuellen CLR-Assembly-Dateien finden Sie in der Ansicht sys.assemblies:

```
SELECT *
FROM sys.assemblies asms
WHERE is_user_defined = 1
```

Erstellen Sie eine CLR-Funktion in SQL Server

Wenn Sie eine .NET-Funktion erstellt, in eine DLL-Datei kompiliert und als Assembly in den SQL-Server importiert haben, können Sie eine benutzerdefinierte Funktion erstellen, die auf Funktionen in dieser Assembly verweist:

```
CREATE FUNCTION dbo.TextCompress(@input nvarchar(max))
RETURNS varbinary(max)
AS EXTERNAL NAME MyLibrary.[Name.Space.ClassName].TextCompress
```

Sie müssen den Namen der Funktion und die Signatur mit Eingabeparametern angeben und Werte zurückgeben, die der .Net-Funktion entsprechen. In der Klausel AS EXTERNAL NAME müssen Sie den Namen der Assembly, den Namespace / Class-Namen, an dem diese Funktion platziert ist, und den Namen der Methode in der Klasse angeben, die den Code enthält, der als Funktion verfügbar gemacht wird.

Informationen zu den CLR-Funktionen finden Sie in der folgenden Abfrage:

```
SELECT * FROM dbo.sysobjects WHERE TYPE ='FS'
```

Erstellen Sie einen benutzerdefinierten CLR-Typ in SQL Server

Wenn Sie eine .Net-Klasse erstellen, die einen benutzerdefinierten Typ darstellt, in eine DLL-Datei kompiliert und als Assembly in den SQL-Server importiert hat, können Sie eine benutzerdefinierte Funktion erstellen, die auf diese Klasse verweist:

```
CREATE TYPE dbo.Point
EXTERNAL NAME MyLibrary.[Name.Space.Point]
```

Sie müssen den Namen des Typs angeben, der in T-SQL-Abfragen verwendet wird. In der Klausel EXTERNAL NAME müssen Sie den Namen der Assembly, den Namespace und den Klassennamen angeben.

Erstellen Sie eine CLR-Prozedur in SQL Server

Wenn Sie die .Net-Methode in einer Klasse erstellt, in eine DLL-Datei kompiliert und als Assembly in den SQL-Server importiert haben, können Sie eine benutzerdefinierte gespeicherte Prozedur erstellen, die auf die Methode in dieser Assembly verweist:

```
CREATE PROCEDURE dbo.DoSomethng(@input nvarchar(max))
AS EXTERNAL NAME MyLibrary.[Name.Space.ClassName].DoSomething
```

Sie müssen den Namen der Prozedur und die Signatur mit Eingabeparametern angeben, die der .Net-Methode entsprechen. In der Klausel AS EXTERNAL NAME müssen Sie den Namen der Assembly, den Namespace / Class-Namen, an dem diese Prozedur platziert wird, und den Namen der Methode in der Klasse angeben, die den Code enthält, der als Prozedur verfügbar gemacht wird.

Common Language Runtime Integration online lesen: https://riptutorial.com/de/sql-server/topic/7116/common-language-runtime-integration

Kapitel 24: Cursor

Syntax

- DECLARE Cursorname CURSOR [LOCAL | GLOBAL]
  - [FORWARD_ONLY | SCROLL]
    [STATIC | KEYSET | DYNAMISCH | FAST_FORWARD]
    [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
    [TYPE_WARNING]
  - FOR select_statement
  - [FOR UPDATE [OF Spaltenname [, ... n]]]

Bemerkungen

Normalerweise möchten Sie die Verwendung von Cursors vermeiden, da diese die Leistung beeinträchtigen können. In einigen speziellen Fällen müssen Sie Ihren Datensatz jedoch Datensatz für Datensatz durchlaufen und einige Aktionen ausführen.

Examples

Einfacher Vorwärtspfeil

Normalerweise möchten Sie die Verwendung von Cursors vermeiden, da diese die Leistung beeinträchtigen können. In einigen speziellen Fällen müssen Sie Ihren Datensatz jedoch Datensatz für Datensatz durchlaufen und einige Aktionen ausführen.

```
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
```

Rudimentäre Cursor-Syntax

Eine einfache Cursor-Syntax, die an einigen Beispiel-Testzeilen ausgeführt wird:

```
/* 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;
```

Ergebnisse aus SSMS. Beachten Sie, dass es sich hierbei um separate Abfragen handelt, die in keiner Weise vereinheitlicht werden. Beachten Sie, wie die Abfrage-Engine jede Iteration einzeln

und nicht als Satz verarbeitet.

Ich würde	Val
1	Foo
(1 Zeile (n) betroffen)	

Ich würde	Val
2	Bar
(1 Zeile (n) betroffen)	

Ich würde	Val
3	Baz
(1 Zeile (n) betroffen)	

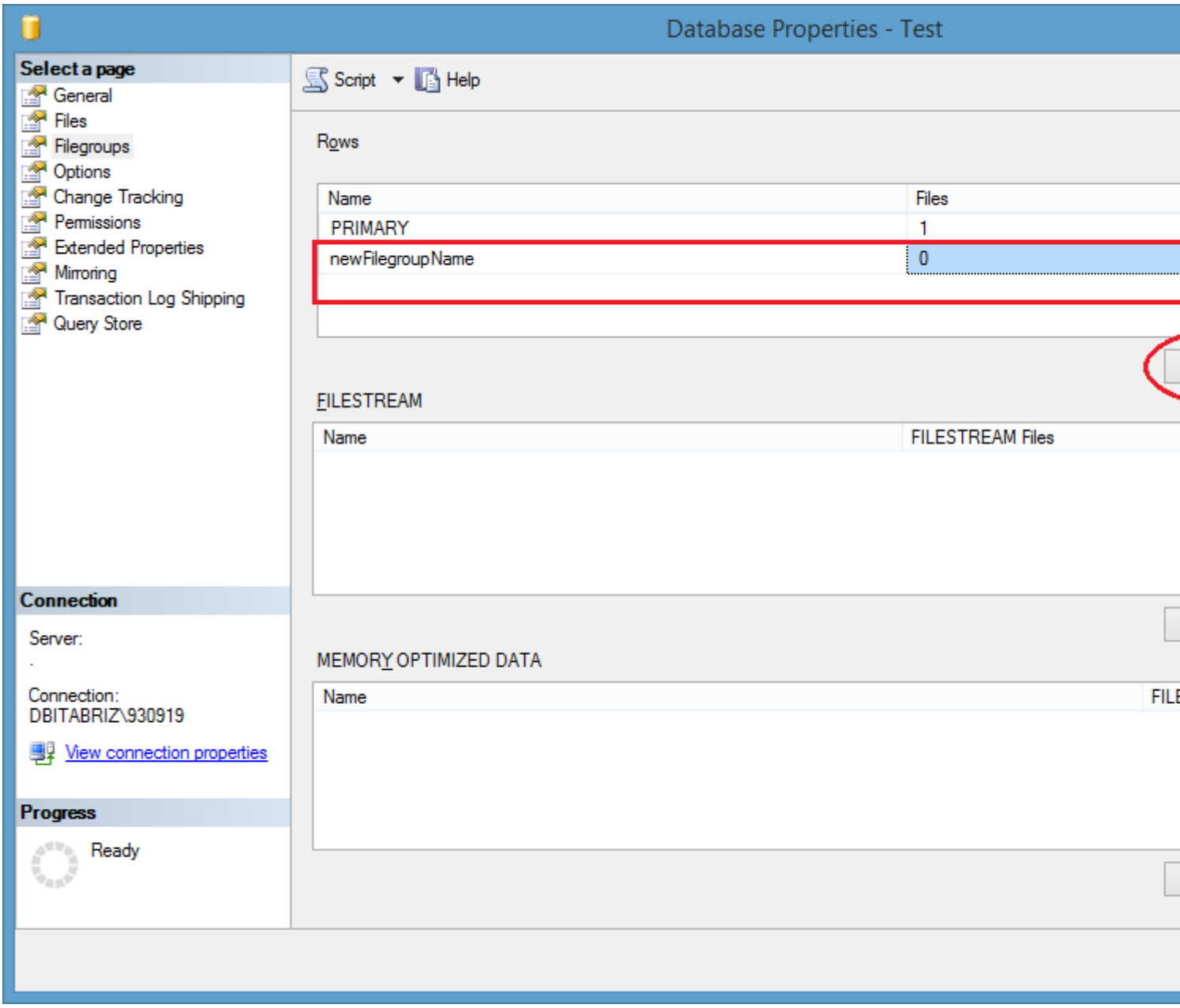
Cursor online lesen: https://riptutorial.com/de/sql-server/topic/870/cursor

Kapitel 25: Dateigruppe

Examples

Erstellen Sie eine Dateigruppe in der Datenbank

Wir können es auf zwei Wegen schaffen. Zuerst aus dem Datenbankeigenschaftsmodus:



Und von SQL-Skripten:

```
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
```

Dateigruppe online lesen: https://riptutorial.com/de/sql-server/topic/5461/dateigruppe

Kapitel 26: Datenbank sichern und wiederherstellen

Syntax

- BACKUP DATABASE- *Datenbank* TO *backup_device [, ... n]* WITH *with_options [, ... o]*
- RESTORE DATABASE *Datenbank* FROM *backup_device [, ... n]* WITH *with_options [, ... o]*

Parameter

Parameter	Einzelheiten
Datenbank	Der Name der Datenbank, die gesichert oder wiederhergestellt werden soll
Sicherungsgerät	Das Gerät, aus dem die Datenbank gesichert oder wiederhergestellt werden soll, wie {DISK oder TAPE}. Kann durch Kommas getrennt werden (,)
with_options	Verschiedene Optionen, die während des Vorgangs verwendet werden können. Wie beim Formatieren des Datenträgers, auf dem die Sicherung abgelegt werden soll, oder beim Wiederherstellen der Datenbank mit der Option zum Ersetzen.

Examples

Grundlegende Sicherung auf Festplatte ohne Optionen

Der folgende Befehl sichert die Datenbank *'Benutzer'* in der Datei *'D: \ DB_Backup'* . Es ist besser, keine Verlängerung zu geben.

```
BACKUP DATABASE Users TO DISK = 'D:\DB_Backup'
```

Grundlegende Wiederherstellung von der Festplatte ohne Optionen

Der folgende Befehl stellt die *Benutzerdatenbank* aus der Datei *'D: \ DB_Backup' wieder her* .

```
RESTORE DATABASE Users FROM DISK = 'D:\DB_Backup'
```

RESTORE-Datenbank mit REPLACE

Wenn Sie versuchen, die Datenbank von einem anderen Server wiederherzustellen, wird möglicherweise der folgende Fehler angezeigt:

> Fehler 3154: Der Sicherungssatz enthält eine Sicherung einer anderen Datenbank als

der vorhandenen Datenbank.

In diesem Fall sollten Sie die WITH REPLACE-Option verwenden, um die Datenbank durch die Datenbank aus der Sicherung zu ersetzen:

```
RESTORE DATABASE WWIDW
FROM DISK = 'C:\Backup\WideWorldImportersDW-Full.bak'
WITH REPLACE
```

Selbst in diesem Fall erhalten Sie möglicherweise die Fehlermeldung, dass sich Dateien nicht in einem bestimmten Pfad befinden können:

> Meldung 3156, Ebene 16, Status 3, Zeile 1 Die Datei 'WWI_Primary' kann nicht in 'D: \ Data \ WideWorldImportersDW.mdf' wiederhergestellt werden. Verwenden Sie WITH MOVE, um einen gültigen Speicherort für die Datei zu ermitteln.

Dieser Fehler tritt möglicherweise auf, weil Ihre Dateien nicht im selben Ordnerpfad abgelegt wurden, der auf dem neuen Server vorhanden ist. In diesem Fall sollten Sie einzelne Datenbankdateien an einen neuen Speicherort verschieben:

```
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'
```

Mit dieser Anweisung können Sie die Datenbank ersetzen, indem alle Datenbankdateien an einen neuen Speicherort verschoben werden.

Datenbank sichern und wiederherstellen online lesen: https://riptutorial.com/de/sql-server/topic/5826/datenbank-sichern-und-wiederherstellen

Kapitel 27: Datenbankberechtigungen

Bemerkungen

Grundlegende Syntax:

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

- {GRANT | REVOKE | DENY} - Was Sie erreichen wollen
  - Grant: "Diese Erlaubnis dem angegebenen Auftraggeber geben"
  - Widerruf: "Nehmen Sie diese Erlaubnis vom angegebenen Auftraggeber weg"
  - Verweigern: "Stellen Sie sicher, dass der angegebene Principal nie über diese Berechtigung verfügt (dh" `DENY SELECT` "bedeutet, dass unabhängig von anderen Berechtigungen `SELECT` für diesen Principal fehlschlägt).
- PERMISSION_NAME - Der Vorgang, den Sie ausführen möchten. Dies hängt von der Sicherheit ab. Zum Beispiel macht es keinen Sinn, `GRANT SELECT` für eine gespeicherte Prozedur zu verwenden.
- SECURABLE - Der Name der Sache, für die Sie die Berechtigungen beeinflussen möchten. Dies ist *optional* . `GRANT SELECT TO [aUser];` ist vollkommen akzeptabel; es bedeutet , „für ein zu sicherndes , für die die `SELECT` - Berechtigung Sinn macht, `GRANT` , dass die Erlaubnis“.
- PRINCIPAL - Für wen Sie versuchen, Berechtigungen zu beeinflussen. Auf Datenbankebene kann dies zum Beispiel eine Rolle (Anwendung oder Datenbank) oder ein Benutzer (zugeordnet zu einem Login oder nicht) sein.

Examples

Berechtigungen ändern

```
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];
```

BENUTZER ERSTELLEN

```
--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];
```

ROLLE ERSTELLEN

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

Rollenmitgliedschaft ändern

```
-- 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];
```

Hinweis: Rollenmitglieder können ein beliebiger Prinzipal auf Datenbankebene sein. Das heißt, Sie können eine Rolle als Mitglied in einer anderen Rolle hinzufügen. Das Hinzufügen / Löschen von Rollenmitgliedern ist idempotent. Das heißt, der Versuch des Hinzufügens / Ablegens hat zur Folge, dass die Rolle (bzw. deren Abwesenheit) unabhängig vom aktuellen Status ihrer Rollenmitgliedschaft ist.

Datenbankberechtigungen online lesen: https://riptutorial.com/de/sql-server/topic/6788/datenbankberechtigungen

Kapitel 28: Datenbank-Snapshots

Bemerkungen

Ein Datenbank-Snapshot ist eine schreibgeschützte, statische Ansicht einer SQL Server-Datenbank, die zum Zeitpunkt der Erstellung des Snapshots transaktionskonform mit der Quelldatenbank ist.

Ein Datenbank-Snapshot befindet sich immer in derselben Serverinstanz wie seine Quellendatenbank. Wenn die Quellendatenbank aktualisiert wird, wird der Datenbank-Snapshot aktualisiert.

Eine Momentaufnahme unterscheidet sich von einer Sicherung, da der Moment der Momentaufnahmeerstellung sofort erfolgt und die Momentaufnahme nur Speicherplatz beansprucht, wenn Änderungen in der Quellendatenbank vorgenommen werden. Eine Sicherung hingegen speichert eine vollständige Kopie der Daten zum Zeitpunkt der Sicherungserstellung.

Darüber hinaus bietet ein Snapshot eine sofortige Nur-Lese-Kopie der Datenbank, während eine Sicherung auf einem Server wiederhergestellt werden muss, damit sie lesbar ist (und nach dem Wiederherstellen auch geschrieben werden kann).

Datenbank-Momentaufnahmen sind nur in den Editionen Enterprise und Developer verfügbar.

Examples

Erstellen Sie einen Datenbank-Snapshot

Ein Datenbanksnapshot ist eine schreibgeschützte, statische Ansicht einer SQL Server-Datenbank (der Quellendatenbank). Es ist dem Backup ähnlich, aber es ist wie jede andere Datenbank verfügbar, sodass der Client die Momentaufnahmedatenbank abfragen kann.

```
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
```

Sie können auch eine Momentaufnahme der Datenbank mit mehreren Dateien erstellen:

```
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;
```

Wiederherstellen eines Datenbank-Snapshots

Wenn Daten in einer Quellendatenbank beschädigt werden oder falsche Daten in die Datenbank geschrieben werden, kann in einigen Fällen die Wiederherstellung der Datenbank auf einen Datenbank-Snapshot vor dem Schaden eine geeignete Alternative zum Wiederherstellen der Datenbank aus einer Sicherung darstellen.

```
RESTORE DATABASE MYDATABASE FROM DATABASE_SNAPSHOT='MyDatabase_morning';
```

> **Warnung:** Dadurch werden *alle Änderungen gelöscht* , die seit der Momentaufnahme der Quelldatenbank vorgenommen wurden.

LÖSCHEN Schnappschuss

Sie können vorhandene Momentaufnahmen der Datenbank mit der DELETE DATABASE-Anweisung löschen:

```
DROP DATABASE Mydatabase_morning
```

In dieser Anweisung sollten Sie den Namen des Datenbank-Snapshots angeben.

Datenbank-Snapshots online lesen: https://riptutorial.com/de/sql-server/topic/677/datenbank-snapshots

Kapitel 29: Datenfluss

Einführung

FILESTREAM integriert die SQL Server-Datenbank-Engine in ein NTFS-Dateisystem, indem varbinäre (max) Binärdaten (Binary Large Object, BLOB) als Dateien im Dateisystem gespeichert werden. Transact-SQL-Anweisungen können FILESTREAM-Daten einfügen, aktualisieren, abfragen, suchen und sichern. Win32-Dateisystemschnittstellen bieten Streaming-Zugriff auf die Daten.

Examples

Beispiel

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

Datenfluss online lesen: https://riptutorial.com/de/sql-server/topic/9509/datenfluss

Kapitel 30: Datentypen

Einführung

In diesem Abschnitt werden die Datentypen beschrieben, die SQL Server verwenden kann, einschließlich deren Datenbereich, Länge und Einschränkungen (falls vorhanden)

Examples

Genaue Numerik

Es gibt zwei grundlegende Klassen für genaue numerische Datentypen - **Integer** und **Fixed Precision und Scale** .

Ganzzahlige Datentypen

- bisschen
- Winzige
- smallint
- int
- Bigint

Ganzzahlen sind numerische Werte, die niemals einen Bruchteil enthalten und immer eine feste Speichermenge verwenden. Der Bereich und die Speichergrößen der Ganzzahldatentypen sind in dieser Tabelle dargestellt:

Datentyp	Angebot	Lager
bisschen	0 oder 1	1 bit **
Winzige	0 bis 255	1 Byte
smallint	-2 ^ 15 (-32.768) bis 2 ^ 15-1 (32.767)	2 Bytes
int	-2 ^ 31 (-2.147.483.648) bis 2 ^ 31-1 (2.147.483.647)	4 Bytes
Bigint	-2 ^ 63 (-9.223.372.036.854.775,808) bis 2 ^ 63-1 (9.223.372.036.854.775.807)	8 Bytes

Fixed Precision und Scale Datentypen

- numerisch
- Dezimal
- kleines geld

- Geld

Diese Datentypen sind nützlich, um Zahlen genau darzustellen. Solange die Werte in den Bereich der Werte passen können, die im Datentyp gespeichert werden können, hat der Wert keine Rundungsprobleme. Dies ist nützlich für alle Finanzberechnungen, bei denen Rundungsfehler bei Buchhaltern zu klinischem Wahnsinn führen.

Beachten Sie, dass **Dezimal** und **Numerisch** Synonyme für denselben Datentyp sind.

Datentyp	Angebot	Lager
Dezimal [(p [, s])] oder Numerisch [(p [, s])]]	-10 ^ 38 + 1 bis 10 ^ 38 - 1	Siehe **Präzisionstabelle**

Bei der Definition eines *dezimalen* oder *numerischen* Datentyps müssen Sie möglicherweise die Genauigkeit [p] und die Skala [s] angeben.

Präzision ist die Anzahl der Ziffern, die gespeichert werden können. Wenn Sie beispielsweise Werte zwischen 1 und 999 speichern müssen, benötigen Sie eine Genauigkeit von 3 (um die drei Ziffern in 100 zu halten). Wenn Sie keine Genauigkeit angeben, ist die Standardgenauigkeit 18.

Skala ist die Anzahl der Stellen nach dem Dezimalpunkt. Wenn Sie eine Zahl zwischen 0,00 und 999,99 speichern müssen, müssen Sie eine Genauigkeit von 5 (fünf Stellen) und eine Skala von 2 (zwei Stellen nach dem Dezimalzeichen) angeben. Sie müssen eine Genauigkeit angeben, um eine Skala festzulegen. Die Standardskala ist Null.

Die Genauigkeit eines *dezimalen* oder *numerischen* Datentyps definiert die Anzahl der zum Speichern des Werts erforderlichen Bytes (siehe unten):

Präzisionstabelle

Präzision	Speicherbytes
1 - 9	5
10-19	9
20-28	13
29-38	17

Monetäre feste Datentypen

Diese Datentypen sind speziell für Buchhaltungs- und andere monetäre Daten vorgesehen. Diese Typen haben eine feste Skala von 4 - Sie sehen immer vier Stellen nach der Dezimalstelle. Für die meisten Systeme, die mit den meisten Währungen arbeiten, reicht es aus, einen *numerischen* Wert mit einer Skala von 2 zu verwenden. Beachten Sie, dass mit dem Wert keine Informationen zum dargestellten Währungstyp gespeichert werden.

Datentyp	Angebot	Lager
Geld	-922.337.203,685,477,5808 bis 922,337,203,685,477,5807	8 Bytes
kleines geld	-214,748,3648 bis 214,748,3647	4 Bytes

Ungefähre Numerik

- Float [(**n**)]
- echt

Diese Datentypen werden zum Speichern von Gleitkommazahlen verwendet. Da diese Typen nur annähernd numerische Werte enthalten sollen, sollten diese nicht in Fällen verwendet werden, in denen Rundungsfehler nicht akzeptabel sind. Wenn Sie jedoch mit sehr großen Zahlen oder Zahlen mit einer unbestimmten Anzahl von Nachkommastellen umgehen müssen, sind diese möglicherweise die beste Option.

Datentyp	Angebot	Größe
schweben	-1,79E + 308 bis -2,23E-308, 0 und 2,23E-308 bis 1,79E + 308	hängt von **n** in der Tabelle unten ab
echt	-3,40E + 38 bis -1,18E - 38, 0 und 1,18E - 38 bis 3,40E + 38	4 Bytes

n Wertetabelle für *Float*- Nummern. Wenn in der Deklaration des Gleitkommas kein Wert angegeben ist, wird der Standardwert 53 verwendet. Beachten Sie, dass *Float (24)* einem *realen* Wert entspricht.

n Wert	Präzision	Größe
1-24	7 Ziffern	4 Bytes
25-53	15 Ziffern	8 Bytes

Datum und Uhrzeit

Diese Typen sind in allen Versionen von SQL Server enthalten

- Terminzeit
- Kleinzeit

Diese Typen sind in allen Versionen von SQL Server nach SQL Server 2012 enthalten

- Datum
- datetimeoffset
- datetime2
- Zeit

Zeichenketten

- verkohlen
- Varchar
- Text

Unicode-Zeichenfolgen

- nchar
- nvarchar
- ntext

Binäre Zeichenfolgen

- binär
- varbinary
- Bild

Andere Datentypen

- Mauszeiger
- Zeitstempel
- Hierarchieid
- eindeutige Kennung
- sql_variant
- xml
- Tabelle
- Räumliche Typen

Datentypen online lesen: https://riptutorial.com/de/sql-server/topic/5260/datentypen

Kapitel 31: Datentypen konvertieren

Examples

VERSUCHEN SIE PARSE

SQL Server 2012

Es konvertiert den String-Datentyp in den Zieldatentyp (Datum oder Numerisch).

Quelldaten sind beispielsweise ein String-Typ, und wir müssen den Datumstyp abdecken. Wenn der Konvertierungsversuch fehlschlägt, wird ein NULL-Wert zurückgegeben.

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

String_value - Dieses Argument ist ein Quellwert vom Typ NVARCHAR (4000).
Data_type - Dieses Argument ist entweder Zieldatum oder numerisch als Zieldatentyp.
Kultur - Dies ist ein optionales Argument, mit dessen Hilfe der Wert in das Kulturformat konvertiert werden kann. Angenommen, Sie möchten das Datum in Französisch anzeigen, müssen Sie den Kulturtyp als 'Fr-FR' übergeben. Wenn Sie keinen gültigen Kulturnamen übergeben, gibt PARSE einen Fehler aus.

```
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
```

TRY CONVERT

SQL Server 2012

Der Wert wird in den angegebenen Datentyp konvertiert. Wenn die Konvertierung fehlschlägt, wird NULL zurückgegeben. Beispiel: Quellwert im String-Format, und wir benötigen das Datums- / Ganzzahlformat. Dann wird uns dies helfen, dasselbe zu erreichen.

Syntax: TRY_CONVERT (Datentyp [(Länge)], Ausdruck [, Stil])

TRY_CONVERT () gibt einen Wert in den angegebenen Datentyp um, wenn die Umwandlung erfolgreich ist. Andernfalls wird null zurückgegeben.

Datentyp: Der Datentyp, in den konvertiert werden soll. Hier ist Länge ein optionaler Parameter,

der dazu dient, die angegebene Länge zu ermitteln.
Ausdruck - Der zu konvertierende Wert
Style - Dies ist ein optionaler Parameter, der die Formatierung bestimmt. Angenommen, Sie möchten ein Datumsformat wie "18. Mai 2013" und dann einen Pass-Style als 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
```

VERSUCHEN SIE CAST

SQL Server 2012

Der Wert wird in den angegebenen Datentyp konvertiert. Wenn die Konvertierung fehlschlägt, wird NULL zurückgegeben. Beispiel: Quellwert im String-Format und wir benötigen ihn im Double / Integer-Format. Dann hilft uns das dabei.

Syntax: TRY_CAST (Ausdruck AS data_type [(Länge)])

TRY_CAST () gibt einen Umwandlungswert in den angegebenen Datentyp zurück, wenn die Umwandlung erfolgreich ist. Andernfalls wird null zurückgegeben.

Ausdruck - Der Quellwert, der umgewandelt werden soll.
Data_type - Der Zieldatentyp, den der Quellwert umwandeln soll.
Länge - Dies ist ein optionaler Parameter, der die Länge des Zieldatentyps angibt.

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

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

Besetzung

Mit der Cast () - Funktion können Sie eine Datentypvariable oder Daten von einem Datentyp in einen anderen Datentyp konvertieren.

Syntax

CAST ([Ausdruck] AS-Datentyp)

Der Datentyp, in den Sie einen Ausdruck umwandeln, ist der Zieltyp. Der Datentyp des Ausdrucks, aus dem Sie konvertieren, ist der Quelltyp.

```
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.
```

Löst Fehler aus, wenn ein Fehler auftritt

Konvertieren

Wenn Sie Ausdrücke von einem Typ in einen anderen konvertieren, besteht in vielen Fällen die Notwendigkeit, Daten von einem Datetime-Typ in eine gespeicherte Prozedur oder eine andere Routine in einen Varchar-Typ zu konvertieren. Die Convert-Funktion wird für solche Dinge verwendet. Mit der Funktion CONVERT () können Datums- / Zeitdaten in verschiedenen Formaten angezeigt werden. Syntax

CONVERT (Datentyp (Länge), Ausdruck, Stil)

Stilstilwerte für die Konvertierung von Datetime oder Smalldatetime in Zeichendaten. Fügen Sie dem Stilwert 100 hinzu, um ein Jahr mit vier Stellen zu erhalten, das das Jahrhundert (JJJJ) einschließt.

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

13:27:16
```

Datentypen konvertieren online lesen: https://riptutorial.com/de/sql-server/topic/5034/datentypen-konvertieren

Kapitel 32: Datumsbereich generieren

Parameter

Parameter	Einzelheiten
@Ab Datum	Die inklusive untere Grenze des generierten Datumsbereichs.
@Miteinander ausgehen	Die inklusive obere Grenze des generierten Datumsbereichs.

Bemerkungen

Die meisten Experten scheinen eine Dates-Tabelle zu erstellen, anstatt eine Sequenz zu erzeugen. Siehe http://dba.stackexchange.com/questions/86435/filling-in-date-holes-in-grouped-by-date-sql-data

Examples

Datumsbereich mit rekursivem CTE erzeugen

Mit einem rekursiven CTE können Sie einen inklusiven Datumsbereich generieren:

```
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)
```

Die Standardeinstellung für `MaxRecursion` ist 100. `MaxRecursion` mit dieser Methode mehr als 100 Datumsangaben zu `Option (MaxRecursion N)` , ist das Segment `Option (MaxRecursion N)` der Abfrage erforderlich, wobei `N` die gewünschte `MaxRecursion` Einstellung ist. Wenn Sie diesen `MaxRecursion` auf `0` wird die `MaxRecursion` Einschränkung vollständig `MaxRecursion` .

Generieren eines Datumsbereichs mit einer Zählungstabelle

Eine andere Möglichkeit, einen Datumsbereich zu generieren, besteht darin, eine Datumstabelle zu verwenden, um die Daten zwischen dem Bereich zu erstellen:

```
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
```

Datumsbereich generieren online lesen: https://riptutorial.com/de/sql-server/topic/3232/datumsbereich-generieren

Kapitel 33: DBCC

Examples

DBCC-Wartungsbefehle

DBCC-Befehle ermöglichen es Benutzern, Speicherplatz in der Datenbank zu erhalten, Caches zu leeren, Datenbanken und Tabellen zu verkleinern.

Beispiele sind:

```
DBCC DROPCLEANBUFFERS
```

Entfernt alle sauberen Puffer aus dem Pufferpool und Columnstore-Objekte aus dem Columnstore-Objektpool.

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

Entfernt alle SQL-Abfragen im Plan-Cache. Jeder neue Plan wird neu kompiliert: Sie können einen Plan-Handle und einen Abfrage-Handle angeben, um die Pläne für den spezifischen Abfrageplan oder die SQL-Anweisung zu bereinigen.

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

Bereinigt alle zwischengespeicherten Einträge, die vom System erstellt wurden. Es kann Einträge in allen oder in einem bestimmten Ressourcenpool **bereinigen** (**myresourcepool** im obigen Beispiel).

```
DBCC FLUSHAUTHCACHE
```

Leert den Datenbankauthentifizierungscache mit Informationen zu Anmeldungen und Firewallregeln.

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

Verkleinert die Datenbank MyDB auf 10%. Der zweite Parameter ist optional. Sie können die Datenbank-ID anstelle des Namens verwenden.

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

Verkleinert die Datendatei mit dem Namen DataFile1 in der aktuellen Datenbank. Die Zielgröße beträgt 7 MB (dieser Parameter ist optional).

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

Fordert ein Leerzeichen aus der angegebenen Tabelle zurück

DBCC-Validierungsanweisungen

DBCC-Befehle ermöglichen es dem Benutzer, den Status der Datenbank zu überprüfen.

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

Die Prüfbeschränkung wird mit den nocheck-Optionen hinzugefügt. Sie wird daher nicht für vorhandene Daten geprüft. DBCC löst eine Einschränkungsprüfung aus.

Folgende DBCC-Befehle prüfen die Integrität von Datenbank, Tabelle oder Katalog:

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

DBCC-Informationsanweisungen

DBCC-Befehle können Informationen zu Datenbankobjekten anzeigen.

```
DBCC PROCCACHE
```

Zeigt Informationen in einem Tabellenformat über den Prozedurcache an.

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

Gibt den aktuellen Ausgabepuffer im Hexadezimal- und ASCII-Format für die angegebene session_id (und die optionale request_id) zurück.

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

Zeigt die letzte Anweisung an, die von einem Client an eine Instanz von Microsoft SQL Server gesendet wurde.

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

DBCC Trace-Befehle

Ablaufverfolgungsflags in SQL Server werden verwendet, um das Verhalten des SQL-Servers zu ändern und einige Funktionen zu aktivieren / deaktivieren. DBCC-Befehle können

Ablaufverfolgungsflags steuern:

Im folgenden Beispiel wird das Ablaufverfolgungsflag 3205 global und 3206 für die aktuelle Sitzung aktiviert:

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

Im folgenden Beispiel werden das Trace-Flag 3205 global und 3206 für die aktuelle Sitzung deaktiviert:

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

Das folgende Beispiel zeigt den Status der Trace-Flags 2528 und 3205:

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

DBCC-Anweisung

DBCC-Anweisungen fungieren als Datenbankkonsolenbefehle für SQL Server. Um die Syntaxinformationen für den angegebenen DBCC-Befehl zu erhalten, verwenden Sie die DBCC-Anweisung HELP (...).

Das folgende Beispiel gibt alle DBCC-Anweisungen zurück, für die Hilfe verfügbar ist:

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

Das folgende Beispiel gibt Optionen für die DBCC CHECKDB-Anweisung zurück:

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

DBCC online lesen: https://riptutorial.com/de/sql-server/topic/7316/dbcc

Kapitel 34: DBMAIL

Syntax

- sp_send_dbmail [[@profile_name =] 'profilname'] [, [@recipients =] 'Empfänger [; ... n] '] [, [@copy_recipients =]' copy_recipient [; ... n] '] [, [@blind_copy_recipients =]' blind_copy_recipient [; ... n] '] [, [@von_Adresse =]' von_Adresse '] [, [@reply_to =]' reply_to '] [, [@subject =]' subject '] [, [@body =]' body '] [, [@body_format =] 'body_format'] [, [@importance =] 'Wichtigkeit'] [, [@sensitivity =] 'Sensitivität'] [, [@file_attachments =] 'Anhang [; ... n] '] [, [@query =]' query '] [, [@execute_query_database =]' execute_query_database '] [, [@attach_query_result_as_file =] attach_query_result_as_file] [, [@query_attachment_filename =] query_attaching_date @query_result_header =] query_result_header] [, [@query_result_width =] query_result_width] [, [@query_result_separator =] 'query_result_separator'] [, [@clude_query_output =] exclude_query_output] [, [@endend_query_output]] query_no_truncate] [, [@query_result_no_padding =] @query_result_no_padding] [, [@mailitem_id =] mailitem_id] [OUTPUT]

Examples

Senden Sie eine einfache E-Mail

Dieser Code sendet eine einfache Text-E-Mail an Empfänger@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'
```

Ergebnisse einer Abfrage senden

Dadurch werden die Ergebnisse der Abfrage `SELECT * FROM Users` und an `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;
```

HTML-E-Mail senden

HTML-Inhalte müssen an `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>';
```

Verwenden Sie dann die Variable `@html` mit dem `@body argument` . Die HTML-Zeichenfolge kann auch direkt an `@body` , obwohl der Code dadurch möglicherweise schwieriger zu lesen ist.

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

DBMAIL online lesen: https://riptutorial.com/de/sql-server/topic/4908/dbmail

Kapitel 35: Die STUFF-Funktion

Parameter

Parameter	Einzelheiten
Zeichenausdruck	die vorhandene Zeichenfolge in Ihren Daten
Startposition	die Position in `character_expression` , um die `length` zu löschen und dann die `replacement_string` einzufügen
Länge	Die Anzahl der Zeichen, die aus `character_expression` gelöscht werden sollen
replace_string	Die Zeichenfolge, die in `character_expression` eingefügt werden soll

Examples

Grundzeichen ersetzen mit STUFF ()

Die Funktion `STUFF()` fügt eine Zeichenfolge in eine andere Zeichenfolge ein, indem zunächst eine angegebene Anzahl von Zeichen gelöscht wird. Im folgenden Beispiel wird "Svr" gelöscht und durch "Server" ersetzt. Dies geschieht durch Angabe der `start_position` und der `length` der Ersetzung.

```
SELECT STUFF('SQL Svr Documentation', 5, 3, 'Server')
```

Wenn Sie dieses Beispiel `SQL Svr Documentation.` wird die `SQL Server Documentation` anstelle der `SQL Svr Documentation.`

Verwenden von FOR XML zum Verketten von Werten aus mehreren Zeilen

Eine übliche Verwendung der `FOR XML` Funktion besteht darin, die Werte mehrerer Zeilen zu verketten.

Hier ein Beispiel mit der Customers-Tabelle :

```
SELECT
    STUFF( (SELECT ';' + Email
        FROM Customers
        where (Email is not null and Email <> '')
        ORDER BY Email ASC
        FOR XML PATH('')),
    1, 1, '')
```

Im obigen Beispiel wird `FOR XML PATH('')` zum Verketten von E-Mail-Adressen verwendet `;` als

Trennzeichen. Der Zweck von `STUFF` besteht auch darin, die führende `STUFF` zu entfernen `;` von der verketteten Zeichenfolge. `STUFF` setzt den verketteten String implizit auch von XML in varchar um.

Hinweis: Das Ergebnis des obigen Beispiels wird XML-codiert, dh es werden `<` Zeichen durch `<` Wenn Sie dies nicht möchten, ändern Sie `FOR XML PATH(''))` in `FOR XML PATH, TYPE).value('.[1]','varchar(MAX)')` , zB:

```
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, '')
```

Dies kann verwendet werden, um ein ähnliches Ergebnis wie `GROUP_CONCAT` in MySQL oder `string_agg` in PostgreSQL 9.0+ zu erzielen, obwohl anstelle von GROUP BY-Aggregaten Unterabfragen verwendet werden. (Alternativ können Sie ein benutzerdefiniertes Aggregat wie dieses installieren, wenn Sie eine Funktionalität suchen, die der von `GROUP_CONCAT`)

Spaltennamen mit Komma trennen (keine 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
```

Zeug für Komma getrennt in SQL Server

`FOR XML PATH` und `STUFF` zum Verketten der mehreren Zeilen in einer einzigen Zeile:

```
  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;
```

Grundbeispiel der STUFF () - Funktion.

STUFF (Originalausdruck, Start, Länge, Ersetzungsausdruck)

Die Funktion STUFF () fügt Replacement_expression an der angegebenen Startposition zusammen mit dem Entfernen der mit dem Parameter Length angegebenen Zeichen ein.

```
Select FirstName, LastName,Email, STUFF(Email, 2, 3, '*****') as StuffedEmail From Employee
```

Wenn Sie dieses Beispiel ausführen, wird die angegebene Tabelle zurückgegeben

Vorname	Nachname	Email	StuffedEmail
Jomes	Jäger	James@hotmail.com	J*****s@hotmail.com
Shyam	Rathod	Shyam@hotmail.com	S*****m@hotmail.com
RAM	shinde	Ram@hotmail.com	R ***** hotmail.com

Die STUFF-Funktion online lesen: https://riptutorial.com/de/sql-server/topic/703/die-stuff-funktion

Kapitel 36: Dienstmakler

Examples

1. Grundlagen

Service Broker ist eine Technologie, die auf asynchroner Kommunikation zwischen zwei (oder mehr) Entitäten basiert. Service Broker besteht aus: Nachrichtentypen, Verträgen, Warteschlangen, Diensten, Routen und mindestens Instanzendpunkten

Weitere Informationen : https://msdn.microsoft.com/de-de/library/bb522893.aspx

2. Aktivieren Sie den Service Broker für die Datenbank

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

3. Erstellen Sie eine grundlegende Service Broker-Konstruktion für die Datenbank (einzelne Datenbankkommunikation).

```
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
```

Wir brauchen keine Route für eine Datenbankkommunikation.

4. So senden Sie grundlegende Kommunikation über einen Service Broker

Für diese Demonstration verwenden wir eine Service-Broker-Konstruktion, die in einem anderen Teil dieser Dokumentation erstellt wurde. Der erwähnte Teil wird benannt. **3. Erstellen Sie eine grundlegende Service Broker-Konstruktion für die Datenbank (einzelne Datenbankkommunikation)** .

```
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;
```

Nach dieser Konversation wird Ihre Nachricht in TargetQueue angezeigt

5. So empfangen Sie Konversationen automatisch von TargetQueue

Für diese Demonstration verwenden wir eine Service-Broker-Konstruktion, die in einem anderen Teil dieser Dokumentation erstellt wurde. Der erwähnte Teil heißt **3. Erstellen Sie eine grundlegende Service-Broker-Konstruktion für die Datenbank (einzelne Datenbankkommunikation)** .

Zuerst müssen wir eine Prozedur erstellen, die Daten aus der Warteschlange lesen und verarbeiten kann

```
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
```

Zweiter Schritt: Erlauben Sie Ihrem TargetQueue, Ihre Prozedur automatisch auszuführen:

```
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  )
```

Dienstmakler online lesen: https://riptutorial.com/de/sql-server/topic/7651/dienstmakler

Kapitel 37: Dynamische Datenmaskierung

Examples

E-Mail-Adresse mit Dynamic Data Masking maskieren

Wenn Sie eine E-Mail-Spalte haben, können Sie diese mit der E-Mail-Maske () maskieren:

```
ALTER TABLE  Company
ALTER COLUMN Email ADD MASKED WITH (FUNCTION = 'email()')
```

Wenn der Benutzer versucht, E-Mails aus der Unternehmenstabelle auszuwählen, werden folgende Werte angezeigt:

mXXX@XXXX.com

zXXX@XXXX.com

rXXX@XXXX.com

Fügen Sie der Maske eine Teilmaske hinzu

Sie können in die Spalte eine Teilmaske einfügen, die nur wenige Zeichen vom Anfang und Ende der Zeichenfolge und der Zeichenmaske anstelle der Zeichen in der Mitte anzeigt:

```
ALTER TABLE  Company
ALTER COLUMN Phone ADD MASKED WITH (FUNCTION = 'partial(5,"XXXXXXX",2)')
```

In den Parametern der Partialfunktion können Sie festlegen, wie viele Werte von Anfang an angezeigt werden, wie viele Werte vom Ende angezeigt werden und welche Art von Muster in der Mitte angezeigt wird.

Wenn der Benutzer versucht, E-Mails aus der Unternehmenstabelle auszuwählen, werden folgende Werte angezeigt:

(381) XXXXXXX39

(360) XXXXXXX01

(415) XXXXXXX05

Anzeige eines Zufallswerts aus dem Bereich mithilfe der Maske random ()

Die Zufallsmaske zeigt eine zufällige Nummer aus dem angegebenen Bereich anstelle des tatsächlichen Werts:

```
ALTER TABLE  Product
```

```
ALTER COLUMN Price ADD MASKED WITH (FUNCTION = 'random(100,200)')
```

Beachten Sie, dass in einigen Fällen der angezeigte Wert mit dem tatsächlichen Wert in der Spalte übereinstimmt (wenn die zufällig ausgewählte Zahl dem Wert in der Zelle entspricht).

Standardmaske für die Spalte hinzufügen

Wenn Sie der Spalte eine Standardmaske hinzufügen, wird anstelle des tatsächlichen Werts in der SELECT-Anweisung eine Maske angezeigt:

```
ALTER TABLE  Company
ALTER COLUMN Postcode ADD MASKED WITH (FUNCTION = 'default()')
```

Kontrolle, wer unmaskierte Daten sehen kann

Sie können berechtigten Benutzern das Recht gewähren, mit der folgenden Anweisung nicht maskierte Werte anzuzeigen:

```
GRANT UNMASK TO MyUser
```

Wenn ein Benutzer bereits über die Berechtigung zum Aufheben der Maske verfügt, können Sie diese Berechtigung widerrufen:

```
REVOKE UNMASK TO MyUser
```

Dynamische Datenmaskierung online lesen: https://riptutorial.com/de/sql-server/topic/7052/dynamische-datenmaskierung

Kapitel 38: Dynamisches SQL

Examples

Führen Sie die als Zeichenfolge bereitgestellte SQL-Anweisung aus

In einigen Fällen müssen Sie eine SQL-Abfrage ausführen, die in einer Zeichenfolge platziert ist. EXEC, EXECUTE oder die Systemprozedur sp_executesql kann jede als String bereitgestellte SQL-Abfrage ausführen:

```
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')
```

Diese Prozedur gibt die gleiche Ergebnismenge zurück wie die SQL-Abfrage, die als Anweisungstext bereitgestellt wird. sp_executesql kann eine SQL-Abfrage ausführen, die als String-Literal, Variable / Parameter oder sogar als Ausdruck bereitgestellt wird:

```
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
```

Sie benötigen die Funktion QUOTENAME, um Sonderzeichen in der @ tabellenvariablen zu kennzeichnen. Ohne diese Funktion würden Sie einen Syntaxfehler erhalten, wenn die Variable @table etwas wie Leerzeichen, Klammern oder andere Sonderzeichen enthält.

Dynamic SQL wird als anderer Benutzer ausgeführt

Sie können die SQL-Abfrage als unterschiedlichen Benutzer mit AS USER = 'Name des Datenbankbenutzers' ausführen.

```
EXEC(N'SELECT * FROM product') AS USER = 'dbo'
```

Die SQL-Abfrage wird unter dbo-Datenbankbenutzer ausgeführt. Alle Berechtigungsprüfungen für den Dbo-Benutzer werden in der SQL-Abfrage geprüft.

SQL Injection mit dynamischem SQL

Dynamische Abfragen sind

```
SET @sql = N'SELECT COUNT(*) FROM AppUsers WHERE Username = ''' + @user + ''' AND Password =
''' + @pass + ''''
EXEC(@sql)
```

Wenn der Wert der Benutzervariable ***myusername '' ODER 1 = 1 ist, wird*** die folgende Abfrage ausgeführt:

```
SELECT COUNT(*)
FROM AppUsers
WHERE Username = 'myusername' OR 1=1 --' AND Password = ''
```

Der Kommentar am Ende des Werts der Variablen @username wird am Ende der Abfrage auskommentiert und die Bedingung 1 = 1 wird ausgewertet. Anwendung, die überprüft, dass mindestens ein Benutzer, der von dieser Abfrage zurückgegeben wird, einen Wert größer als 0 zurückgibt und die Anmeldung erfolgreich ist.

Mit diesem Ansatz kann sich der Angreifer in die Anwendung einloggen, auch wenn er keinen gültigen Benutzernamen und kein gültiges Kennwort kennt.

Dynamisches SQL mit Parametern

Um Injection- und Escape-Probleme zu vermeiden, sollten dynamische SQL-Abfragen mit Parametern ausgeführt werden, z.

```
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
```

Der zweite Parameter ist eine Liste von Parametern, die in der Abfrage mit ihren Typen verwendet werden. Nach dieser Liste werden Variablen bereitgestellt, die als Parameterwerte verwendet werden.

sp_executesql gibt Sonderzeichen aus und führt eine SQL-Abfrage aus.

Dynamisches SQL online lesen: https://riptutorial.com/de/sql-server/topic/6871/dynamisches-sql

Kapitel 39: Dynamisches SQL-Pivot

Einführung

In diesem Thema wird beschrieben, wie Sie ein dynamisches Pivot in SQL Server ausführen.

Examples

Grundlegendes dynamisches SQL-Pivot

```
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
```

Dynamisches SQL-Pivot online lesen: https://riptutorial.com/de/sql-server/topic/10751/dynamisches-sql-pivot

Kapitel 40: Eine Abfrage analysieren

Examples

Scan vs Seek

Wenn Sie einen Ausführungsplan anzeigen, stellen Sie möglicherweise fest, dass sich SQL Server für eine Suche oder einen Scan entschieden hat.

Ein Suchvorgang tritt auf, wenn SQL Server weiß, wohin es gehen muss, und nur bestimmte Elemente greifen soll. Dies tritt normalerweise auf, wenn gute Filter in eine Abfrage eingefügt werden, z. B. `where name = 'Foo'` .

Ein Scan ist, wenn SQL Server nicht genau weiß, wo sich alle benötigten Daten befinden, oder entschieden hat, dass der Scan effizienter ist als ein Suchvorgang, wenn genügend Daten ausgewählt sind.

Suchvorgänge sind normalerweise schneller, da sie nur einen Unterabschnitt der Daten erfassen, wohingegen Scans einen Großteil der Daten auswählen.

Eine Abfrage analysieren online lesen: https://riptutorial.com/de/sql-server/topic/7713/eine-abfrage-analysieren

Kapitel 41: Einfügen

Examples

Fügen Sie einer Tabelle mit dem Namen Invoices eine Zeile hinzu

```
INSERT INTO Invoices [ /* column names may go here */ ]
VALUES (123, '1234abc', '2016-08-05 20:18:25.770', 321, 5, '2016-08-04');
```

- Spaltennamen sind erforderlich, wenn die Tabelle, in die Sie einfügen, eine Spalte mit dem Attribut IDENTITY enthält.

```
INSERT INTO Invoices ([ID], [Num], [DateTime], [Total], [Term], [DueDate])
VALUES (123, '1234abc', '2016-08-05 20:18:25.770', 321, 5, '2016-08-25');
```

Einfügen online lesen: https://riptutorial.com/de/sql-server/topic/5323/einfugen

Kapitel 42: EINFÜGEN IN

Einführung

Die Anweisung INSERT INTO wird verwendet, um neue Datensätze in eine Tabelle einzufügen.

Examples

INSERT Hello World INTO-Tabelle

```
CREATE TABLE MyTableName
(
    Id INT,
    MyColumnName NVARCHAR(1000)
)
GO

INSERT INTO MyTableName (Id, MyColumnName)
VALUES (1, N'Hello World!')
GO
```

INSERT für bestimmte Spalten

Um bestimmte Spalten (nicht alle Spalten) einzufügen, müssen Sie die Spalten angeben, die Sie aktualisieren möchten.

```
INSERT INTO USERS (FIRST_NAME, LAST_NAME)
VALUES ('Stephen', 'Jiang');
```

Dies funktioniert nur, wenn die Spalten, die Sie nicht auflisten, Nullwerte, Identität, Zeitstempeldatentyp oder berechnete Spalten sind. oder Spalten mit einer Standardwerteinschränkung. Wenn einer dieser Spalten nicht nullfähige, nicht identifizierte, nicht zeitgesteuerte, nicht berechnete und nicht standardmäßige Spalten enthält, wird beim Versuch, diese Art des Einfügens zu versuchen, eine Fehlernachricht ausgegeben, die Sie dazu auffordert, eine Wert für das / die zutreffenden Feld (e).

FÜGEN Sie mehrere Datenzeilen ein

So fügen Sie mehrere Datenzeilen in SQL Server 2008 oder höher ein:

```
INSERT INTO USERS VALUES
(2, 'Michael', 'Blythe'),
(3, 'Linda', 'Mitchell'),
(4, 'Jillian', 'Carson'),
(5, 'Garrett', 'Vargas');
```

Um mehrere Datenzeilen in früheren Versionen von SQL Server einzufügen, verwenden Sie

"UNION ALL" wie folgt:

```
INSERT INTO USERS (FIRST_NAME, LAST_NAME)
SELECT 'James', 'Bond' UNION ALL
SELECT 'Miss', 'Moneypenny' UNION ALL
SELECT 'Raoul', 'Silva'
```

Beachten Sie, dass das Schlüsselwort "INTO" in INSERT-Abfragen optional ist. Eine weitere Warnung ist, dass der SQL Server nur 1000 Zeilen in einer INSERT unterstützt. Sie müssen also in Stapel aufgeteilt werden.

FÜGEN Sie eine einzelne Datenzeile ein

Eine einzelne Datenzeile kann auf zwei Arten eingefügt werden:

```
INSERT INTO USERS(Id, FirstName, LastName)
VALUES (1, 'Mike', 'Jones');
```

Oder

```
INSERT INTO USERS
VALUES (1, 'Mike', 'Jones');
```

Beachten Sie, dass die zweite Einfügeanweisung nur die Werte in genau derselben Reihenfolge wie die Tabellenspalten zulässt, während in der ersten Einfügung die Reihenfolge der Werte wie folgt geändert werden kann:

```
INSERT INTO USERS(FirstName, LastName, Id)
VALUES ('Mike', 'Jones', 1);
```

Verwenden Sie OUTPUT, um die neue ID zu erhalten

Beim EINFÜGEN können Sie `OUTPUT INSERTED.ColumnName` , um Werte aus der neu eingefügten Zeile `OUTPUT INSERTED.ColumnName` , z. B. die neu generierte Id - nützlich, wenn Sie eine `IDENTITY` Spalte oder eine Art Standard- oder berechneten Wert haben.

Wenn Sie dies programmgesteuert aufrufen (z. B. von ADO.net), würden Sie es als normale Abfrage behandeln und die Werte lesen, als hätten Sie eine `SELECT` -Anweisung gemacht.

```
-- CREATE TABLE OutputTest ([Id] INT NOT NULL PRIMARY KEY IDENTITY, [Name] NVARCHAR(50))

INSERT INTO OutputTest ([Name])
OUTPUT INSERTED.[Id]
VALUES ('Testing')
```

Wenn die ID der kürzlich hinzugefügten Zeile innerhalb derselben Abfrage- oder gespeicherten Prozedur erforderlich ist.

```
-- 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)
```

EINFÜGEN aus SELECT-Abfrageergebnissen

So fügen Sie aus einer SQL-Abfrage abgerufene Daten ein (einzelne oder mehrere Zeilen)

```
INSERT INTO Table_name (FirstName, LastName, Position)
SELECT FirstName, LastName, 'student' FROM Another_table_name
```

Beachten Sie, dass 'student' in SELECT eine String-Konstante ist, die in jede Zeile eingefügt wird.

Bei Bedarf können Sie Daten aus derselben Tabelle auswählen und einfügen

EINFÜGEN IN online lesen: https://riptutorial.com/de/sql-server/topic/3814/einfugen-in

Kapitel 43: Ergebnismenge begrenzen

Einführung

Wenn Datenbanktabellen wachsen, ist es oft nützlich, die Ergebnisse von Abfragen auf eine feste Anzahl oder einen bestimmten Prozentsatz zu beschränken. Dies kann mit dem Schlüsselwort `TOP` von SQL Server oder der Klausel `OFFSET FETCH` .

Parameter

Parameter	Einzelheiten
`TOP`	Begrenzen des Schlüsselworts Verwenden Sie mit einer Nummer.
`PERCENT`	Prozentsatz Keyword Kommt nach `TOP` und Begrenzungsnummer.

Bemerkungen

Wenn die `ORDER BY` Klausel verwendet wird, gilt die Einschränkung für die geordnete Ergebnismenge.

Examples

Begrenzung mit TOP

Dieses Beispiel begrenzt das `SELECT` Ergebnis auf 100 Zeilen.

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

Es ist auch möglich, eine Variable zu verwenden, um die Anzahl der Zeilen anzugeben:

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

Begrenzung mit PERCENT

In diesem Beispiel wird das `SELECT` Ergebnis auf 15 Prozent der gesamten Zeilenanzahl begrenzt.

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

Begrenzung mit FETCH

SQL Server 2012

`FETCH` ist im Allgemeinen für die Paginierung nützlicher, kann jedoch als Alternative zu `TOP` :

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

Ergebnismenge begrenzen online lesen: https://riptutorial.com/de/sql-server/topic/1555/ergebnismenge-begrenzen

Kapitel 44: Erweiterte Optionen

Examples

Aktivieren und zeigen Sie erweiterte Optionen

```
Exec sp_configure 'show advanced options' ,1
RECONFIGURE
GO
-- Show all configure
sp_configure
```

Aktivieren Sie die Standardeinstellung für die Sicherungskomprimierung

```
Exec sp_configure 'backup compression default',1
GO
RECONFIGURE;
```

Legen Sie den Standardfüllfaktor in Prozent fest

```
sp_configure 'fill factor', 100;
GO
RECONFIGURE;
```

Der Server muss neu gestartet werden, bevor die Änderung wirksam werden kann.

Legen Sie das Systemwiederherstellungsintervall fest

```
USE master;
GO
-- Set recovery every 3 min
EXEC sp_configure 'recovery interval', '3';
RECONFIGURE WITH OVERRIDE;
```

Aktivieren Sie die Cmd-Berechtigung

```
EXEC sp_configure 'xp_cmdshell', 1
GO
RECONFIGURE
```

Legen Sie die maximale Serverspeichergröße fest

```
USE master
EXEC sp_configure 'max server memory (MB)', 64
RECONFIGURE WITH OVERRIDE
```

Legen Sie die Anzahl der Prüfpunktaufgaben fest

```
EXEC sp_configure "number of checkpoint tasks", 4
```

Erweiterte Optionen online lesen: https://riptutorial.com/de/sql-server/topic/5185/erweiterte-optionen

Kapitel 45: Exportieren Sie Daten in TXT-Datei mithilfe von SQLCMD

Syntax

- sqlcmd -S SHERAZM-E7450 \ SQL2008R2 -d Baseline_DB_Aug_2016 -o c: \ mitarbeiter.txt

Examples

Mit SQLCMD an der Eingabeaufforderung

Befehlsstruktur ist

sqlcmd -S IhrServername \ Instanzname -d Datenbankname -o Ausgabedateiname_mitPfad

Die Schalter sind wie folgt

-S für Servername und Instanzname

-d für die Quellendatenbank

-o für Zielausgabedatei (erstellt Ausgabedatei)

-Q für Abfrage zum Abrufen von Daten

Exportieren Sie Daten in TXT-Datei mithilfe von SQLCMD online lesen: https://riptutorial.com/de/sql-server/topic/7076/exportieren-sie-daten-in-txt-datei-mithilfe-von-sqlcmd

Kapitel 46: Fensterfunktionen

Examples

Zentrierter gleitender Durchschnitt

Berechnen Sie einen 6-monatigen gleitenden Durchschnitt (126 Arbeitstage) eines Preises:

```
SELECT TradeDate, AVG(Px) OVER (ORDER BY TradeDate ROWS BETWEEN 63 PRECEDING AND 63 FOLLOWING)
AS PxMovingAverage
FROM HistoricalPrices
```

Beachten Sie, dass es vor und nach jeder zurückgegebenen Zeile *bis zu* 63 Zeilen dauern wird. Am Anfang und Ende des TradeDate-Bereichs wird es nicht zentriert: Wenn es das größte TradeDate erreicht, kann es nur 63 vorangehende Werte finden im Durchschnitt enthalten.

Suchen Sie das neueste Element in einer Liste mit zeitgestempelten Ereignissen

In Tabellen, in denen Ereignisse aufgezeichnet werden, gibt es häufig ein Datumsfeld, in dem die Zeit erfasst wird, zu der ein Ereignis stattgefunden hat. Die Suche nach dem neuesten Ereignis kann schwierig sein, da immer zwei Ereignisse mit exakt identischen Zeitstempeln aufgezeichnet wurden. Sie können row_number () over (order by ...) verwenden, um sicherzustellen, dass alle Datensätze eindeutig angeordnet sind, und wählen Sie den obersten Datensatz aus (my_ranking = 1).

```
select *
from (
    select
        *,
        row_number() over (order by crdate desc) as my_ranking
    from sys.sysobjects
) g
where my_ranking=1
```

Dieselbe Technik kann verwendet werden, um eine einzelne Zeile aus einem Dataset mit möglicherweise doppelten Werten zurückzugeben.

Durchschnitt der letzten 30 Elemente

Durchschnitt der letzten 30 verkauften Artikel

```
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
```

Fensterfunktionen online lesen: https://riptutorial.com/de/sql-server/topic/3209/fensterfunktionen

Kapitel 47: Fremde Schlüssel

Examples

Fremdschlüsselbeziehung / Einschränkung

Mit Fremdschlüsseln können Sie die Beziehung zwischen zwei Tabellen definieren. Eine (übergeordnete) Tabelle muss über einen Primärschlüssel verfügen, der Zeilen in der Tabelle eindeutig identifiziert. Eine andere (untergeordnete) Tabelle kann in einer der Spalten den Wert des Primärschlüssels vom übergeordneten Element haben. Die FOREIGN KEY REFERENCES-Einschränkung stellt sicher, dass Werte in der untergeordneten Tabelle als Primärschlüsselwert in der übergeordneten Tabelle vorhanden sein müssen.

In diesem Beispiel haben wir eine übergeordnete Company-Tabelle mit dem Primärschlüssel CompanyId und eine untergeordnete Employee-Tabelle mit der ID der Firma, in der dieser Mitarbeiter arbeitet.

```
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)
)
```

Fremdschlüsselreferenzen stellen sicher, dass Werte, die in die Spalte Employee.CompanyId eingefügt werden, auch in der Spalte Company.CompanyId vorhanden sein müssen. Außerdem kann niemand die Firma in der Firmentabelle löschen, wenn mindestens ein Mitarbeiter mit einer passenden CompanyId in der Child-Tabelle vorhanden ist.

Durch die FOREIGN KEY-Beziehung wird sichergestellt, dass Zeilen in zwei Tabellen nicht "nicht verknüpft" werden können.

Beibehaltung der Beziehung zwischen übergeordneten / untergeordneten Zeilen

Nehmen wir an, wir haben eine Zeile in der Company-Tabelle mit companyId 1. Wir können eine Zeile in die Employee-Tabelle einfügen, die companyId 1 hat:

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

Wir können jedoch keinen Mitarbeiter einfügen, der über eine nicht vorhandene CompanyId verfügt:

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

Meldung 547, Ebene 16, Status 0, Zeile 12 Die INSERT-Anweisung steht in Konflikt mit der FOREIGN KEY-Einschränkung "FK__Employee__Compan__1EE485AA". Der Konflikt ist in der Datenbank "MyDb", Tabelle "dbo.Company", Spalte "CompanyId" aufgetreten. Die Anweisung wurde beendet.

Außerdem können wir die übergeordnete Zeile in der Firmentabelle nicht löschen, solange in der Employee-Tabelle mindestens eine untergeordnete Zeile vorhanden ist.

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

Meldung 547, Ebene 16, Status 0, Zeile 14 Die DELETE-Anweisung steht in Konflikt mit der REFERENCE-Einschränkung "FK__Employee__Compan__1EE485AA". Der Konflikt ist in der Datenbank "MyDb", Tabelle "dbo.Employee", Spalte "CompanyId" aufgetreten. Die Anweisung wurde beendet.

Durch die Fremdschlüsselbeziehung wird sichergestellt, dass die Zeilen "Unternehmen" und "Mitarbeiter" nicht "getrennt" werden.

Fremdschlüsselbeziehung für vorhandene Tabelle hinzufügen

Die FOREIGN KEY- Einschränkung kann für vorhandene Tabellen hinzugefügt werden, die noch nicht in Beziehung stehen. Stellen Sie sich vor, wir haben Tabellen für Unternehmen und Mitarbeiter, in denen die Spalte CompanyId der Tabelle Mitarbeiter aufgeführt ist, aber keine Fremdschlüsselbeziehung haben. Mit der Anweisung ALTER TABLE können Sie eine **Fremdschlüsseleinschränkung** für eine vorhandene Spalte hinzufügen, die auf andere Tabellen und Primärschlüssel in dieser Tabelle verweist:

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

Fügen Sie einen Fremdschlüssel für eine vorhandene Tabelle hinzu

FOREIGN KEY- Spalten mit Einschränkung können zu vorhandenen Tabellen hinzugefügt werden, die noch nicht in Beziehung stehen. Stellen Sie sich vor, wir haben Firmen- und Mitarbeiter-Tabellen, in denen die Employee-Tabelle keine CompanyId-Spalte hat. Mit der Anweisung ALTER TABLE können Sie eine neue Spalte mit **Fremdschlüsseleinschränkung** hinzufügen, die auf eine andere Tabelle und einen Primärschlüssel in dieser Tabelle verweist:

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

Informationen über Fremdschlüsseleinschränkungen erhalten

Die Systemansicht sys.foreignkeys gibt Informationen zu allen Fremdschlüsselbeziehungen in der Datenbank zurück:

```
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
```

Fremde Schlüssel online lesen: https://riptutorial.com/de/sql-server/topic/5355/fremde-schlussel

Kapitel 48: FÜR JSON

Examples

FÜR JSON PATH

Formatiert die Ergebnisse der SELECT-Abfrage als JSON-Text. Die FOR JSON PATH-Klausel wird nach der Abfrage hinzugefügt:

```
SELECT top 3 object_id, name, type, principal_id FROM sys.objects
FOR JSON PATH
```

Spaltennamen werden in JSON als Schlüssel verwendet, und Zellenwerte werden als JSON-Werte generiert. Ergebnis der Abfrage wäre ein Array von JSON-Objekten:

```
[
  {"object_id":3,"name":"sysrscols","type":"S "},
  {"object_id":5,"name":"sysrowsets","type":"S "},
  {"object_id":6,"name":"sysclones","type":"S "}
]
```

NULL-Werte in der Prinzipal_id-Spalte werden ignoriert (sie werden nicht generiert).

FOR JSON PATH mit Spaltenaliasnamen

Mit FOR JSON PATH können Sie das Format der Ausgabe-JSON mithilfe von Spaltenaliasnamen steuern:

```
SELECT top 3 object_id as id, name as [data.name], type as [data.type]
FROM sys.objects
FOR JSON PATH
```

Spaltenalias wird als Schlüsselname verwendet. Punktgetrennte Spaltenaliase (data.name und data.type) werden als verschachtelte Objekte generiert. Wenn zwei Spalten das gleiche Präfix in Punktnotation haben, werden sie in einem einzelnen Objekt zusammengefasst (Daten in diesem Beispiel):

```
[
  {"id":3,"data":{"name":"sysrscols","type":"S "}},
  {"id":5,"data":{"name":"sysrowsets","type":"S "}},
  {"id":6,"data":{"name":"sysclones","type":"S "}}
]
```

FOR JSON-Klausel ohne Array-Wrapper (einzelnes Objekt in Ausgabe)

Mit der Option WITHOUT_ARRAY_WRAPPER können Sie ein einzelnes Objekt anstelle des Arrays generieren. Verwenden Sie diese Option, wenn Sie wissen, dass Sie eine einzelne Zeile /

ein einzelnes Objekt zurückgeben werden:

```
SELECT top 3 object_id, name, type, principal_id
FROM sys.objects
WHERE object_id = 3
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

Ein einzelnes Objekt wird in diesem Fall zurückgegeben:

```
{"object_id":3,"name":"sysrscols","type":"S "}
```

INCLUDE_NULL_VALUES

Die FOR JSON-Klausel ignoriert NULL-Werte in Zellen. Wenn Sie "Schlüssel": Null-Paare für Zellen erzeugen möchten, die NULL-Werte enthalten, fügen Sie die Option INCLUDE_NULL_VALUES zur Abfrage hinzu:

```
SELECT top 3 object_id, name, type, principal_id
FROM sys.objects
FOR JSON PATH, INCLUDE_NULL_VALUES
```

Es werden NULL-Werte in der principal_id-Spalte generiert:

```
[
  {"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}
]
```

Ergebnisse mit ROOT-Objekt umschließen

Wraps Zurückgegebenes JSON-Array in einem zusätzlichen Stammobjekt mit dem angegebenen Schlüssel:

```
SELECT top 3 object_id, name, type FROM sys.objects
FOR JSON PATH, ROOT('data')
```

Ergebnis der Abfrage wäre ein Array von JSON-Objekten im Wrapper-Objekt:

```
{
  "data":[
            {"object_id":3,"name":"sysrscols","type":"S "},
            {"object_id":5,"name":"sysrowsets","type":"S "},
            {"object_id":6,"name":"sysclones","type":"S "}
          ]
}
```

FÜR JSON AUTO

Verschachtelt Werte aus der zweiten Tabelle automatisch als verschachteltes Unterarray von

JSON-Objekten:

```
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
```

Ergebnis der Abfrage wäre ein Array von JSON-Objekten:

```
[
  {
   "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"}
     ]
  }
]
```

Erstellen einer benutzerdefinierten geschachtelten JSON-Struktur

Wenn Sie eine komplexe JSON-Struktur benötigen, die nicht mit FOR JSON PATH oder FOR JSON AUTO erstellt werden kann, können Sie Ihre JSON-Ausgabe anpassen, indem Sie FOR JSON-Unterabfragen als Spaltenausdrücke verwenden:

```
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
```

Jede Unterabfrage erzeugt ein JSON-Ergebnis, das in den Hauptinhalt von JSON aufgenommen wird.

FÜR JSON online lesen: https://riptutorial.com/de/sql-server/topic/4661/fur-json

Kapitel 49: FÜR XML-PFAD

Bemerkungen

Es gibt auch mehrere andere `FOR XML` Modi:

- `FOR XML RAW` - Erzeugt pro Zeile ein `<row>` -Element.
- `FOR XML AUTO` - Versucht, automatisch eine Hierarchie zu erstellen.
- `FOR XML EXPLICIT` - Bietet mehr Kontrolle über die Form von XML, ist jedoch umständlicher als `FOR XML PATH` .

Examples

Hallo Welt XML

```
SELECT 'Hello World' FOR XML PATH('example')
```

```
<example>Hello World</example>
```

Namespaces angeben

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>
```

Struktur mit XPath-Ausdrücken angeben

```
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>
```

In `FOR XML PATH` werden Spalten ohne Namen zu Textknoten. `NULL` oder `''` daher zu leeren Textknoten. Hinweis: Sie können eine benannte Spalte mithilfe von `AS *` in eine unbenannte Spalte konvertieren.

```
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>
```

Die Verwendung von (leeren) Textknoten hilft, den vorherigen Ausgabeknoten vom nächsten zu trennen, sodass SQL Server weiß, dass ein neues Element für die nächste Spalte gestartet werden muss. Andernfalls wird es verwirrt, wenn das Attribut für das Element "current" bereits vorhanden ist.

Ohne die leeren Zeichenfolgen zwischen dem Element und dem Attribut in der `SELECT` Anweisung gibt SQL Server beispielsweise einen Fehler aus:

> Die Attribut-zentrierte Spalte 'FLD / @ NAME' darf nicht nach einem nicht-Attribut-zentrischen Geschwister in XML-Hierarchie in FOR XML PATH stehen.

Beachten Sie auch, dass dieses Beispiel die XML-Datei auch in ein Wurzelelement namens `row` ,

das durch `ROOT('row')`

Verwenden von FOR XML PATH zum Verketten von Werten

Der `FOR XML PATH` kann zum Verketten von Werten in Zeichenfolgen verwendet werden. Das folgende Beispiel verkettet Werte in einer `CSV` Zeichenfolge:

```
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
    ,''
);
```

Einige wichtige Hinweise:

- Die `ORDER BY` Klausel kann verwendet werden, um die Werte auf eine bevorzugte Weise zu sortieren
- Wenn ein längerer Wert als Verkettungstrennzeichen verwendet wird, muss auch der Funktionsparameter `STUFF` geändert werden.

```
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
    ,''
);
```

- `.value` die Option `TYPE` und `.value` Funktion `.value` verwendet werden, funktioniert die Verkettung mit der `NVARCHAR(MAX)`

FÜR XML-PFAD online lesen: https://riptutorial.com/de/sql-server/topic/727/fur-xml-pfad

Kapitel 50: Geplante Aufgabe oder Auftrag

Einführung

Der SQL Server-Agent verwendet SQL Server zum Speichern von Auftragsinformationen. Jobs enthalten einen oder mehrere Jobschritte. Jeder Schritt enthält eine eigene Aufgabe, dh das Sichern einer Datenbank. Der SQL Server-Agent kann einen Job nach einem Zeitplan, als Reaktion auf ein bestimmtes Ereignis oder auf Anforderung ausführen.

Examples

Erstellen Sie einen geplanten Job

Einen Job erstellen

- Um zuerst einen Job hinzuzufügen, müssen Sie eine gespeicherte Prozedur namens sp_add_job verwenden

```
USE msdb ;
GO
EXEC dbo.sp_add_job
@job_name = N'Weekly Job' ;  -- the job name
```

- Dann müssen wir mit einer gespeicherten Prozedur namens sp_add_jobStep einen Jobschritt hinzufügen

```
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
```

- Richten Sie den Job auf einen Server aus

```
EXEC dbo.sp_add_jobserver
@job_name = N'Weekly Sales Data Backup',
@server_name = 'MyPC\data;   -- Default is LOCAL
GO
```

Erstellen Sie einen Zeitplan mit SQL

Um einen Zeitplan zu erstellen, müssen wir eine gespeicherte Systemprozedur mit dem Namen sp_add_schedule verwenden

```
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
```

Es gibt weitere Parameter, die mit `sp_add_schedule` können. Weitere `sp_add_schedule` Sie in dem oben angegebenen Link.

Zeitplan an einen Job anhängen

Um einen Zeitplan an einen SQL-Agent-Auftrag anzuhängen, müssen Sie eine gespeicherte Prozedur namens sp_attach_schedule verwenden

```
-- 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
```

Geplante Aufgabe oder Auftrag online lesen: https://riptutorial.com/de/sql-server/topic/5329/geplante-aufgabe-oder-auftrag

Kapitel 51: Gespeicherte Prozeduren

Einführung

In SQL Server ist eine Prozedur ein gespeichertes Programm, an das Sie Parameter übergeben können. Es gibt keinen Wert wie eine Funktion zurück. Es kann jedoch einen Erfolgs- / Fehlerstatus an die Prozedur zurückgeben, die es aufgerufen hat.

Syntax

- ERSTELLEN {PROCEDURE | PROC} [Schemaname.] Prozedurname
- [@parameter [Typ_Schema_Name.] Datentyp
- [VARYING] [= Standard] [OUT | AUSGABE | SCHREIBGESCHÜTZT]
- , @parameter [Typ_Schema_Name.] Datentyp
- [VARYING] [= Standard] [OUT | AUSGABE | SCHREIBGESCHÜTZT]]
- [MIT {ENCRYPTION | RECOMPILE | EXECUTE AS-Klausel}]
- [ZUR ANTWORTUNG]
- WIE
- START
- [Deklarationsabschnitt]
- executable_section
- ENDE;

Examples

Erstellen und Ausführen einer einfachen gespeicherten Prozedur

Verwenden der `Authors` Tabelle in der Bibliotheksdatenbank

```
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
```

Sie können eine Prozedur mit einigen unterschiedlichen Syntaxen ausführen. Erstens können Sie `EXECUTE` oder `EXEC`

```
EXECUTE GetName @id = 1
EXEC Getname @name = 'Ernest Hemingway'
```

Darüber hinaus können Sie den Befehl EXEC weglassen. Sie müssen auch nicht angeben, welchen Parameter Sie übergeben, da Sie alle Parameter übergeben.

```
GetName NULL, 'Ernest Hemingway'
```

Wenn Sie die Eingabeparameter in einer anderen Reihenfolge angeben möchten als in der Prozedur deklariert, können Sie den Parameternamen angeben und Werte zuweisen. Zum Beispiel

```
 CREATE PROCEDURE dbo.sProcTemp
 (
    @Param1 INT,
    @Param2 INT
)
AS
BEGIN

    SELECT
        Param1 = @Param1,
        Param2 = @Param2

END
```

Die normale Reihenfolge zum Ausführen dieser Prozedur besteht darin, zuerst den Wert für @ Param1 und dann für @ Param2 anzugeben. So wird es ungefähr so aussehen

```
  EXEC dbo.sProcTemp @Param1 = 0,@Param2=1
```

Es ist jedoch auch möglich, dass Sie Folgendes verwenden können

```
  EXEC dbo.sProcTemp @Param2 = 0,@Param1=1
```

Dabei legen Sie zuerst den Wert für @ param2 und @ Param1 Sekunde fest. Das bedeutet, dass Sie nicht die gleiche Reihenfolge einhalten müssen, wie sie in der Prozedur angegeben ist. Sie können jedoch eine beliebige Reihenfolge haben, wie Sie möchten. Sie müssen jedoch angeben, für welchen Parameter Sie den Wert einstellen

Greifen Sie auf eine gespeicherte Prozedur aus einer beliebigen Datenbank zu

Außerdem können Sie eine Prozedur mit einem Präfix `sp_` Diese Procuedres können wie alle gespeicherten Systemprozeduren aufgrund des Standardverhaltens von SQL Server ohne Angabe der Datenbank ausgeführt werden. Wenn Sie eine gespeicherte Prozedur ausführen, die mit "sp_" beginnt, sucht SQL Server zuerst nach der Prozedur in der Master-Datenbank. Wenn die Prozedur nicht im Master gefunden wird, wird in der aktiven Datenbank gesucht. Wenn Sie über eine gespeicherte Prozedur verfügen, auf die Sie aus allen Datenbanken zugreifen möchten, erstellen Sie sie in master und verwenden Sie einen Namen, der das Präfix "sp_" enthält.

```
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
```

GESPEICHERTES VERFAHREN mit OUT-Parametern

Gespeicherte Prozeduren können Werte mit dem Schlüsselwort `OUTPUT` in ihrer Parameterliste zurückgeben.

Erstellen einer gespeicherten Prozedur mit einem einzelnen Out-Parameter

```
CREATE PROCEDURE SprocWithOutParams
(
    @InParam VARCHAR(30),
    @OutParam VARCHAR(30) OUTPUT
)
AS
BEGIN
    SELECT @OutParam = @InParam + ' must come out'
    RETURN
END
GO
```

Ausführen der gespeicherten Prozedur

```
DECLARE @OutParam VARCHAR(30)
EXECUTE SprocWithOutParams 'what goes in', @OutParam OUTPUT
PRINT @OutParam
```

Erstellen einer gespeicherten Prozedur mit mehreren Out-Parametern

```
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
```

Ausführen der gespeicherten Prozedur

```
DECLARE @OutParam VARCHAR(30)
DECLARE @OutParam2 VARCHAR(30)
EXECUTE SprocWithOutParams2 'what goes in', @OutParam OUTPUT, @OutParam2 OUTPUT
PRINT @OutParam
PRINT @OutParam2
```

Gespeicherte Prozedur mit If ... Else und Insert Ino Operation

Erstellen Sie eine Beispieltabelle `Employee` :

```
CREATE TABLE Employee
(
    Id INT,
    EmpName VARCHAR(25),
    EmpGender VARCHAR(6),
    EmpDeptId INT
)
```

Erstellt eine gespeicherte Prozedur, die prüft, ob die in der gespeicherten Prozedur übergebenen Werte nicht null oder nicht leer sind, und führt eine Einfügeoperation in der Employee-Tabelle aus.

```
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
```

Führen Sie die gespeicherte Prozedur aus

```
DECLARE @ID INT,
    @Name VARCHAR(25),
    @Gender VARCHAR(6),
    @DeptId INT

EXECUTE spSetEmployeeDetails
    @ID = 1,
    @Name = 'Subin Nepal',
    @Gender = 'Male',
    @DeptId = 182666
```

Dynamisches SQL in gespeicherter Prozedur

Mit Dynamic SQL können wir zur Laufzeit SQL-Anweisungen generieren und ausführen. Dynamisches SQL ist erforderlich, wenn unsere SQL-Anweisungen einen Bezeichner enthalten, der sich zu verschiedenen Kompilierungszeiten ändern kann.

Einfaches Beispiel für dynamisches SQL:

```
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
```

In der obigen SQL-Abfrage können wir sehen, dass wir die obige Abfrage verwenden `@table_name, @col_name, and @col_value` zur Laufzeit Werte in `@table_name, @col_name, and @col_value` . Die Abfrage wird zur Laufzeit generiert und ausgeführt. Dies ist eine Technik, bei der wir ganze Skripte als String in einer Variablen erstellen und ausführen können. Wir können komplexere Abfragen mit dynamischen SQL- und Verkettungskonzepten erstellen. Dieses Konzept ist sehr nützlich, wenn Sie ein Skript erstellen möchten, das unter verschiedenen Bedingungen verwendet werden kann.

Gespeicherte Prozedur ausführen

```
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
```

Tabelle, die ich verwendet habe

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

Ausgabe

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

Einfaches Looping

`#systables` uns zunächst einige Daten in eine temporäre Tabelle mit dem Namen `#systables` und eine `#systables` sodass wir jeweils einen Datensatz abfragen können

```
select
    o.name,
    row_number() over (order by o.name) as rn
into
    #systables
from
    sys.objects as o
where
    o.type = 'S'
```

Als Nächstes deklarieren wir einige Variablen zur Steuerung der Schleife und speichern den Tabellennamen in diesem Beispiel

```
declare
    @rn int = 1,
    @maxRn int = (
```

```
                    select
                        max(rn)
                    from
                        #systables as s
                    )
declare    @tablename sys name
```

Jetzt können wir eine einfache Schleife verwenden. Wir `@rn` in der `select` Anweisung, dies könnte jedoch auch eine separate Anweisung für ex `set @rn = @rn + 1` sein, je nach Ihren Anforderungen. Wir verwenden auch den Wert von `@rn` bevor er erhöht wird, um einen einzelnen Datensatz aus `#systables` . Zuletzt drucken wir den Tabellennamen.

```
while @rn <= @maxRn
    begin

        select
            @tablename = name,
            @rn = @rn + 1
        from
            #systables as s
        where
            s.rn = @rn

        print @tablename
    end
```

Einfaches Looping

```
CREATE PROCEDURE SprocWithSimpleLoop
(
    @SayThis VARCHAR(30),
    @ThisManyTimes INT
)
AS
BEGIN
    WHILE @ThisManyTimes > 0
    BEGIN
        PRINT @SayThis;
        SET @ThisManyTimes = @ThisManyTimes - 1;
    END

    RETURN;
END
GO
```

Gespeicherte Prozeduren online lesen: https://riptutorial.com/de/sql-server/topic/3213/gespeicherte-prozeduren

Kapitel 52: Grundlegende DDL-Vorgänge in MS SQL Server

Examples

Fertig machen

In diesem Abschnitt werden einige grundlegende **DDL-** Befehle (= " **D** ata **D** efinition **L** anguage") zum Erstellen einer Datenbank, einer Tabelle innerhalb einer Datenbank, einer Ansicht und schließlich einer gespeicherten Prozedur beschrieben.

Datenbank erstellen

Mit dem folgenden SQL-Befehl wird eine neue Datenbank `Northwind` auf dem aktuellen Server erstellt. Verwenden Sie hierzu den Pfad `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
```

Hinweis: Eine T-SQL-Datenbank besteht aus zwei Dateien, der Datenbankdatei `*.mdf` und dem Transaktionsprotokoll `*.ldf` . Beide müssen angegeben werden, wenn eine neue Datenbank erstellt wird.

Tabelle erstellen

Der folgende SQL - Befehl erstellt eine neue Tabelle `Categories` in der aktuellen Datenbank - Schema mit `dbo` (Sie Datenbankkontext mit umschalten `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
```

Ansicht erstellen

Der folgende SQL-Befehl erstellt eine neue Sicht `Summary_of_Sales_by_Year` in der aktuellen Datenbank mit dem Schema `dbo` (Sie können den Datenbankkontext mit `Use <DatabaseName>` wechseln):

```
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
```

Dadurch werden die Tabellen `Orders` und `[Order Subtotals]` , um die Spalten `ShippedDate` , `OrderID` und `Subtotal OrderID` . Da die Tabelle `[Order Subtotals]` in der Northwind-Datenbank ein Leerzeichen enthält, muss sie in eckigen Klammern stehen.

Prozedur erstellen

Der folgende SQL-Befehl erstellt eine neue gespeicherte Prozedur `CustOrdersDetail` in der aktuellen Datenbank mit dem Schema `dbo` (Sie können den Datenbankkontext mit `Use <DatabaseName>` wechseln):

```
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
```

Diese gespeicherte Prozedur kann nach ihrer Erstellung folgendermaßen aufgerufen werden:

```
exec dbo.MyCustOrdersDetail 10248
```

Dabei werden alle Bestelldetails mit @ OrderId = 10248 (und menge> = 0 als Standardwert) zurückgegeben. Oder Sie können den optionalen Parameter angeben

```
exec dbo.MyCustOrdersDetail 10248, 10
```

Dadurch werden nur Bestellungen mit einer Mindestmenge von 10 (oder mehr) zurückgegeben.

Grundlegende DDL-Vorgänge in MS SQL Server online lesen: https://riptutorial.com/de/sql-server/topic/5463/grundlegende-ddl-vorgange-in-ms-sql-server

Kapitel 53: GRUPPIERE NACH

Examples

Einfache Gruppierung

Bestellungstabelle

Kundennummer	Produkt ID	Menge	Preis
1	2	5	100
1	3	2	200
1	4	1	500
2	1	4	50
3	5	6	700

Bei der Gruppierung nach einer bestimmten Spalte werden nur eindeutige Werte dieser Spalte zurückgegeben.

```
SELECT customerId
FROM orders
GROUP BY customerId;
```

Rückgabewert:

Kundennummer
1
2
3

Aggregatfunktionen wie `count()` gelten für jede Gruppe und nicht für die vollständige Tabelle:

```
SELECT customerId,
       COUNT(productId) as numberOfProducts,
       sum(price) as totalPrice
FROM orders
GROUP BY customerId;
```

Rückgabewert:

Kundennummer	Anzahl der Produkte	totalPreis
1	3	800
2	1	50
3	1	700

GROUP BY mehrere Spalten

Möglicherweise möchten Sie GROUP BY mehr als eine Spalte

```
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
```

wird nach Alter und Namen gruppiert und produziert:

Alter	Name	Anzahl
18	luke	3
21	luke	2
18	matt	1
21	matt	2

Gruppieren mit mehreren Tabellen, mehreren Spalten

Group by wird oft mit der join-Anweisung verwendet. Nehmen wir an, wir haben zwei Tabellen. Der erste ist der Schultisch:

Ich würde	Vollständiger Name	Alter
1	Matt Jones	20
2	Frank Blue	21

Ich würde	Vollständiger Name	Alter
3	Anthony Angel	18

Die zweite Tabelle ist die Tabelle, die jeder Student belegen kann:

Subject_Id	Gegenstand
1	Mathe
2	PE
3	Physik

Und da ein Schüler viele Fächer besuchen kann und ein Fach von vielen Schülern besucht werden kann (daher N: N-Beziehung), müssen wir einen dritten "Begrenzungstisch" haben. Nennen wir die Tabelle Students_subjects:

Subject_Id	Studenten ID
1	1
2	2
2	1
3	2
1	3
1	1

Nehmen wir an, wir möchten wissen, wie viele Fächer jeder Schüler besucht. Hier ist die eigenständige `GROUP BY` Anweisung nicht ausreichend, da die Informationen nicht über eine einzige Tabelle verfügbar sind. Daher müssen wir `GROUP BY` mit der `JOIN` Anweisung verwenden:

```
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
```

Das Ergebnis der angegebenen Abfrage lautet wie folgt:

Vollständiger Name	Betreffnummer
Matt Jones	3
Frank Blue	2

Vollständiger Name	Betreffnummer
Anthony Angel	1

Für ein noch komplexeres Beispiel für die Verwendung von GROUP BY kann der Student seinem Namen möglicherweise mehr als einmal das gleiche Thema zuweisen (wie in der Tabelle Students_Subjects gezeigt). In diesem Szenario können wir möglicherweise zählen, wie oft jedes Subjekt einem Schüler von GROUPing durch mehr als eine Spalte zugewiesen wurde:

```
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
```

Diese Abfrage liefert das folgende Ergebnis:

Vollständiger Name	Gegenstand	Betreffnummer
Matt Jones	Mathe	2
Matt Jones	PE	1
Frank Blue	PE	1
Frank Blue	Physik	1
Anthony Angel	Mathe	1

HABEN

Da die `WHERE` Klausel vor `GROUP BY` ausgewertet wird, können Sie `WHERE` nicht verwenden, um die Ergebnisse der Gruppierung (in der Regel eine Aggregatfunktion, wie z. B. `COUNT(*)`), zu `COUNT(*)` . Um diesem Bedarf `HAVING` , kann die `HAVING` Klausel verwendet werden.

Verwenden Sie beispielsweise folgende Daten:

```
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' )
```

Wenn wir die Anzahl der Bestellungen erhalten möchten, die jede Person aufgegeben hat, würden wir sie verwenden

```
SELECT Name, COUNT(*) AS 'Orders'
FROM @orders
GROUP BY Name
```

und bekomme

Name	Aufträge
Matt	2
John	5
Luke	4

Wenn wir dies jedoch auf Personen beschränken möchten, die mehr als zwei Bestellungen `HAVING` haben, können wir eine `HAVING` Klausel hinzufügen.

```
SELECT Name, COUNT(*) AS 'Orders'
FROM @orders
GROUP BY Name
HAVING COUNT(*) > 2
```

wird nachgeben

Name	Aufträge
John	5
Luke	4

Beachten Sie, dass die Spalten, die in `HAVING` werden, genau wie `GROUP BY` genau ihren Entsprechungen in der `SELECT` Anweisung entsprechen müssen. Wenn wir im obigen Beispiel stattdessen gesagt hätten

```
SELECT Name, COUNT(DISTINCT OrderID)
```

Unsere `HAVING` Klausel müsste sagen

```
HAVING COUNT(DISTINCT OrderID) > 2
```

GROUP BY mit ROLLUP und CUBE

Der ROLLUP-Operator ist nützlich, um Berichte zu generieren, die Zwischensummen und Summen enthalten.

- CUBE generiert eine Ergebnismenge, in der Aggregate für alle Wertekombinationen in den ausgewählten Spalten angezeigt werden.
- ROLLUP generiert eine Ergebnismenge, die Aggregate für eine Hierarchie von Werten in den ausgewählten Spalten anzeigt.

Artikel	Farbe	Menge
Tabelle	Blau	124
Tabelle	rot	223
Stuhl	Blau	101
Stuhl	rot	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 Zeile (n) betroffen)

Wenn das ROLLUP-Schlüsselwort in der Abfrage in CUBE geändert wird, ist die CUBE-Ergebnismenge gleich, mit Ausnahme dieser beiden zusätzlichen Zeilen werden am Ende zurückgegeben:

```
ALL                  Blue                 225.00
ALL                  Red                  433.00
```

https://technet.microsoft.com/de-de/library/ms189305(v=sql.90).aspx

GRUPPIERE NACH online lesen: https://riptutorial.com/de/sql-server/topic/3231/gruppiere-nach

Kapitel 54: Index

Examples

Clustered-Index erstellen

Bei einem gruppierten Index enthalten die Blattseiten die tatsächlichen Tabellenzeilen. Daher kann es nur einen Clusterindex geben.

```
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
```

Erstellen Sie einen nicht gruppierten Index

Nicht gruppierte Indizes haben eine von den Datenzeilen getrennte Struktur. Ein nicht gruppierter Index enthält die nicht gruppierten Indexschlüsselwerte, und jeder Schlüsselwerteintrag enthält einen Zeiger auf die Datenzeile, die den Schlüsselwert enthält. Es kann maximal 999 nicht gruppierten Index für SQL Server 2008/2012 geben.

Link als Referenz: 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
```

Indexinfo anzeigen

```
SP_HELPINDEX tableName
```

Index wird angezeigt

```
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 löschen

```
DROP INDEX IX_NonClustered ON Employees
```

Gibt Größen- und Fragmentierungsindizes zurück

```
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');
```

Index neu anordnen und neu erstellen

avg_fragmentation_in_percent value	Korrekturerklärung
> 5% und <= 30%	REORGANISIEREN
> 30%	REBUILD

```
ALTER INDEX IX_NonClustered ON tableName REORGANIZE;


ALTER INDEX ALL ON Production.Product
 REBUILD WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON,
          STATISTICS_NORECOMPUTE = ON);
```

Erstellen Sie alle Indizes einer Tabelle neu oder ordnen Sie sie neu

Die Neuerstellung von Indizes erfolgt mit der folgenden Anweisung

```
ALTER INDEX All ON tableName REBUILD;
```

Dies löscht den Index und erstellt ihn neu, entfernt Fragementation, stellt Speicherplatz frei und ordnet Indexseiten neu an.

Man kann einen Index auch mit reorganisieren

```
ALTER INDEX All ON tableName REORGANIZE;
```

Dabei werden nur minimale Systemressourcen benötigt und die Blattebene gruppierter und nicht gruppierter Indizes für Tabellen und Ansichten defragmentiert, indem die Seiten der Blattebene physisch neu angeordnet werden, um der logischen Reihenfolge der Blattknoten von links nach rechts zu entsprechen

Erstellen Sie alle Indexdatenbank neu

```
EXEC sp_MSForEachTable 'ALTER INDEX ALL ON ? REBUILD'
```

Indexuntersuchungen

Sie können "SP_HELPINDEX Table_Name" verwenden, aber Kimberly Tripp verfügt über eine gespeicherte Prozedur (die Sie hier finden können). Dies ist ein besseres Beispiel, da mehr Informationen zu den Indizes einschließlich Spalten und Filterdefinitionen angezeigt werden. Beispiel:
Verwendungszweck:

```
USE Adventureworks
EXEC sp_SQLskills_SQL2012_helpindex 'dbo.Product'
```

Alternativ verfügt Tibor Karaszi über eine gespeicherte Prozedur (hier zu finden). In diesem werden auch Informationen zur Indexnutzung angezeigt und optional eine Liste mit Indexvorschlägen bereitgestellt. Verwendungszweck:

```
USE Adventureworks
EXEC sp_indexinfo 'dbo.Product'
```

Index online lesen: https://riptutorial.com/de/sql-server/topic/4998/index

Kapitel 55: In-Memory-OLTP (Hekaton)

Examples

Speicheroptimierte Tabelle erstellen

```
-- 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
```

Erstellte DLL-Dateien und Tabellen für speicheroptimierte Tabellen anzeigen

```
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
```

Alle speicheroptimierten Tabellen anzeigen:

```
SELECT
    name,type_desc,durability_desc,Is_memory_Optimized
FROM sys.tables
    WHERE Is_memory_Optimized = 1
GO
```

Speicheroptimierte Tabellentypen und Temp-Tabellen

Dies ist zum Beispiel der traditionelle Tempdb-basierte Tabellentyp:

```
CREATE TYPE dbo.testTableType AS TABLE
(
   col1 INT NOT NULL,
   col2 CHAR(10)
);
```

Zur Speicheroptimierung dieses Tabellentyps fügen Sie einfach die Option `memory_optimized=on` und fügen Sie einen Index hinzu, wenn der ursprüngliche Typ keinen enthält:

```
CREATE TYPE dbo.testTableType AS TABLE
(
   col1 INT NOT NULL,
   col2 CHAR(10)
```

```
)WITH (MEMORY_OPTIMIZED=ON);
```

Globale temporäre Tabelle ist wie folgt:

```
CREATE TABLE ##tempGlobalTabel
(
    Col1   INT   NOT NULL ,
    Col2   NVARCHAR(4000)
);
```

Speicheroptimierte globale temporäre Tabelle:

```
CREATE TABLE dbo.tempGlobalTabel
(
    Col1   INT   NOT NULL   INDEX ix NONCLUSTERED,
    Col2   NVARCHAR(4000)
)
    WITH
        (MEMORY_OPTIMIZED = ON,
         DURABILITY = SCHEMA_ONLY);
```

So optimieren Sie globale Temp-Tabellen (## temp) im Speicher:

1. Erstellen Sie eine neue `SCHEMA_ONLY` speicheroptimiert Tabelle mit demselben Schema wie die globale `##temp` - Tabelle
    - Stellen Sie sicher, dass die neue Tabelle mindestens einen Index hat
2. Ändern Sie alle Verweise auf `##temp` in Ihren Transact-SQL-Anweisungen in die neue speicheroptimierte Tabellentemperatur
3. Ersetzen Sie die Anweisungen `DROP TABLE ##temp` in Ihrem Code durch `DELETE FROM temp` , um den Inhalt zu bereinigen
4. Entfernen Sie die `CREATE TABLE ##temp` Anweisungen aus Ihrem Code - diese sind jetzt überflüssig

Mehr Informationen

Deklarieren Sie speicheroptimierte Tabellenvariablen

Für eine schnellere Leistung können Sie Ihre Tabellenvariable speicheroptimieren. Hier ist das T-SQL für eine traditionelle Tabellenvariable:

```
DECLARE @tvp TABLE
(
    col1   INT NOT NULL ,
    Col2   CHAR(10)
);
```

Um speicheroptimierte Variablen zu definieren, müssen Sie zuerst einen speicheroptimierten Tabellentyp erstellen und dann eine Variable daraus deklarieren:

```
CREATE TYPE dbo.memTypeTable
AS TABLE
```

```
(
    Col1  INT NOT NULL INDEX ix1,
    Col2  CHAR(10)
)
WITH
    (MEMORY_OPTIMIZED = ON);
```

Dann können wir den Tabellentyp folgendermaßen verwenden:

```
DECLARE @tvp memTypeTable
insert INTO @tvp
values (1,'1'),(2,'2'),(3,'3'),(4,'4'),(5,'5'),(6,'6')

SELECT * FROM @tvp
```

Ergebnis:

```
Col1    Col2
1       1
2       2
3       3
4       4
5       5
6       6
```

Speicheroptimierte, systemversionierte temporale Tabelle erstellen

```
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 )
)
```

Mehr Informationen

In-Memory-OLTP (Hekaton) online lesen: https://riptutorial.com/de/sql-server/topic/5295/in-memory-oltp--hekaton-

Kapitel 56: Isolationsstufen und Verriegelung

Bemerkungen

Ich habe diesen Link gefunden - er ist nützlich als Referenz: "Isolation Levels"

Examples

Beispiele für die Einstellung der Isolationsstufe

Beispiel für die Einstellung der Isolationsstufe:

```
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` - bedeutet, dass eine Abfrage in der aktuellen Transaktion nicht auf die geänderten Daten einer anderen Transaktion zugreifen kann, die noch nicht festgeschrieben ist - keine fehlerhaften Lesevorgänge! ABER nicht wiederholbare Lesevorgänge und Phantomlesevorgänge sind möglich, da Daten noch durch andere Transaktionen geändert werden können.

2. `REPEATABLE READ` - bedeutet, dass eine Abfrage in der aktuellen Transaktion nicht auf die geänderten Daten einer anderen Transaktion zugreifen kann, die noch nicht festgeschrieben ist - keine fehlerhaften Lesevorgänge! Keine anderen Transaktionen können die von der aktuellen Transaktion gelesenen Daten bis zu ihrem Abschluss modifizieren, wodurch NONREPEATABLE-Lesevorgänge ausgeschlossen werden. ABER, wenn eine andere Transaktion NEW ROWS einfügt und die Abfrage mehrmals ausgeführt wird, können Phantomzeilen beim zweiten Lesevorgang erscheinen (wenn sie mit der Where-Anweisung der Abfrage übereinstimmen).

3. `SNAPSHOT` - kann nur Daten zurückgeben, die zu Beginn der Abfrage vorhanden sind. Gewährleistet die Konsistenz der Daten. Es verhindert schmutzige Lesevorgänge, nicht wiederholbare Lesevorgänge und Phantomlesevorgänge. Um dies zu verwenden, ist eine DB-Konfiguration erforderlich:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

4. `READ COMMITTED` - Standardisolation des SQL-Servers. Es verhindert, dass Daten gelesen werden, die von einer anderen Transaktion geändert wurden, bis sie festgeschrieben sind. Es verwendet gemeinsames Sperren und Zeilenversionieren für die Tabellen, wodurch Dirty Reads verhindert werden. Dies hängt von der DB-Konfiguration ab. READ_COMMITTED_SNAPSHOT - falls aktiviert - wird die Zeilenversionierung verwendet. aktivieren - verwenden Sie dies:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED; --return to the default one
```

5. `SERIALIZABLE` - Verwendet physische Sperren, die bis zum Ende der Transaktion `SERIALIZABLE` und gehalten werden. Dadurch werden Dirty Reads, Phantom Reads und nicht wiederholbare Reads verhindert. ABER, es wirkt sich auf die Leistung der Datenbank aus, da die gleichzeitigen Transaktionen serialisiert werden und nacheinander ausgeführt werden.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
```

Isolationsstufen und Verriegelung online lesen: https://riptutorial.com/de/sql-server/topic/5331/isolationsstufen-und-verriegelung

Kapitel 57: JSON im SQL Server

Syntax

- **JSON_VALUE** (Ausdruck, Pfad) - Extrahieren Sie einen Skalarwert aus einem JSON-String.
- **JSON_QUERY** (Ausdruck [, Pfad]) - **Extrahiert** ein Objekt oder ein Array aus einem JSON-String.
- **OPENJSON** (jsonExpression [, path]) - Tabellenwertfunktion, die JSON-Text analysiert und Objekte und Eigenschaften in JSON als Zeilen und Spalten zurückgibt.
- **ISJSON** (Ausdruck) - **Testet,** ob eine Zeichenfolge gültige JSON enthält.
- **JSON_MODIFY** (Ausdruck, Pfad, NeuerWert) - Aktualisiert den Wert einer Eigenschaft in einem JSON-String und gibt den aktualisierten JSON-String zurück.

Parameter

Parameter	Einzelheiten
Ausdruck	In der Regel der Name einer Variablen oder einer Spalte, die JSON-Text enthält.
Pfad	Ein JSON-Pfadausdruck, der die zu aktualisierende Eigenschaft angibt. path hat die folgende Syntax: [anfügen] [lax | strict] $. <Json-Pfad>
JsonExpression	Ein Unicode-Zeichenausdruck, der den JSON-Text enthält.

Bemerkungen

Die OPENJSON-Funktion ist nur unter Kompatibilitätsstufe 130 verfügbar. Wenn Ihre Datenbankkompatibilitätsstufe niedriger als 130 ist, kann der SQL Server die OPENJSON-Funktion nicht finden und ausführen. Derzeit sind alle Azure SQL-Datenbanken standardmäßig auf 120 festgelegt. Sie können den Kompatibilitätsgrad einer Datenbank mit dem folgenden Befehl ändern:

```
ALTER DATABASE <Database-Name-Here> SET COMPATIBILITY_LEVEL = 130
```

Examples

Formatieren Sie die Abfrageergebnisse als JSON mit FOR JSON

Eingabetabellendaten (Personentabelle)

Ich würde	Name	Alter
1	John	23
2	Jane	31

Abfrage

```
SELECT Id, Name, Age
FROM People
FOR JSON PATH
```

Ergebnis

```
[
    {"Id":1,"Name":"John","Age":23},
    {"Id":2,"Name":"Jane","Age":31}
]
```

Analysieren Sie den JSON-Text

JSON_VALUE- und **JSON_QUERY-** Funktionen analysieren JSON-Text und geben skalare Werte oder Objekte / Arrays im Pfad in JSON-Text zurück.

```
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
```

Ergebnis

Ich würde	Name	UserObject	Kompetenzen	Fähigkeit0
1	John	{"name": "John"}	["C #", "SQL"]	C #

Verbinden Sie übergeordnete und untergeordnete JSON-Entitäten mit CROSS APPLY OPENJSON

Verbinden Sie übergeordnete Objekte mit ihren untergeordneten Entitäten. Zum Beispiel möchten wir eine relationale Tabelle für jede Person und ihre Hobbys

```
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"}
        ]
    }
 ]'
```

Abfrage

```
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
```

Alternativ kann diese Abfrage mit der WITH-Klausel geschrieben werden.

```
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'
)
```

Ergebnis

Ich würde	Name der Person	Hobby
1	John	lesen
1	John	Surfen
2	Jane	Programmierung
2	Jane	Laufen

Indexieren Sie die JSON-Eigenschaften mithilfe von berechneten Spalten

Beim Speichern von JSON-Dokumenten in SQL Server müssen wir die Abfrageergebnisse nach

Eigenschaften der JSON-Dokumente effizient filtern und sortieren können.

```
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
```

In Anbetracht der obigen Tabelle Wenn wir die Zeile mit dem Namen = 'Adam' finden möchten, führen wir die folgende Abfrage aus.

```
SELECT *
FROM JsonTable Where
JSON_VALUE(jsonInfo, '$.Name') = 'Adam'
```

Dies erfordert jedoch, dass der SQL Server eine vollständige Tabelle ausführt, die bei einer großen Tabelle nicht effizient ist.

Um dies zu beschleunigen, möchten wir einen Index hinzufügen. Wir können jedoch nicht direkt auf Eigenschaften im JSON-Dokument verweisen. Die Lösung besteht darin, eine berechnete Spalte im JSON-Pfad `$.Name` hinzuzufügen und dann einen Index für die berechnete Spalte hinzuzufügen.

```
ALTER TABLE JsonTable
ADD vName as JSON_VALUE(jsonInfo, '$.Name')

CREATE INDEX idx_name
ON JsonTable(vName)
```

Wenn wir nun dieselbe Abfrage ausführen, verwendet der SQL Server anstelle einer vollständigen Tabellensuche einen Index, um den nicht gruppierten Index zu durchsuchen und die Zeilen zu finden, die die angegebenen Bedingungen erfüllen.

Anmerkung: Damit der SQL-Server den Index verwenden kann, müssen Sie die berechnete Spalte mit demselben Ausdruck erstellen, den Sie in Ihren Abfragen verwenden `JSON_VALUE(jsonInfo, '$.Name')` - in diesem Beispiel `JSON_VALUE(jsonInfo, '$.Name')` . Sie können jedoch auch den Namen verwenden der berechneten Spalte `vName`

Formatieren Sie eine Tabellenzeile mit FOR JSON als einzelnes JSON-Objekt

Die Option **WITHOUT_ARRAY_WRAPPER** in der *FOR JSON*- Klausel entfernt Array-Klammern aus der JSON-Ausgabe. Dies ist nützlich, wenn Sie eine einzelne Zeile in der Abfrage

zurückgeben.

> Hinweis: Diese Option erzeugt eine ungültige JSON-Ausgabe, wenn mehr als eine Zeile zurückgegeben wird.

Eingabetabellendaten (Personentabelle)

Ich würde	Name	Alter
1	John	23
2	Jane	31

Abfrage

```
SELECT Id, Name, Age
FROM People
WHERE Id = 1
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

Ergebnis

```
{"Id":1,"Name":"John","Age":23}
```

Analysieren Sie JSON-Text mit der Funktion OPENJSON

Die Funktion **OPENJSON** analysiert den JSON-Text und gibt mehrere Ausgaben zurück. Werte, die zurückgegeben werden sollen, werden mithilfe der in der WITH-Klausel definierten Pfade angegeben. Wenn für eine Spalte kein Pfad angegeben ist, wird der Spaltenname als Pfad verwendet. Diese Funktion wandelt zurückgegebene Werte in die in der WITH-Klausel definierten SQL-Typen um. Die AS JSON-Option muss in der Spaltendefinition angegeben werden, wenn ein Objekt / Array zurückgegeben werden soll.

```
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]')
```

Ergebnis

Ich würde	Name	UserObject	Kompetenzen	Fähigkeit0
1	John	{"name": "John"}	["C #", "SQL"]	C #

JSON im SQL Server online lesen: https://riptutorial.com/de/sql-server/topic/2568/json-im-sql-server

Kapitel 58: Kreuz anwenden

Examples

Verbinden Sie Tabellenzeilen mit dynamisch generierten Zeilen aus einer Zelle

Mit CROSS APPLY können Sie Zeilen aus einer Tabelle "verbinden", wobei dynamisch generierte Zeilen von einer Tabellenwertfunktion zurückgegeben werden.

Stellen Sie sich vor, Sie haben eine Firmentabelle mit einer Spalte, die ein Array von Produkten (Spalte ProductList) enthält, und eine Funktion, die diese Werte analysiert und einen Satz von Produkten zurückgibt. Sie können alle Zeilen aus einer Company-Tabelle auswählen, diese Funktion auf eine ProductList-Spalte anwenden und die erzeugten Ergebnisse mit der übergeordneten Company-Zeile "verbinden":

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

Für jede Zeile wird der Funktion ein Wert der *ProductList*- Zelle bereitgestellt, und die Funktion gibt diese Produkte als eine Reihe von Zeilen zurück, die mit der übergeordneten Zeile verbunden werden können.

Verbinden Sie Tabellenzeilen mit einem in Zelle gespeicherten JSON-Array

Mit CROSS APPLY können Sie Zeilen aus einer Tabelle mit einer Sammlung von JSON-Objekten verbinden, die in einer Spalte gespeichert sind.

Stellen Sie sich vor, Sie haben eine Firmentabelle mit einer Spalte, die ein Array von Produkten (Spalte ProductList) enthält, die als JSON-Array formatiert sind. Die Tabellenwertfunktion OPENJSON kann diese Werte analysieren und die Menge der Produkte zurückgeben. Sie können alle Zeilen aus einer Company-Tabelle auswählen, JSON-Produkte mit OPENJSON analysieren und generierte Ergebnisse mit der übergeordneten Company-Zeile "verknüpfen":

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

Für jede Zeile wird der Wert der *ProductList*- Zelle an die OPENJSON-Funktion übergeben, die JSON-Objekte in Zeilen mit dem in WITH-Klausel definierten Schema umwandelt.

Zeilen nach Arraywerten filtern

Wenn Sie eine Liste von Tags in einer Zeile als durch *Komma* getrennte Werte *speichern* , können Sie mit der Funktion *STRING_SPLIT* die Liste der Tags in eine Wertetabelle umwandeln. **Mit**

CROSS APPLY können Sie von der *STRING_SPLIT*- Funktion analysierte Werte mit einer übergeordneten Zeile " *verknüpfen* ".

Stellen Sie sich vor, Sie haben eine Produkttabelle mit einer Spalte, die ein Array mit durch Kommas getrennten Tags enthält (z. B. Promo, Vertrieb, neu). Mit STRING_SPLIT und CROSS APPLY können Sie Produktzeilen mit ihren Tags verknüpfen, sodass Sie Produkte nach Tags filtern können:

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

Für jede Zeile wird der Wert der Zelle *Tags* an die Funktion STRING_SPLIT übergeben, die die Tag-Werte zurückgibt. Dann können Sie Zeilen nach diesen Werten filtern.

Hinweis: Die *STRING_SPLIT*- Funktion ist erst ab **SQL Server 2016** verfügbar

Kreuz anwenden online lesen: https://riptutorial.com/de/sql-server/topic/5462/kreuz-anwenden

Kapitel 59: Letzte eingefügte Identität

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();
```

Dies gibt den zuletzt hinzugefügten Identitätswert zurück, der für dieselbe Verbindung innerhalb des aktuellen Bereichs erzeugt wurde. In diesem Fall 1 für die erste Zeile in der Tabelle dbo.person.

@@IDENTITÄT

```
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;
```

Dadurch wird die zuletzt hinzugefügte Identität für dieselbe Verbindung unabhängig vom Bereich zurückgegeben. Unabhängig vom aktuellen Wert der Identitätsspalte in logging_table gilt in

diesem Fall, dass in der Instanz von SQL Server keine anderen Aktivitäten ausgeführt werden und keine anderen Auslöser von dieser Einfügung ausgelöst werden.

IDENT_CURRENT ('Tabellenname')

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

Dadurch wird der zuletzt hinzugefügte Identitätswert für die ausgewählte Tabelle ausgewählt, unabhängig von Verbindung oder Bereich.

IDENTITY und 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
```

Die Werte der letzten beiden SELECT-Anweisungen sind gleich.

Letzte eingefügte Identität online lesen: https://riptutorial.com/de/sql-server/topic/5674/letzte-eingefugte-identitat

Kapitel 60: Logische Funktionen

Examples

WÄHLEN

SQL Server 2012

Gibt das Element am angegebenen Index aus einer Liste von Werten zurück. Wenn der `index` die Grenzen der `values` überschreitet `values` wird `NULL` zurückgegeben.

Parameter:

1. `index` : Ganzzahl, Index zu Element in `values` . 1-basiert.
2. `values` : beliebiger Typ, kommagetrennte Liste

```
SELECT CHOOSE (1, 'apples', 'pears', 'oranges', 'bananas') AS chosen_result

chosen_result
-------------
apples
```

IIF

SQL Server 2012

Gibt einen von zwei Werten zurück, abhängig davon, ob ein bestimmter boolescher Ausdruck als wahr oder falsch ausgewertet wird.

Parameter:

1. `boolean_expression` ausgewertet, um festzulegen, welchen Wert zurückgegeben werden soll
2. `true_value` zurückgegeben, wenn `boolean_expression` als true ausgewertet wird
3. `false_value` zurückgegeben, wenn `boolean_expression` als false ausgewertet wird

```
SELECT IIF (42 > 23, 'I knew that!', 'That is not true.') AS iif_result

iif_result
------------
I knew that!
```

SQL Server 2012

`IIF` kann durch eine `CASE` Anweisung ersetzt werden. Das obige Beispiel kann als geschrieben werden

```
SELECT CASE WHEN 42 > 23 THEN 'I knew that!' ELSE 'That is not true.' END AS iif_result

iif_result
```

```
------------
I knew that!
```

Logische Funktionen online lesen: https://riptutorial.com/de/sql-server/topic/10647/logische-funktionen

Kapitel 61: Microsoft SQL Server Management Studio-Tastenkombinationen

Examples

Verknüpfungsbeispiele

1. Öffnet ein neues Abfragefenster mit der aktuellen Verbindung (`Strg` + `N`)
2. Zwischen geöffneten Tabs wechseln (`Strg` + `Tab`)
3. Ergebnisbereich einblenden / ausblenden (`Strg` + `R`)
4. Markierte Abfrage ausführen (`Strg` + `E`)
5. Ausgewählten Text in Groß- oder Kleinbuchstaben umwandeln (`Strg` + `Umschalt` + `U` , `Strg` + `Umschalt` + `L`)
6. Intellisense-Listenmitglied und vollständiges Wort (`Strg` + `Leerzeichen` , `Tabulator`)
7. Gehen Sie zur Zeile (`Strg` + `G`)
8. Schließen Sie eine Registerkarte in SQL Server Managament Studio (`Strg` + `F4`).

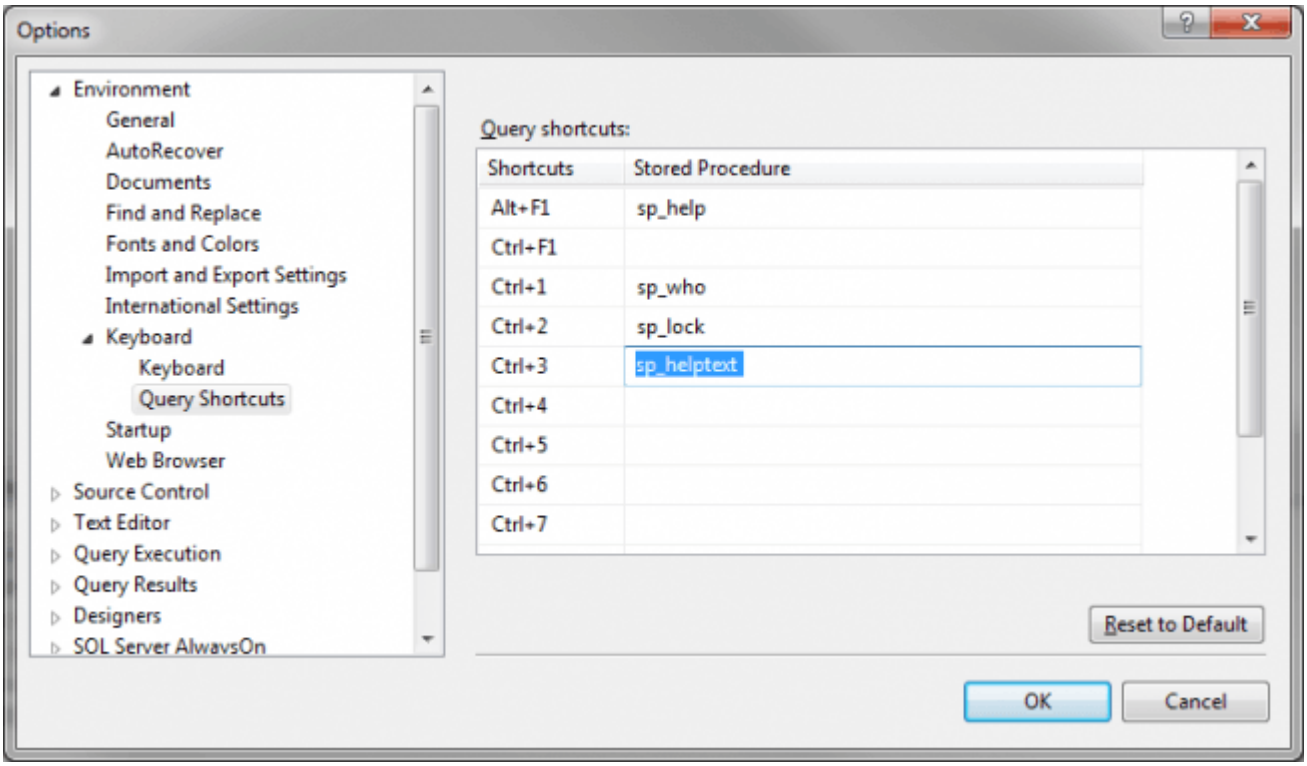
Tastenkombinationen für die Menüaktivierung

1. Wechseln Sie in die Menüleiste von SQL Server Management Studio (`ALT`).
2. Aktivieren Sie das Menü für eine Werkzeugkomponente (`ALT` + `HYPHEN`)
3. Anzeige des Kontextmenüs (`UMSCHALT` + `F`)
4. Rufen Sie das Dialogfeld "Neue Datei" auf, um eine Datei zu erstellen (`STRG` + `N`).
5. Zeigen Sie das Dialogfeld Projekt öffnen an, um ein vorhandenes Projekt zu öffnen (`STRG` + `UMSCHALT` + `O`).
6. Zeigen Sie das Dialogfeld Neues Element hinzufügen an, um dem aktuellen Projekt eine neue Datei hinzuzufügen (`STRG` + `UMSCHALT` + `A`).
7. Zeigen Sie das Dialogfeld Vorhandenes Element hinzufügen an, um dem aktuellen Projekt eine vorhandene Datei hinzuzufügen (`STRG` + `UMSCHALT` + `A`).
8. Anzeigen des Abfrage-Designers (`STRG` + `UMSCHALT` + `Q`)
9. Schließen Sie ein Menü oder ein Dialogfeld, um die Aktion abzubrechen (`ESC`).

Benutzerdefinierte Tastenkombinationen

Gehen Sie zu Extras -> Optionen. Gehen Sie zu Umgebung -> Tastatur -> Abfrageverknüpfungen

Auf der rechten Seite sehen Sie einige Verknüpfungen, die standardmäßig in SSMS verwendet werden. Wenn Sie jetzt eine neue hinzufügen möchten, klicken Sie einfach auf eine Spalte unter der Spalte Gespeicherte Prozedur.



OK klicken. Gehen Sie nun zu einem Abfragefenster und wählen Sie die gespeicherte Prozedur aus. Drücken Sie dann STRG + 3, um das Ergebnis der gespeicherten Prozedur anzuzeigen.

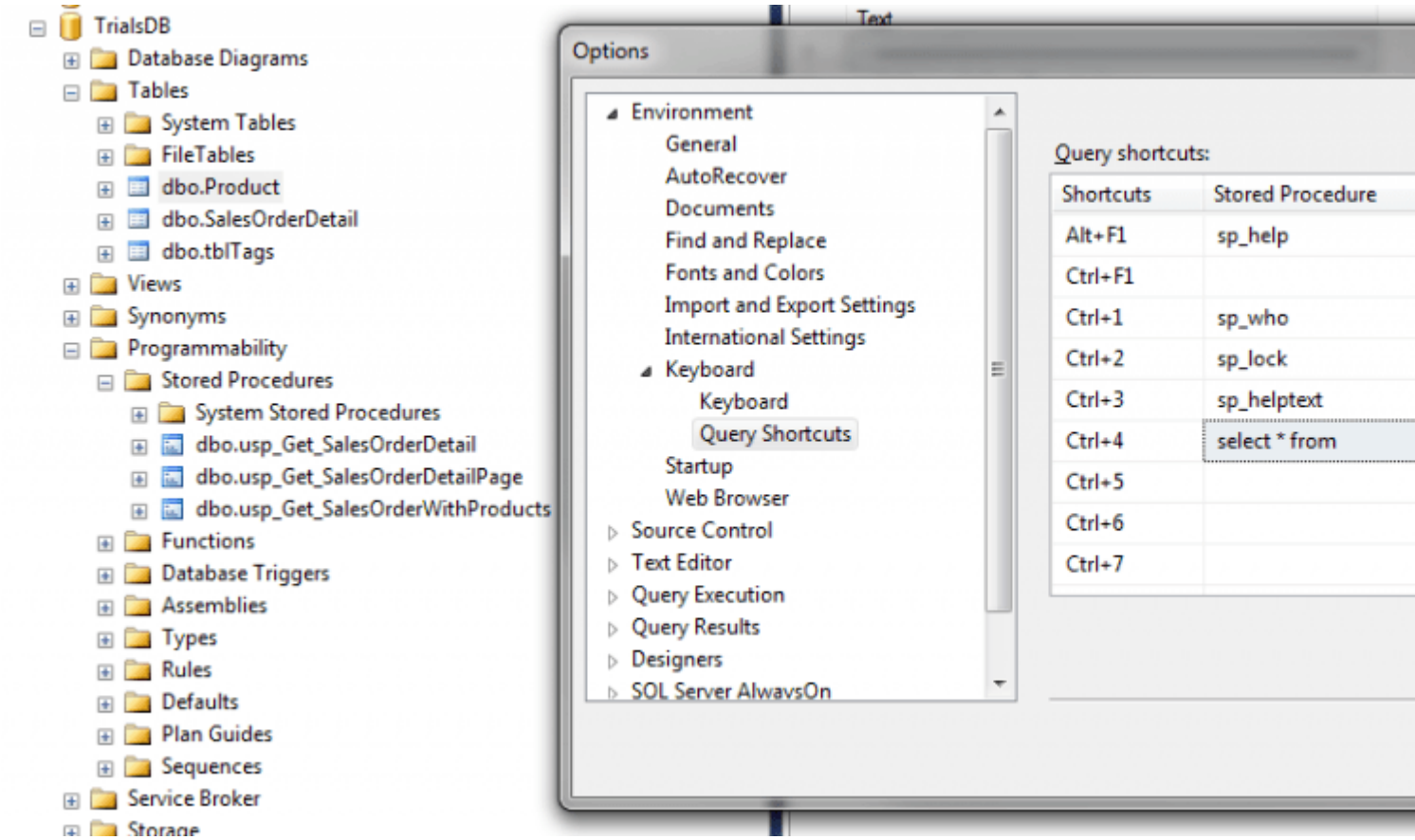
```
usp_Get_SalesOrderDetail
```

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

Wenn Sie nun alle Datensätze aus einer Tabelle auswählen müssen, wählen Sie die Tabelle aus und drücken Sie STRG + 5 (Sie können eine beliebige Taste auswählen). Sie können die Verknüpfung wie folgt erstellen.



Gehen Sie nun weiter und wählen Sie den Tabellennamen aus dem Abfragefenster aus und drücken Sie STRG + 4 (die ausgewählte Taste). Das Ergebnis wird angezeigt.

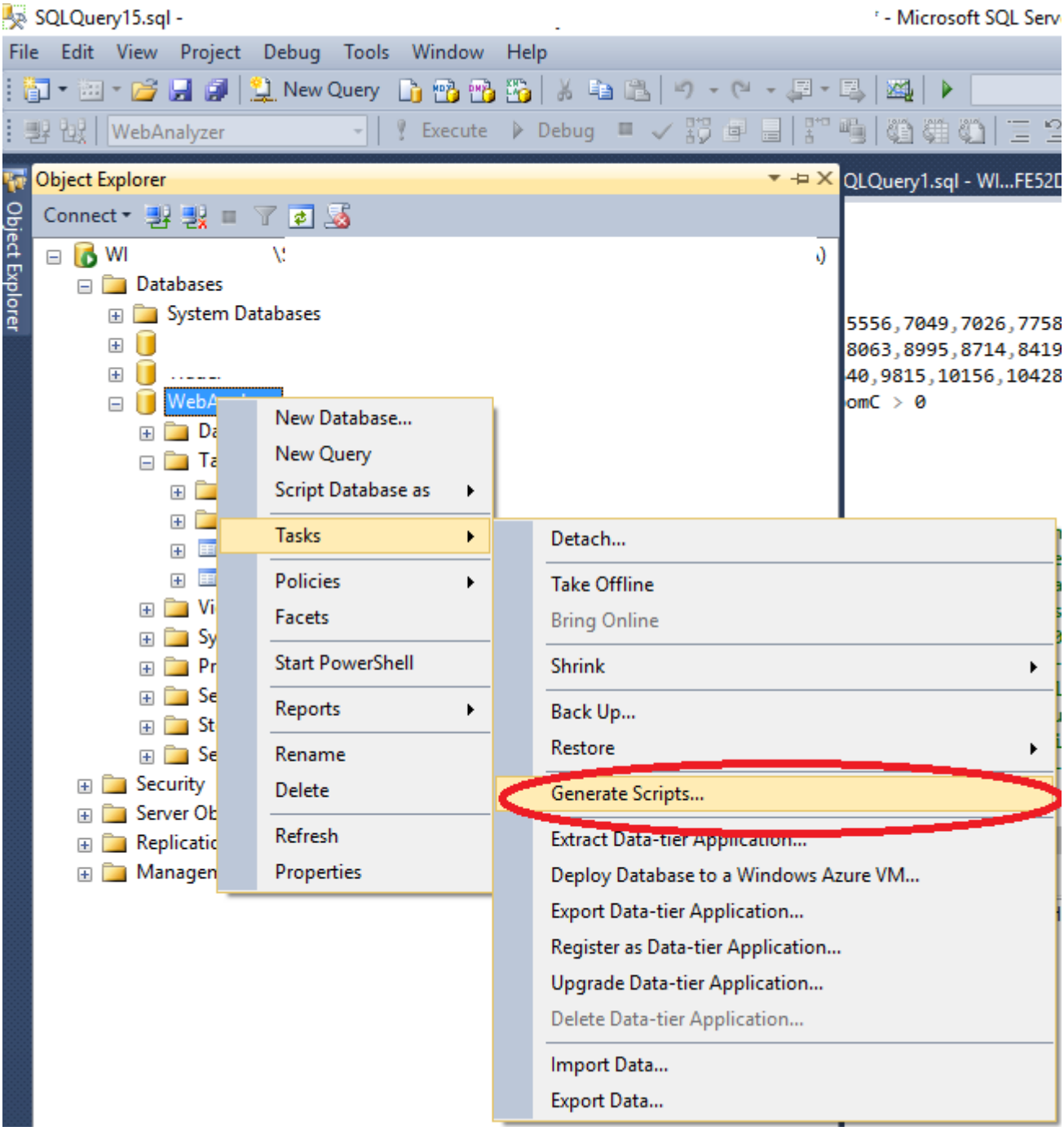
Microsoft SQL Server Management Studio-Tastenkombinationen online lesen: https://riptutorial.com/de/sql-server/topic/7749/microsoft-sql-server-management-studio-tastenkombinationen

Kapitel 62: Migration

Examples

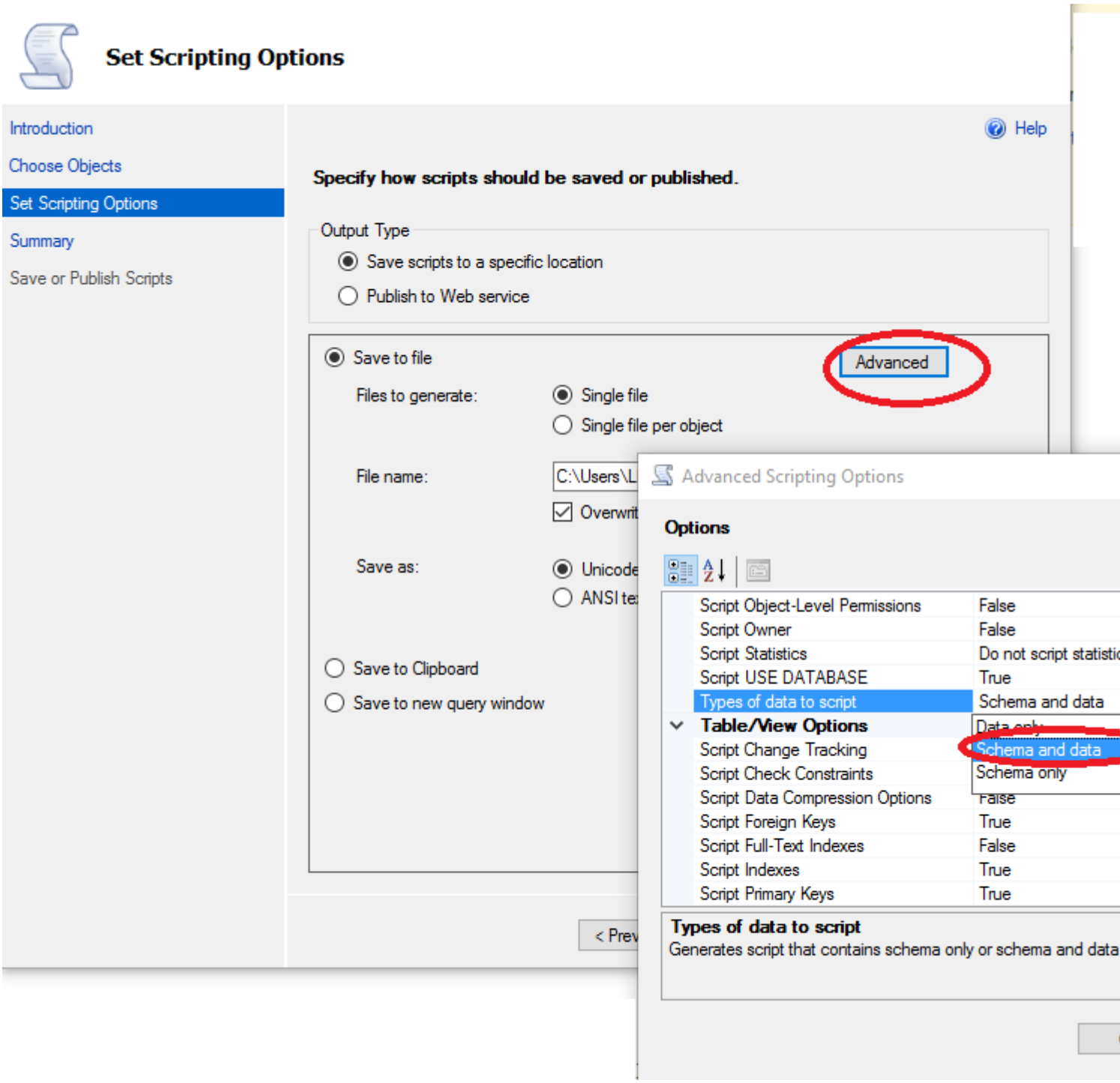
So erstellen Sie Migrationsskripte

1. Klicken Sie mit der `rechten Maustaste` auf die Datenbank, die Sie migrieren möchten, und wählen Sie dann -> `Tasks` -> `Generate Scripts...`



2. Assistent wird geöffnet Klicken `Next` dann Objekte wählen Sie migrieren möchten , und klicken Sie auf `Next` erneut, klicken Sie dann auf `Advanced` blättern ein wenig nach unten und in `Types of data to script` das `Schema and data Types of data to script` wählen `Schema and`

`data` (es sei denn , Sie wollen nur Strukturen)



3. Klicken paar Mal `Next` und `Finish` stellen, und Sie sollten Ihre Datenbank in scripted haben `.sql` - Datei.

4. Führen Sie die `.sql` Datei auf Ihrem neuen Server aus, und Sie sollten fertig sein.

Migration online lesen: https://riptutorial.com/de/sql-server/topic/4451/migration

Kapitel 63: Mit Krawatten Option

Examples

Testdaten

```
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'
```

Unten ist die Ausgabe der obigen Tabelle. Wie Sie sehen, wird die Id-Spalte dreimal wiederholt.

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

Jetzt können wir die Ausgabe mit der einfachen Reihenfolge nach prüfen.

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

Ausgabe: (Die Ausgabe der obigen Abfrage ist nicht immer gleich).

```
Id   Name
1    B
```

Lass die gleiche Abfrage mit der Option Ties ausführen.

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

Ausgabe :

```
Id   Name
```

```
1    A
1    B
1    C
```

Wie Sie sehen, gibt SQL Server alle Zeilen aus, **die mit der** Reihenfolge nach Spalte verbunden sind. Sehen wir uns noch ein Beispiel an, um das besser zu verstehen.

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

Ausgabe:

```
Id   Name
1    A
```

Zusammenfassend gibt SQL Server bei Verwendung von Ties Option alle gebundenen Zeilen aus, unabhängig von den von uns festgelegten Grenzwerten

Mit Krawatten Option online lesen: https://riptutorial.com/de/sql-server/topic/2546/mit-krawatten-option

Kapitel 64: Nativ zusammengestellte Module (Hekaton)

Examples

Nativ kompilierte gespeicherte Prozedur

In einer Prozedur mit systemeigener Kompilierung wird T-SQL-Code in dll kompiliert und als nativer C-Code ausgeführt. Um eine native, kompilierte gespeicherte Prozedur zu erstellen, müssen Sie:

- Verwenden Sie die standardmäßige CREATE PROCEDURE-Syntax
- Legen Sie die Option NATIVE_COMPILATION in der Definition der gespeicherten Prozedur fest
- Verwenden Sie die Option SCHEMABINDING in der Definition gespeicherter Prozeduren
- Definieren Sie die Option EXECUTE AS OWNER in der Definition der gespeicherten Prozedur

Anstelle des Standard-BEGIN END-Blocks müssen Sie den BEGIN ATOMIC-Block verwenden:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Beispiel:

```
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
```

Nativ kompilierte Skalarfunktion

Code in nativ kompilierter Funktion wird in C-Code umgewandelt und als DLL kompiliert. Um eine native kompilierte Skalarfunktion zu erstellen, müssen Sie:

- Verwenden Sie die standardmäßige CREATE FUNCTION-Syntax
- Legen Sie die Option NATIVE_COMPILATION in der Funktionsdefinition fest
- Verwenden Sie die Option SCHEMABINDING in der Funktionsdefinition

Anstelle des Standard-BEGIN END-Blocks müssen Sie den BEGIN ATOMIC-Block verwenden:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Beispiel:

```
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)
```

Native Inline-Tabellenwertfunktion

Die native kompilierte Tabellenwertfunktion gibt eine Tabelle als Ergebnis zurück. Code in nativ kompilierter Funktion wird in C-Code umgewandelt und als DLL kompiliert. In Version 2016 werden nur Inline-Tabellenwertfunktionen unterstützt. Um eine native Tabellenwertfunktion zu erstellen, müssen Sie:

- Verwenden Sie die standardmäßige CREATE FUNCTION-Syntax
- Legen Sie die Option NATIVE_COMPILATION in der Funktionsdefinition fest
- Verwenden Sie die Option SCHEMABINDING in der Funktionsdefinition

Anstelle des Standard-BEGIN END-Blocks müssen Sie den BEGIN ATOMIC-Block verwenden:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Beispiel:

```
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
);
```

Nativ zusammengestellte Module (Hekaton) online lesen: https://riptutorial.com/de/sql-server/topic/6089/nativ-zusammengestellte-module--hekaton-

Kapitel 65: NULL

Einführung

In SQL Server steht `NULL` für fehlende oder unbekannte Daten. Dies bedeutet, dass `NULL` nicht wirklich ein Wert ist. Es ist besser als Platzhalter für einen Wert beschrieben. Dies ist auch der Grund, warum Sie `NULL` mit keinem Wert und nicht einmal mit einem anderen `NULL` .

Bemerkungen

SQL Server bietet andere Methoden zum Behandeln von Nullwerten, z. B. `IS NULL` , `IS NOT NULL` , `ISNULL()` , `COALESCE()` und andere.

Examples

NULL-Vergleich

`NULL` ist ein Sonderfall, wenn es um Vergleiche geht.

Nehmen Sie die folgenden Daten an.

```
id someVal
 ----
 0 NULL
 1 1
 2 2
```

Mit einer Abfrage:

```
 SELECT id
 FROM table
 WHERE someVal = 1
```

würde id `1`

```
 SELECT id
 FROM table
 WHERE someVal <> 1
```

würde id `2`

```
 SELECT id
 FROM table
 WHERE someVal IS NULL
```

würde id `0`

```
SELECT id
FROM table
WHERE someVal IS NOT NULL
```

würde beide IDs `1` und `2` .

Wenn Sie möchten, dass NULL-Werte als Werte in einem Vergleich von `=` , `<>` "gezählt" werden, muss dieser zunächst in einen abzählbaren Datentyp konvertiert werden:

```
SELECT id
FROM table
WHERE ISNULL(someVal, -1) <> 1
```

ODER

```
SELECT id
FROM table
WHERE someVal IS NULL OR someVal <> 1
```

gibt `0` und `2`

Oder Sie können Ihre ANSI-Null- Einstellung ändern.

ANSI-NULLEN

Von MSDN

> In einer zukünftigen Version von SQL Server ist ANSI_NULLS immer ON, und alle Anwendungen, die die Option explizit auf OFF setzen, erzeugen einen Fehler. Vermeiden Sie die Verwendung dieser Funktion in neuen Entwicklungsarbeiten und planen Sie, Anwendungen zu ändern, die diese Funktion derzeit verwenden.

`ANSI NULLS` auf off gesetzt ist, können `ANSI NULLS` zwischen `=` / `<>` verglichen werden.

Angesichts der folgenden Daten:

```
id someVal
 ----
 0 NULL
 1 1
 2 2
```

Und mit ANSI NULLS an, diese Abfrage:

```
SELECT id
FROM table
WHERE someVal = NULL
```

würde keine Ergebnisse produzieren. Jedoch die gleiche Abfrage mit ANSI NULLS aus:

```
set ansi_nulls off

SELECT id
FROM table
WHERE someVal = NULL
```

Würde id `0` .

IST NULL()

Die `IsNull()` Funktion akzeptiert zwei Parameter und gibt den zweiten Parameter zurück, wenn der erste `null` .

Parameter:

1. überprüfe den Ausdruck. Jeder Ausdruck eines beliebigen Datentyps.
2. Wiederbeschaffungswert. Dies ist der Wert, der zurückgegeben würde, wenn der Prüfausdruck null ist. Der Ersetzungswert muss von einem Datentyp sein, der implizit in den Datentyp des Prüfausdrucks konvertiert werden kann.

Die `IsNull()` Funktion gibt denselben Datentyp wie der `IsNull()` zurück.

```
DECLARE @MyInt int -- All variables are null until they are set with values.

SELECT ISNULL(@MyInt, 3) -- Returns 3.
```

Siehe auch `COALESCE` oben

Ist null / ist nicht null

Da null kein Wert ist, können Sie keine Vergleichsoperatoren mit Nullwerten verwenden.
Um zu überprüfen, ob eine Spalte oder Variable null enthält, müssen Sie `is null` :

```
DECLARE @Date date = '2016-08-03'
```

Die folgende Anweisung wählt den Wert `6` , da alle Vergleiche mit Nullwerten als falsch oder unbekannt ausgewertet werden:

```
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
```

Wenn Sie den Inhalt der @Date-Variablen auf `null` und erneut versuchen, gibt die folgende Anweisung `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 ()` Wertet die Argumente in der Reihenfolge aus und gibt den aktuellen Wert des ersten Ausdrucks zurück, der anfangs nicht `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.
```

Obwohl ISNULL () ähnlich wie COALESCE () arbeitet, akzeptiert die Funktion ISNULL () nur zwei Parameter - einen zu überprüfenden Parameter und einen, wenn der erste Parameter NULL ist. Siehe auch `ISNULL` unten

NULL mit NOT IN SubQuery

Bei der Bearbeitung in einer Unterabfrage mit Null in der Unterabfrage müssen wir NULLS entfernen, um die erwarteten Ergebnisse zu erhalten

```
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
```

Seien Sie bei der Bearbeitung nicht in einer Unterabfrage mit null vorsichtig mit Ihrer erwarteten Ausgabe

NULL online lesen: https://riptutorial.com/de/sql-server/topic/5044/null

Kapitel 66: OPENJSON

Examples

Schlüssel abrufen: Wertepaare aus JSON-Text

Die Funktion OPENJSON analysiert den JSON-Text und gibt alle Schlüssel / Wert-Paare auf der ersten Ebene von JSON zurück:

```
declare @json NVARCHAR(4000) = N'{"Name":"Joe","age":27,"skills":["C#","SQL"]}';
SELECT * FROM OPENJSON(@json);
```

Schlüssel	Wert	Art
Name	Joe	1
Alter	27	2
Kompetenzen	["C #", "SQL"]	4

Der Spaltentyp beschreibt den Wertetyp, dh null (0), String (1), number (2), boolean (3), Array (4) und Objekt (5).

Wandeln Sie das JSON-Array in einen Satz von Zeilen um

Die Funktion OPENJSON analysiert die Sammlung von JSON-Objekten und gibt Werte aus dem JSON-Text als Satz von Zeilen zurück.

```
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
  )
```

In der WITH-Klausel ist das Rückgabeschema der OPENJSON-Funktion angegeben. Schlüssel in den JSON-Objekten werden anhand von Spaltennamen abgerufen. Wenn ein Schlüssel in JSON nicht in der WITH-Klausel angegeben ist (z. B. Price in diesem Beispiel), wird er ignoriert. Werte werden automatisch in angegebene Typen konvertiert.

Nummer	Datum	Kunde	Menge
SO43659	2011-05-31T00: 00: 00	MSFT	1
SO43661	2011-06-01T00: 00: 00	Nokia	3

Wandeln Sie geschachtelte JSON-Felder in Zeilengruppen um

Die Funktion OPENJSON analysiert die Sammlung von JSON-Objekten und gibt Werte aus dem JSON-Text als Satz von Zeilen zurück. Wenn die Werte im Eingabeobjekt verschachtelt sind, können in jeder Spalte in WITH-Klausel zusätzliche Zuordnungsparameter angegeben werden:

```
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',
  )
```

In der WITH-Klausel ist das Rückgabeschema der OPENJSON-Funktion angegeben. Nachdem der Typ angegeben ist, wird der Pfad zu den JSON-Knoten angegeben, auf denen der zurückgegebene Wert gefunden werden soll. Schlüssel in den JSON-Objekten werden von diesen Pfaden abgerufen. Werte werden automatisch in angegebene Typen konvertiert.

Nummer	Datum	Kunde	Menge
SO43659	2011-05-31T00: 00: 00	MSFT	1
SO43661	2011-06-01T00: 00: 00	Nokia	3

Extrahieren von inneren JSON-Unterobjekten

OPENJSON kann Fragmente von JSON-Objekten im JSON-Text extrahieren. Setzen Sie in der Spaltendefinition, die auf ein JSON-Unterobjekt verweist, den Typ nvarchar (max) und die Option 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
  )
```

Die Info-Spalte wird dem "Info" -Objekt zugeordnet. Ergebnisse werden sein:

Nummer	Datum	Info
SO43659	2011-05-31T00: 00: 00	{"Kunde": "MSFT", "Preis": 59,99, "Anzahl": 1}
SO43661	2011-06-01T00: 00: 00	{"Kunde": "Nokia", "Preis": 24,99, "Anzahl": 3}

Arbeiten mit verschachtelten JSON-Subarrays

JSON kann eine komplexe Struktur mit inneren Arrays aufweisen. In diesem Beispiel haben wir ein Array von Bestellungen mit verschachtelten Unterarrays von 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}]}

]'
```

Wir können Eigenschaften der Stammebene mit OPENJSON analysieren, wodurch das Items-Array AS JSON-Fragment zurückgegeben wird. Dann können wir OPENJSON erneut auf das Items-Array anwenden und die innere JSON-Tabelle öffnen. Die Tabelle der ersten Ebene und die innere Tabelle werden wie beim JOIN zwischen Standardtabellen "verbunden":

```
SELECT    *
FROM
    OPENJSON (@json)
    WITH (  Number varchar(200), Date datetime,
            Items nvarchar(max) AS JSON )
        CROSS APPLY
            OPENJSON (Items)
            WITH ( Price float, Quantity int)
```

Ergebnisse:

Nummer	Datum	Artikel	Preis	Menge
SO43659	2011-05-31 00: 00: 00.000	[{"Preis": 11,99, "Anzahl": 1}, {"Preis": 12,99, "Anzahl": 5}]	11,99	1

Nummer	Datum	Artikel	Preis	Menge
SO43659	2011-05-31 00: 00: 00.000	[{"Preis": 11,99, "Anzahl": 1}, {"Preis": 12,99, "Anzahl": 5}]	12,99	5
SO43661	2011-06-01 00: 00: 00.000	[{"Preis": 21.99, "Anzahl": 3}, {"Preis": 22.99, "Menge": 2}, {"Preis": 23.99, "Menge": 2}]	21,99	3
SO43661	2011-06-01 00: 00: 00.000	[{"Preis": 21.99, "Anzahl": 3}, {"Preis": 22.99, "Menge": 2}, {"Preis": 23.99, "Menge": 2}]	22,99	2
SO43661	2011-06-01 00: 00: 00.000	[{"Preis": 21.99, "Anzahl": 3}, {"Preis": 22.99, "Menge": 2}, {"Preis": 23.99, "Menge": 2}]	23,99	2

OPENJSON online lesen: https://riptutorial.com/de/sql-server/topic/5030/openjson

Kapitel 67: Parsename

Syntax

- PARSENAME ('object_name', object_piece)

Parameter

'Objektname'	object_piece
Der Name des Objekts, für das der angegebene Objektteil abgerufen werden soll. Objektname ist Sysname. Dieser Parameter ist ein optional qualifizierter Objektname. Wenn alle Teile des Objektnamens qualifiziert sind, kann dieser Name aus vier Teilen bestehen: dem Servernamen, dem Datenbanknamen, dem Eigentümernamen und dem Objektnamen.	Ist der Objektteil, der zurückgegeben werden soll. object_piece ist vom Typ int und kann folgende Werte annehmen: 1 = Objektname 2 = Schemaname 3 = Datenbankname 4 = Servername

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
```

Kehrt zurück:

Server	DB
HeadofficeSQL1	Nordwind

Inhaber	Objekt
dbo	Autoren

Parsename online lesen: https://riptutorial.com/de/sql-server/topic/5775/parsename

Kapitel 68: Partitionierung

Examples

Partitionsgrenzwerte abrufen

```
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
```

Partitionen wechseln

Gemäß dieser [TechNet-Microsoft-Seite] [1]

> **Durch die Partitionierung von Daten** können Sie Teilmengen Ihrer Daten schnell und effizient verwalten und auf sie zugreifen, während die Integrität der gesamten Datenerfassung erhalten bleibt.

Wenn Sie die folgende Abfrage aufrufen, werden die Daten nicht physisch verschoben. Nur die Metadaten zum Speicherort der Daten ändern sich.

```
ALTER TABLE [SourceTable] SWITCH TO [TargetTable]
```

Die Tabellen müssen dieselben Spalten mit denselben Datentypen und NULL-Einstellungen aufweisen, sie müssen sich in derselben Dateigruppe befinden und die neue Zieltabelle muss leer sein. Weitere Informationen zum Wechseln von Partitionen finden Sie unter dem Seitenlink oben.

[1]: https://technet.microsoft.com/en-us/library/ms191160(v=sql.105).aspx Die `IDENTITY` Eigenschaft der Spalte kann abweichen.

Rufen Sie die Werte der Partitionstabelle, der Spalte, des Schemas, der Funktion, der Gesamtzahl und der Min-Max-Grenze mithilfe einer einzelnen Abfrage ab

```
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)
```

Sie müssen nur die `where` Klausel `table_name` und den `table_name` durch den `actual table name` ersetzen, um die Details des gewünschten Objekts anzuzeigen.

Partitionierung online lesen: https://riptutorial.com/de/sql-server/topic/3212/partitionierung

Kapitel 69: PHANTOM lesen

Einführung

In Datenbanksystemen bestimmt die Isolation, wie die Transaktionsintegrität für andere Benutzer und Systeme sichtbar ist. Sie definiert, wie und wann die durch einen Vorgang vorgenommenen Änderungen für andere sichtbar werden. Das Phantom Read kann auftreten, wenn Sie Daten erhalten, die noch nicht in die Datenbank übernommen wurden.

Bemerkungen

Sie können die verschiedenen `ISOLATION LEVEL` auf MSDN lesen

Examples

Isolationsstufe LESEN UNBESTIMMT

Erstellen Sie eine Beispieltabelle für eine Beispieldatenbank

```
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]
```

Öffnen Sie nun einen Ersten Abfrage-Editor (in der Datenbank), fügen Sie den folgenden Code ein und führen Sie ihn aus (**berühren Sie nicht das --rollback**). In diesem Fall fügen Sie eine Zeile in die Datenbank ein, **ohne** Änderungen zu übernehmen.

```
begin tran

INSERT INTO Table_1 values('Title 1')

SELECT * FROM [Test].[dbo].[Table_1]

--rollback
```

Öffnen Sie nun einen zweiten Abfrage-Editor (in der Datenbank), geben Sie den folgenden Code ein und führen Sie ihn aus

```
begin tran

set transaction isolation level READ UNCOMMITTED
```

```
SELECT * FROM [Test].[dbo].[Table_1]
```

Möglicherweise stellen Sie fest, dass im zweiten Editor die neu erstellte Zeile (jedoch nicht festgeschrieben) aus der ersten Transaktion angezeigt wird. Führen Sie im ersten Editor das Rollback aus (wählen Sie das Rollback-Wort aus und führen Sie es aus).

```
-- Rollback the first transaction
rollback
```

Führen Sie die Abfrage im zweiten Editor aus, und Sie sehen, dass der Datensatz verschwunden ist (Phantom Read). Dies geschieht, weil Sie der zweiten Transaktion mitteilen, alle Zeilen und auch die nicht festgeschriebenen Daten abzurufen.

Dies tritt auf, wenn Sie die Isolationsstufe mit ändern

```
set transaction isolation level READ UNCOMMITTED
```

PHANTOM lesen online lesen: https://riptutorial.com/de/sql-server/topic/8235/phantom-lesen

Kapitel 70: PIVOT / UNPIVOT

Syntax

- SELECT `<non-pivoted column>`
  [erste geschwenkte Spalte] AS `<column name>` ,
  [zweite geschwenkte Spalte] AS `<column name>` ,
  ...
  [letzte geschwenkte Spalte] AS `<column name>`
  VON
  (`<SELECT query that produces the data>`)
  AS `<alias for the source query>`
  PIVOT
  (
  `<aggregation function>` (`<column being aggregated>`)
  ZUM
  [`<column that contains the values that will become column headers>`]
  IN ([erste Schwenksäule], [zweite Schwenksäule],
  ... [letzte geschwenkte Spalte])
) AS `<alias for the pivot table>` `<optional ORDER BY clause>` ;

Bemerkungen

Mit PIVOT- und UNPIVOT-Operatoren transformieren Sie eine Tabelle, indem Sie die Zeilen (Spaltenwerte) einer Tabelle in Spalten verschieben und umgekehrt. Als Teil dieser Transformationsaggregation können Funktionen auf die Tabellenwerte angewendet werden.

Examples

Simple Pivot - Statische Spalten

Lassen Sie uns mithilfe der Artikelverkaufstabelle aus der Beispieldatenbank die von jedem Produkt verkaufte Gesamtmenge berechnen und anzeigen.

Dies kann leicht mit einer Gruppe durch erledigt werden, nehmen wir jedoch an, dass wir unsere Ergebnistabelle so "drehen", dass für jede Produkt-ID eine Spalte vorhanden ist.

```
SELECT [100], [145]
  FROM (SELECT ItemId , Quantity
          FROM #ItemSalesTable
       ) AS pivotIntermediate
 PIVOT (   SUM(Quantity)
           FOR ItemId IN ([100], [145])
       ) AS pivotTable
```

Da unsere 'neuen' Spalten Zahlen sind (in der Quelltabelle), müssen eckige Klammern `[]`

Dies gibt uns eine Ausgabe wie

100	145
45	18

Einfaches PIVOT & UNPIVOT (T-SQL)

Im Folgenden finden Sie ein einfaches Beispiel, das den durchschnittlichen Artikelpreis für jeden Artikel pro Wochentag zeigt.

Angenommen, wir haben eine Tabelle, in der die Preise aller Artikel täglich aufgezeichnet werden.

```
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);
```

Die Tabelle sollte wie folgt aussehen:

```
+========+=========+=======+
|   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 |
+--------+---------+-------+
```

Um eine Aggregation durchzuführen, mit der der Durchschnittspreis pro Artikel für jeden Wochentag ermittelt werden soll, verwenden wir den relationalen Operator `PIVOT` , um den Spalten-`weekday` des Ausdrucks in Tabellenwerten wie folgt in aggregierte `PIVOT` zu drehen:

```
SELECT * FROM tbl_stock
PIVOT (
    AVG(price) FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) pvt;
```

Ergebnis:

```
+--------+------+------+------+------+------+
|  item  |  Mon |  Tue |  Wed |  Thu |  Fri |
+--------+------+------+------+------+------+
|  Item1 |  116 |  112 |  117 |  109 |  120 |
|  Item2 |  227 |  233 |  230 |  228 |  210 |
|  Item3 |  145 |  158 |  152 |  145 |  125 |
+--------+------+------+------+------+------+
```

Um die umgekehrte Operation von `PIVOT` auszuführen, können wir den relationalen Operator `UNPIVOT` , um Spalten wie `UNPIVOT` in Zeilen zu drehen:

```
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;
```

Ergebnis:

```
+=======+========+=========+
|  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 |
+-------+--------+---------+
```

Dynamisches PIVOT

Ein Problem bei der `PIVOT` Abfrage besteht darin, dass Sie alle Werte in der `IN` Auswahl angeben müssen, wenn Sie sie als Spalten anzeigen möchten. Sie können dieses Problem schnell umgehen, indem Sie eine dynamische IN-Auswahl erstellen, die Ihren `PIVOT` dynamisiert.

Zur Demonstration verwenden wir eine Tabelle `Books` in der Datenbank eines `Bookstore` . Wir gehen davon aus, dass die Tabelle nicht normalisiert ist und folgende Spalten enthält

```
Table: Books
-----------------------------
BookId (Primary Key Column)
Name
Language
NumberOfPages
EditionNumber
YearOfPrint
YearBoughtIntoStore
ISBN
AuthorName
Price
NumberOfUnitsSold
```

Das Erstellungsskript für die Tabelle sieht folgendermaßen aus:

```
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
```

Wenn wir nun die Datenbank abfragen und die Anzahl der Bücher in den Sprachen Englisch, Russisch, Deutsch, Hindi und Latein herausfinden müssen, die jedes Jahr im Buchladen gekauft wurden, und unsere Ausgabe in einem kleinen Berichtsformat präsentieren, können wir die

PIVOT-Abfrage wie folgt verwenden

```
SELECT * FROM
  (
   SELECT YearBoughtIntoStore AS [Year Bought],[Language], NumberOfBooks
   FROM BookList
  ) sourceData
 PIVOT
  (
  SUM(NumberOfBooks)
  FOR [Language] IN (English, Russian, German, Hindi, Latin)
  ) pivotrReport
```

Sonderfall ist, wenn wir keine vollständige Liste der Sprachen haben, also verwenden wir dynamisches SQL wie folgt

```
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)
```

PIVOT / UNPIVOT online lesen: https://riptutorial.com/de/sql-server/topic/591/pivot---unpivot

Kapitel 71: Primärschlüssel

Bemerkungen

Primärschlüssel werden verwendet, um einen Datensatz in einer Tabelle eindeutig zu identifizieren. Eine Tabelle darf nur einen einzigen Primärschlüssel haben (obwohl der Primärschlüssel aus mehreren Spalten bestehen kann), und für bestimmte Replikationstypen ist ein Primärschlüssel erforderlich.

Primärschlüssel werden häufig als Clustered-Index für eine Tabelle verwendet (müssen aber nicht sein).

Examples

Erstellen Sie eine Tabelle mit Identität als Primärschlüssel

```
-- 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
)
```

Erstellen Sie eine Tabelle mit dem GUID-Primärschlüssel

```
-- 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
)
```

Erstellen Sie eine Tabelle mit natürlichem Schlüssel

```
-- 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
)
```

Erstellen Sie eine Tabelle mit zusammengesetztem Schlüssel

```
-- 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)
)
```

Fügen Sie der vorhandenen Tabelle einen Primärschlüssel hinzu

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

Wenn die Primärschlüsselspalte (in diesem Fall `ssn`) mehr als eine Zeile mit demselben Kandidatenschlüssel enthält, `ssn` die obige Anweisung fehl, da Primärschlüsselwerte eindeutig sein müssen.

Primärschlüssel löschen

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

Primärschlüssel online lesen: https://riptutorial.com/de/sql-server/topic/4543/primarschlussel

Kapitel 72: Privilegien oder Berechtigungen

Examples

Einfache Regeln

Erlaubnis zum Erstellen von Tabellen

```
USE AdventureWorks;
GRANT CREATE TABLE TO MelanieK;
GO
```

Erteilen der SHOWPLAN-Berechtigung für eine Anwendungsrolle

```
USE AdventureWorks2012;
GRANT SHOWPLAN TO AuditMonitor;
GO
```

CREATE VIEW mit GRANT OPTION gewähren

```
USE AdventureWorks2012;
GRANT CREATE VIEW TO CarmineEs WITH GRANT OPTION;
GO
```

Alle Rechte für einen Benutzer in einer bestimmten Datenbank erteilen

```
use YourDatabase
go
exec sp_addrolemember 'db_owner', 'UserName'
go
```

Privilegien oder Berechtigungen online lesen: https://riptutorial.com/de/sql-server/topic/5333/privilegien-oder-berechtigungen

Kapitel 73: Ranking-Funktionen

Syntax

- DENSE_RANK () OVER ([<partition_by_clause>] <order_by_clause>)
- RANK () OVER ([partition_by_clause] order_by_clause)

Parameter

Argumente	Einzelheiten
`<partition_by_clause>`	Teilt die von der FROM- Klausel erzeugte Ergebnismenge in Partitionen, auf die die `DENSE_RANK` Funktion angewendet wird. Für die `PARTITION BY` Syntax siehe OVER-Klausel (Transact-SQL) .
`<order_by_clause>`	Bestimmt die Reihenfolge, in der die `DENSE_RANK` Funktion auf die Zeilen in einer Partition angewendet wird.
`OVER ( [ partition_by_clause ] order_by_clause)`	`partition_by_clause` unterteilt die von der `FROM` Klausel erzeugte Ergebnismenge in Partitionen, auf die die Funktion angewendet wird. Wenn nicht angegeben, behandelt die Funktion alle Zeilen der Abfrageergebnisgruppe als eine einzige Gruppe. order_by_clause legt die Reihenfolge der Daten fest, bevor die Funktion angewendet wird. Die order_by_clause ist erforderlich. Die `<rows or range clause>` der `OVER` Klausel kann für die `RANK` Funktion nicht angegeben werden. Weitere Informationen finden Sie unter OVER-Klausel (Transact-SQL) .

Bemerkungen

Wenn zwei oder mehr Zeilen für einen Rang in derselben Partition identisch sind, erhält jede verbundene Zeile den gleichen Rang. Wenn beispielsweise die beiden Top-Vertriebsmitarbeiter den gleichen SalesYTD-Wert haben, werden sie beide auf Platz eins gesetzt. Der Verkäufer mit dem nächsthöheren SalesYTD steht auf Platz zwei. Dies ist eine Zahl mehr als die Anzahl der verschiedenen Reihen, die vor dieser Reihe stehen. Daher haben die von der Funktion `DENSE_RANK` zurückgegebenen Zahlen keine Lücken und haben immer aufeinanderfolgende Ränge.

Die für die gesamte Abfrage verwendete Sortierreihenfolge bestimmt die Reihenfolge, in der die Zeilen in einem Ergebnis angezeigt werden. Dies bedeutet, dass eine Zeile mit der Nummer Eins nicht die erste Zeile in der Partition sein muss.

`DENSE_RANK` ist nicht deterministisch. Weitere Informationen finden Sie unter Deterministische und Nicht-Deterministische Funktionen .

Examples

RANG()

A RANK () Gibt den Rang jeder Zeile in der Ergebnismenge der partitionierten Spalte zurück.

Z.B :

```
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 ()

Entspricht dem von RANK (). Es gibt den Rang ohne Lücken zurück:

```
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
```

Ranking-Funktionen online lesen: https://riptutorial.com/de/sql-server/topic/5031/ranking-funktionen

Kapitel 74: Räumliche Daten

Einführung

Es gibt zwei räumliche Datentypen

Geometrie- X / Y-Koordinatensystem für eine flache Oberfläche

Geographie Breiten- / Längengradkoordinatensystem für eine gekrümmte Oberfläche (die Erde). Es gibt mehrere Projektionen von gekrümmten Flächen, sodass jedes räumliche Geographiegebiet SQL Server wissen muss, welche Projektion verwendet werden soll. Die übliche Spatial Reference ID (SRID) ist 4326, dh Entfernungen in Kilometern. Dies ist die Standard-SRID, die in den meisten Webkarten verwendet wird

Examples

PUNKT

Erzeugt einen einzelnen Punkt. Dies ist abhängig von der verwendeten Klasse eine Geometrie oder ein geographischer Punkt.

Parameter	Detail
Lat oder X	Ist ein Gleitkommaausdruck, der die x-Koordinate des erzeugten Punkts darstellt
Lang oder y	Ist ein Gleitkommaausdruck, der die y-Koordinate des erzeugten Punkts darstellt
String	Bekannter Text (WKB) einer Geometrie- / Geographieform
Binär	Bekanntes binäres (WKB) einer Geometrie- / Geographieform
SRID	Ein int-Ausdruck, der die Raumbezugs-ID (SRID) der Geometrie / Geographie-Instanz darstellt, die Sie zurückgeben möchten

```
--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)
```

Räumliche Daten online lesen: https://riptutorial.com/de/sql-server/topic/6816/raumliche-daten

Kapitel 75: Reihen sortieren / sortieren

Examples

Grundlagen

Lassen Sie uns zunächst die Beispieltabelle einrichten.

```
-- 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
```

Beachten Sie, dass der SQL-Server beim Abrufen von Daten, wenn Sie keine Zeilenreihenfolgeklausel (ORDER BY) angeben, die Sortierung (Reihenfolge der Spalten) **zu keinem Zeitpunkt** gewährleistet. Wirklich zu jeder Zeit. Es gibt keinen Grund, darüber zu streiten, es wurde buchstäblich Tausende Male und im ganzen Internet gezeigt.

Keine ORDER BY == keine Sortierung. Ende der Geschichte.

```
-- 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
```

Es gibt zwei Richtungen, die nach Daten geordnet werden können:

- aufsteigend (aufwärts) mit ASC
- absteigend (abwärts) mit 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
```

Achten Sie bei der Bestellung in der Textspalte ((n) char oder (n) varchar) darauf, dass die Reihenfolge die Sortierung berücksichtigt. Weitere Informationen zur Kollatierung finden Sie unter dem Thema.

Das Sortieren und Sortieren von Daten kann Ressourcen beanspruchen. Hier sind richtig erstellte Indizes hilfreich. Weitere Informationen zu Indizes finden Sie unter dem Thema.

Es besteht die Möglichkeit, die Reihenfolge der Zeilen in Ihrem Resultset pseudozufällig zu sortieren. Erzwingen Sie einfach die Reihenfolge, um nicht deterministisch zu erscheinen.

```
SELECT * FROM SortOrder ORDER BY CHECKSUM(NEWID())
GO
```

Die Reihenfolge kann in einer gespeicherten Prozedur gespeichert werden. Dies ist die Art und Weise, in der Sie das tun sollten, wenn Sie das Rowset als letzten Schritt bearbeiten, bevor Sie es dem Endbenutzer zeigen.

```
CREATE PROCEDURE GetSortOrder
AS
    SELECT *
    FROM SortOrder
    ORDER BY ID DESC
GO

EXEC GetSortOrder
GO
```

Es gibt auch eine begrenzte (und harte) Unterstützung für das Sortieren in den SQL Server-Ansichten. Es wird jedoch empfohlen, diese Option NICHT zu verwenden.

```
/* 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
```

Zur Bestellung können Sie in Ihrem ORDER BY entweder Spaltennamen, Aliasnamen oder Spaltennummern verwenden.

```
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
```

Ich rate davon ab, die Zahlen in Ihrem Code zu verwenden, es sei denn, Sie möchten es gleich nach der Ausführung vergessen.

Bestellung nach Fall

Wenn Sie Ihre Daten numerisch oder alphabetisch sortieren möchten, können Sie einfach `order by [column]` . Wenn Sie nach einer benutzerdefinierten Hierarchie sortieren möchten, verwenden Sie eine case-Anweisung.

```
Group
-----
Total
Young
MiddleAge
Old
Male
Female
```

Mit einer Grundbestellung `order by` :

```
Select * from MyTable
Order by Group
```

gibt eine alphabetische Sortierung zurück, die nicht immer erwünscht ist:

```
Group
-----
Female
Male
MiddleAge
Old
```

```
Total
Young
```

Hinzufügen einer 'case'-Anweisung, wobei aufsteigende numerische Werte in der Reihenfolge zugewiesen werden, in der Ihre Daten sortiert werden sollen:

```
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
```

gibt Daten in der angegebenen Reihenfolge zurück:

```
Group
-----
Total
Male
Female
Young
MiddleAge
Old
```

Reihen sortieren / sortieren online lesen: https://riptutorial.com/de/sql-server/topic/5332/reihen-sortieren---sortieren

Kapitel 76: Ressourcen-Gouverneur

Bemerkungen

Resource Governor in SQL Server ist eine Funktion, mit der Sie die Ressourcennutzung von verschiedenen Anwendungen und Benutzern verwalten können. Es tritt in Echtzeit durch das Setzen von CPU- und Speichergrenzen auf. Dies hilft zu verhindern, dass ein schwerer Prozess alle Systemressourcen aufzehrt, während zum Beispiel kleinere Aufgaben auf sie warten.

Nur in Enterprise Edition verfügbar

Examples

Lesen der Statistiken

```
select *
from sys.dm_resource_governor_workload_groups

select *
from sys.dm_resource_governor_resource_pools
```

Erstellen Sie einen Pool für Ad-hoc-Abfragen

Erstellen Sie zunächst einen Ressourcenpool neben dem Standardpool

```
CREATE RESOURCE POOL [PoolAdhoc] WITH(min_cpu_percent=0,
        max_cpu_percent=50,
        min_memory_percent=0,
        max_memory_percent=50)
GO
```

Erstellen Sie die Worload-Gruppe für den Pool

```
CREATE WORKLOAD GROUP [AdhocMedium] WITH(importance=Medium) USING [PoolAdhoc]
```

Erstellen Sie die Funktion, die die Logik für den Ressourcen-Governor enthält, und fügen Sie sie an

```
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
```

Ressourcen-Gouverneur online lesen: https://riptutorial.com/de/sql-server/topic/4146/ressourcen-gouverneur

Kapitel 77: Rufen Sie Informationen zu Ihrer Instanz ab

Examples

Lokale und Remote-Server abrufen

So rufen Sie eine Liste aller in der Instanz registrierten Server ab:

```
EXEC sp_helpserver;
```

Erhalten Sie Informationen zu aktuellen Sitzungen und zur Ausführung von Abfragen

```
sp_who2
```

Mit diesem Verfahren können Sie Informationen zu aktuellen SQL Server-Sitzungen finden. Da es sich um eine Prozedur handelt, ist es oft hilfreich, die Ergebnisse in einer temporären Tabelle oder Tabellenvariablen zu speichern, damit die Ergebnisse nach Bedarf sortiert, gefiltert und transformiert werden können.

Das Folgende kann für eine abfragbare Version von `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
```

Beispiele:

```
-- 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
```

Rufen Sie Edition und Version der Instanz ab

```
SELECT    SERVERPROPERTY('ProductVersion') AS ProductVersion,
          SERVERPROPERTY('ProductLevel') AS ProductLevel,
          SERVERPROPERTY('Edition') AS Edition,
          SERVERPROPERTY('EngineEdition') AS EngineEdition;
```

Instanzzeit in Tagen abrufen

```
SELECT  DATEDIFF(DAY, login_time, getdate()) UpDays
FROM    master..sysprocesses
WHERE   spid = 1
```

Informationen zur SQL Server-Version

So ermitteln Sie die SQL Server-Edition, die Produktebene und die Versionsnummer sowie den Namen des Host-Computers und den Servertyp:

```
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;
```

Allgemeine Informationen zu Datenbanken, Tabellen, gespeicherten Prozeduren und deren Suche.

Abfrage zur Suche der zuletzt ausgeführten SPs in db

```
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
```

Abfrage zum Durchsuchen von gespeicherten Prozeduren

```
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
```

Abfrage zum Suchen der Spalte aus allen Tabellen der Datenbank

```
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;
```

Abfrage an, um Wiederherstellungsdetails zu überprüfen

```
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
```

Abfrage, um das Protokoll zu finden

```
select top 100 * from databaselog
Order by Posttime desc
```

Abfrage, um die SPS-Details zu überprüfen

```
SELECT name, create_date, modify_date
FROM sys.objects
WHERE type = 'P'
Order by modify_date desc
```

Rufen Sie Informationen zu Ihrer Instanz ab online lesen: https://riptutorial.com/de/sql-server/topic/2029/rufen-sie-informationen-zu-ihrer-instanz-ab

Kapitel 78: Rufen Sie Informationen zur Datenbank ab

Bemerkungen

Wie bei anderen relationalen Datenbanksystemen stellt SQL Server Metadaten zu Ihren Datenbanken bereit.

Dies wird durch das ISO-Standard-Schema `INFORMATION_SCHEMA` oder die SQL Server-spezifischen `sys` Katalogsichten bereitgestellt.

Examples

Zählen Sie die Anzahl der Tabellen in einer Datenbank

Diese Abfrage gibt die Anzahl der Tabellen in der angegebenen Datenbank zurück.

```
USE YourDatabaseName
SELECT COUNT(*) from INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
```

Nachfolgend können Sie dies für alle Benutzertabellen mit SQL Server 2008+ tun. Die Referenz ist hier .

```
SELECT COUNT(*) FROM sys.tables
```

Rufen Sie eine Liste aller gespeicherten Prozeduren ab

Die folgenden Abfragen geben eine Liste aller gespeicherten Prozeduren in der Datenbank mit grundlegenden Informationen zu jeder gespeicherten Prozedur zurück:

SQL Server 2005

```
SELECT *
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_TYPE = 'PROCEDURE'
```

Die `ROUTINE_NAME` , `ROUTINE_SCHEMA` und `ROUTINE_DEFINITION` sind im Allgemeinen am nützlichsten.

SQL Server 2005

```
SELECT *
FROM sys.objects
WHERE type = 'P'
```

SQL Server 2005

```
SELECT *
FROM sys.procedures
```

Beachten Sie, dass diese Version gegenüber der Auswahl aus sys.objects einen Vorteil hat, da sie die zusätzlichen Spalten `is_auto_executed` , `is_execution_replicated` , `is_repl_serializable` und `skips_repl_constraints` .

SQL Server 2005

```
SELECT *
FROM sysobjects
WHERE type = 'P'
```

Beachten Sie, dass die Ausgabe viele Spalten enthält, die sich niemals auf eine gespeicherte Prozedur beziehen.

Der nächste Satz von Abfragen gibt alle gespeicherten Prozeduren in der Datenbank zurück, die die Zeichenfolge 'SearchTerm' enthalten:

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%'
```

Rufen Sie die Liste aller Datenbanken auf einem Server ab

Methode 1: Die folgende Abfrage gilt für die Version SQL Server 2000+ (Enthält 12 Spalten).

```
SELECT * FROM dbo.sysdatabases
```

Methode 2: Unter Abfrage werden Informationen zu Datenbanken mit weiteren Informationen extrahiert (z. B. Status, Isolation, Wiederherstellungsmodell usw.)

Hinweis: Dies ist eine Katalogansicht und wird als SQL SERVER 2005+ -Version verfügbar sein

```
SELECT * FROM sys.databases
```

Methode 3: Um nur Datenbanknamen anzuzeigen, können Sie die undokumentierte sp_MSForEachDB verwenden

```
EXEC sp_MSForEachDB 'SELECT ''?'' AS DatabaseName'
```

Methode 4: Unterhalb von SP können Sie die Datenbankgröße zusammen mit Datenbankname, Eigentümer, Status usw. auf dem Server angeben

```
EXEC sp_helpdb
```

Methode 5 In ähnlicher Weise gibt die unten stehende gespeicherte Prozedur Datenbanknamen, Datenbankgröße und Anmerkungen an

```
EXEC sp_databases
```

Datenbankdateien

Zeigt alle Datendateien für alle Datenbanken mit Größen- und Wachstumsinformationen an

```
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
```

Datenbankoptionen abrufen

Die folgende Abfrage gibt die Datenbankoptionen und Metadaten zurück:

```
select * from sys.databases WHERE name = 'MyDatabaseName';
```

Größe aller Tabellen in der aktuellen Datenbank anzeigen

```
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
```

Bestimmen Sie den Berechtigungspfad einer Windows-Anmeldung

Daraufhin werden der Benutzertyp und der Berechtigungspfad angezeigt (von welcher Windows-Gruppe der Benutzer seine Berechtigungen bezieht).

```
xp_logininfo 'DOMAIN\user'
```

Abrufen von Tabellen mit bekannter Spalte

Diese Abfrage gibt alle `COLUMNS` und die zugehörigen `TABLES` für einen bestimmten Spaltennamen zurück. Es soll Ihnen zeigen, welche Tabellen (unbekannt) eine angegebene Spalte enthalten (bekannt).

```
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%'
```

Prüfen Sie, ob unternehmensspezifische Funktionen verwendet werden

Manchmal ist es nützlich zu überprüfen, ob Ihre Arbeit an der Developer Edition keine Abhängigkeit von auf die Enterprise Edition beschränkten Funktionen verursacht hat.

Sie können dies mit der `sys.dm_db_persisted_sku_features` wie `sys.dm_db_persisted_sku_features` tun:

```
SELECT * FROM sys.dm_db_persisted_sku_features
```

Gegen die Datenbank selbst.

Wenn möglich, werden die verwendeten Funktionen aufgelistet.

Suchen Sie nach allen Tabellen und Spalten, die einen angegebenen Spaltenwert enthalten, und geben Sie sie zurück

Dieses Skript gibt von hier und hier alle Tabellen und Spalten zurück, in denen ein bestimmter Wert vorhanden ist. Auf diese Weise können Sie herausfinden, wo sich ein bestimmter Wert in einer Datenbank befindet. Es kann eine Belastung sein, daher wird empfohlen, es zuerst in einer Backup- / Testumgebung auszuführen.

```
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
```

Rufen Sie alle Schemata, Tabellen, Spalten und Indizes ab

```
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;
```

Gibt eine Liste von SQL Agent-Jobs mit Zeitplaninformationen zurück

```
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]
```

Rufen Sie Informationen zu Sicherungs- und Wiederherstellungsvorgängen ab

So rufen Sie die Liste aller Sicherungsvorgänge ab, die für die aktuelle Datenbankinstanz ausgeführt wurden:

```
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
```

So rufen Sie die Liste aller für die aktuelle Datenbankinstanz ausgeführten Wiederherstellungsvorgänge ab:

```
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
```

Finden Sie jede Erwähnung eines Feldes in der Datenbank

```
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
```

Erwähnung von `myField` in SProcs, Views usw.

Rufen Sie Informationen zur Datenbank ab online lesen: https://riptutorial.com/de/sql-server/topic/697/rufen-sie-informationen-zur-datenbank-ab

Kapitel 79: Schemas

Examples

Schema erstellen

```
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
```

Schema ändern

```
ALTER SCHEMA dvr
    TRANSFER dbo.tbl_Staging;
GO
```

Dadurch würde die Tabelle tbl_Staging vom dbo-Schema in das dvr-Schema übertragen

Schemas löschen

```
DROP SCHEMA dvr
```

Zweck

Das Schema bezieht sich auf eine bestimmte Datenbanktabelle und deren Beziehung zueinander. Es bietet einen Organisationsplan, wie die Datenbank aufgebaut ist. Weitere Vorteile der Implementierung von Datenbankschemas sind, dass Schemas als Methode verwendet werden können, die den Zugriff auf bestimmte Tabellen in einer Datenbank einschränkt / gewährt.

Schemas online lesen: https://riptutorial.com/de/sql-server/topic/5806/schemas

Kapitel 80: Schlüsselwort löschen

Einführung

Das Drop-Schlüsselwort kann mit verschiedenen SQL-Objekten verwendet werden. In diesem Thema finden Sie kurze Beispiele für die unterschiedliche Verwendung von Datenbankobjekten.

Bemerkungen

Links zu MSDN.

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

Examples

Tabellen ablegen

Mit dem Befehl **DROP TABLE werden** die Tabellendefinitionen und alle Daten, Indizes, Trigger, Einschränkungen und zugehörigen Berechtigungen entfernt.

Bevor Sie eine Tabelle löschen, sollten Sie prüfen, ob Objekte (Ansichten, gespeicherte Prozeduren, andere Tabellen) vorhanden sind, die auf die Tabelle verweisen.

Sie können keine Tabelle löschen, auf die eine andere Tabelle von FOREIGN KEY verweist. Sie müssen zuerst den FOREIGN KEY löschen, der auf ihn verweist.

Sie können eine Tabelle löschen, auf die von einer Sicht oder einer gespeicherten Prozedur verwiesen wird. Nach dem Löschen der Tabelle ist die Sicht oder die gespeicherte Prozedur nicht mehr verwendbar.

Die Syntax

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

- `IF EXISTS` - `IF EXISTS` die Tabelle nur, wenn sie existiert
- `database_name` - Geben Sie den Namen der Datenbank an, in der sich die Tabelle befindet
- `schema_name` - `schema_name` den Namen des Schemas an, unter dem sich die Tabelle befindet
- `table_name` - `table_name` den Namen der zu `table_name` Tabelle an

Beispiele

Entfernen Sie die Tabelle mit dem Namen **TABLE_1** aus der aktuellen Datenbank und dem Standardschema dbo

```
DROP TABLE Table_1;
```

Entfernen Sie die Tabelle mit **TABLE_1** aus der Datenbank **HR** und dem Standardschema dbo

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

Entfernen Sie die Tabelle mit **TABLE_1** aus der Datenbank **HR** und dem Schema **external**

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

Datenbanken löschen

Der Befehl **DROP DATABASE** entfernt einen Datenbankkatalog unabhängig von seinem Status (offline, schreibgeschützt, verdächtig usw.) aus der aktuellen SQL Server-Instanz.

Eine Datenbank kann nicht gelöscht werden, wenn Datenbank-Momentaufnahmen zugeordnet sind, da die Datenbank-Momentaufnahmen zuerst gelöscht werden müssen.

Bei einem Datenbank-Drop werden alle von der Datenbank verwendeten physischen Festplattendateien (sofern sie nicht offline sind) entfernt, sofern Sie nicht die gespeicherte Prozedur 'sp_detach_db' verwenden.

Ein Datenbank-Snapshot-Drop löscht den Snapshot aus der SQL Server-Instanz und löscht die physischen Dateien, die auch von ihm verwendet werden.

Eine abgelegte Datenbank kann nur durch Wiederherstellen einer Sicherung (auch nicht aus einem Datenbank-Snapshot) neu erstellt werden.

Die Syntax

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

- `IF EXISTS` - `IF EXISTS` die Tabelle nur, wenn sie existiert
- `database_name` - Gibt den Namen der zu löschenden Datenbank an
- `database_snapshot_name` - Gibt den zu entfernenden Datenbank-Snapshot an
-

Beispiele

Entfernen Sie eine einzelne Datenbank.

```
DROP DATABASE Database1;
```

Mehrere Datenbanken entfernen

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

Schnappschuss entfernen

```
DROP DATABASE Database1_snapshot17;
```

Entfernen, wenn Datenbank vorhanden ist

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

Temporäre Tabellen löschen

In SQL Server gibt es zwei Arten von temporären Tabellen:

1. ## `##GlobalTempTable` ist eine temporäre Tabelle, die zwischen allen Benutzersitzungen angezeigt wird.
2. `#LocalTempTable` Registerkarte - ist eine Art temporärer Tabelle, die nur im aktuellen Bereich vorhanden ist (nur im tatsächlichen Prozess - Sie können die ID Ihres aktuellen Prozesses über `SELECT @@SPID`)

Der Löschvorgang temporärer Tabellen ist derselbe wie bei normalen Tabellen:

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

VOR SQL Server 2016:

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

SQL Server 2016:

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

Schlüsselwort löschen online lesen: https://riptutorial.com/de/sql-server/topic/9532/schlusselwort-loschen

Kapitel 81: SCOPE_IDENTITY ()

Syntax

- SELECT SCOPE_IDENTITY ();
- SELECT SCOPE_IDENTITY () AS [SCOPE_IDENTITY];
- SCOPE_IDENTITY ()

Examples

Einführung mit einfachem Beispiel

SCOPE_IDENTITY () gibt den letzten Identitätswert zurück, der in eine Identitätsspalte im selben Bereich eingefügt wurde. Ein Bereich ist ein Modul: eine gespeicherte Prozedur, ein Trigger, eine Funktion oder ein Stapel. Daher befinden sich zwei Anweisungen im selben Gültigkeitsbereich, wenn sie sich in derselben gespeicherten Prozedur, Funktion oder im selben Stapel befinden.

INSERT INTO ([column1], [column2]) VALUES (8,9);
GEHEN
SELECT SCOPE_IDENTITY () AS [SCOPE_IDENTITY];
GEHEN

SCOPE_IDENTITY () online lesen: https://riptutorial.com/de/sql-server/topic/5326/scope-identity---

Kapitel 82: Seitennummerierung

Einführung

Zeilenversatz und Paging in verschiedenen Versionen von SQL Server

Syntax

- SELECT * FROM TableName ORDER BY ID OFFSET 10 ROWS FETCH NÄCHSTE 10 ROWS NUR;

Examples

Pagination mit ROW_NUMBER mit einem allgemeinen Tabellenausdruck

SQL Server 2008

Die `ROW_NUMBER` Funktion kann jeder Zeile in einer Ergebnismenge eine inkrementierende Nummer zuweisen. In Kombination mit einem Common Table Expression , der einen `BETWEEN` Operator verwendet, können "Ergebnisseiten" erstellt werden. Beispiel: Seite 1 mit den Ergebnissen 1-10, Seite zwei mit den Ergebnissen 11-20, Seite drei mit den Ergebnissen 21-30 usw.

```
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
```

Anmerkung: Es ist nicht möglich, `ROW_NUMBER` in einer `WHERE` Klausel wie `ROW_NUMBER` zu verwenden:

```
SELECT object_id,
    name,
    type,
    create_date
FROM sys.objects
WHERE ROW_NUMBER() OVER (ORDER BY name) BETWEEN 41 AND 50
```

Obwohl dies bequemer wäre, gibt der SQL Server in diesem Fall den folgenden Fehler zurück:

> Meldung 4108, Ebene 15, Status 1, Zeile 6

Fensterfunktionen können nur in den SELECT- oder ORDER BY-Klauseln angezeigt werden.

Paginierung mit OFFSET FETCH

SQL Server 2012

Die Klausel `OFFSET FETCH` implementiert die Paginierung in `OFFSET FETCH` Weise. Damit können Sie N1-Zeilen (in `OFFSET`) überspringen und die nächsten N2-Zeilen (in `FETCH`) zurückgeben:

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 40 ROWS FETCH NEXT 10 ROWS ONLY
```

Die `ORDER BY` Klausel ist erforderlich, um deterministische Ergebnisse bereitzustellen.

Paginaton mit innerer Frage

In früheren Versionen von SQL Server mussten Entwickler die doppelte Sortierung in Kombination mit dem Schlüsselwort `TOP` , um Zeilen auf einer Seite zurückzugeben:

```
 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
```

Die innere Abfrage gibt die ersten 50 Zeilen nach `name` sortiert zurück. Die äußere Abfrage wird dann die Reihenfolge dieser 50 Zeilen umkehren und die ersten 10 Zeilen auswählen (dies sind die letzten 10 Zeilen in der Gruppe vor der Umkehrung).

Paging in verschiedenen Versionen von 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 mit ORDER BY OFFSET und FETCH NEXT

Um die nächsten 10 Zeilen zu erhalten, führen Sie einfach die folgende Abfrage aus:

```
SELECT * FROM TableName ORDER BY id OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

Wichtige Punkte bei der Verwendung:

- `ORDER BY` ist zwingend erforderlich, um die Klausel `OFFSET` und `FETCH` zu verwenden.
- `OFFSET` Klausel ist bei `FETCH` obligatorisch. Sie können niemals `ORDER BY ... FETCH` .
- `TOP` kann nicht mit `OFFSET` und `FETCH` in demselben Abfrageausdruck kombiniert werden.

Seitennummerierung online lesen: https://riptutorial.com/de/sql-server/topic/6874/seitennummerierung

Kapitel 83: SELECT-Anweisung

Einführung

In SQL geben `SELECT` Anweisungen Ergebnissätze aus Datensammlungen wie Tabellen oder Ansichten zurück. `SELECT` Anweisungen können mit verschiedenen anderen Klauseln wie `WHERE` , `GROUP BY` oder `ORDER BY` , um die gewünschten Ergebnisse weiter zu verfeinern.

Examples

Grundauswahl aus der Tabelle

Wählen Sie alle Spalten aus einer Tabelle aus (Systemtabelle in diesem Fall):

```
SELECT *
FROM sys.objects
```

Oder wählen Sie nur einige bestimmte Spalten aus:

```
SELECT object_id, name, type, create_date
FROM sys.objects
```

Zeilen mit der WHERE-Klausel filtern

Die WHERE-Klausel filtert nur die Zeilen, die bestimmte Bedingungen erfüllen:

```
SELECT *
FROM sys.objects
WHERE type = 'IT'
```

Sortieren Sie die Ergebnisse mit ORDER BY

Die ORDER BY-Klausel sortiert Zeilen in der zurückgegebenen Ergebnismenge nach einer Spalte oder einem Ausdruck:

```
SELECT *
FROM sys.objects
ORDER BY create_date
```

Ergebnis mit GROUP BY zusammenfassen

GROUP BY-Klausel gruppiert Zeilen nach einem bestimmten Wert:

```
SELECT type, count(*) as c
FROM sys.objects
```

```
GROUP BY type
```

Sie können für jede Gruppe eine Funktion anwenden (Aggregatfunktion), um die Summe oder Anzahl der Datensätze in der Gruppe zu berechnen.

Art	c
SQ	3
S	72
ES	16
PK	1
U	5

Filtern Sie Gruppen mithilfe der HAVING-Klausel

Die HAVING-Klausel entfernt Gruppen, die die Bedingung nicht erfüllen:

```
SELECT type, count(*) as c
FROM sys.objects
GROUP BY type
HAVING count(*) < 10
```

Art	c
SQ	3
PK	1
U	5

Nur die ersten N Zeilen zurückgeben

TOP-Klausel gibt nur die ersten N Zeilen im Ergebnis zurück:

```
SELECT TOP 10 *
FROM sys.objects
```

Paginierung mit OFFSET FETCH

Die Klausel OFFSET FETCH ist eine fortgeschrittenere Version von TOP. Sie können N1-Zeilen überspringen und die nächsten N2-Zeilen übernehmen:

```
SELECT *
FROM sys.objects
```

```
ORDER BY object_id
OFFSET 50 ROWS FETCH NEXT 10 ROWS ONLY
```

Sie können OFFSET ohne Abruf verwenden, um nur die ersten 50 Zeilen zu überspringen:

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 50 ROWS
```

SELECT ohne FROM (keine Datenquelle)

SELECT-Anweisung kann ohne FROM-Klausel ausgeführt werden:

```
declare @var int = 17;

SELECT @var as c1, @var + 2 as c2, 'third' as c3
```

In diesem Fall wird eine Zeile mit Werten / Ergebnissen von Ausdrücken zurückgegeben.

SELECT-Anweisung online lesen: https://riptutorial.com/de/sql-server/topic/4662/select-anweisung

Kapitel 84: Sequenzen

Examples

Sequenz erstellen

```
CREATE SEQUENCE [dbo].[CustomersSeq]
AS INT
START WITH 10001
INCREMENT BY 1
MINVALUE -1;
```

Verwenden Sie die Sequenz in der Tabelle

```
CREATE TABLE [dbo].[Customers]
(
    CustomerID INT DEFAULT (NEXT VALUE FOR [dbo].[CustomersSeq]) NOT NULL,
    CustomerName VARCHAR(100),
);
```

Mit Sequenz in die Tabelle einfügen

```
INSERT INTO [dbo].[Customers]
       ([CustomerName])
 VALUES
       ('Jerry'),
       ('Gorge')

SELECT * FROM [dbo].[Customers]
```

Ergebnisse

Kundennummer	Kundenname
10001	Jerry
10002	Schlucht

Aus löschen und neu einfügen

```
DELETE FROM [dbo].[Customers]
WHERE CustomerName = 'Gorge';

INSERT INTO [dbo].[Customers]
       ([CustomerName])
 VALUES ('George')

SELECT * FROM [dbo].[Customers]
```

Ergebnisse

Kundennummer	Kundenname
10001	Jerry
10003	George

Sequenzen online lesen: https://riptutorial.com/de/sql-server/topic/5324/sequenzen

Kapitel 85: Sicherheit auf Zeilenebene

Examples

RLS-Filterprädikat

SQL Server 2016+ und Azure SQL-Datenbank können Sie Zeilen, die in der select-Anweisung zurückgegeben werden, mithilfe eines Prädikats automatisch filtern. Diese Funktion wird als **Sicherheit auf Zeilenebene bezeichnet** .

Zunächst benötigen Sie eine Tabellenwertfunktion, die ein Prädikat enthält, das die Bedingung beschreibt, unter der Benutzer Daten aus einer Tabelle lesen können:

```
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

)
```

In diesem Beispiel sagt das Prädikat aus, dass nur Benutzer, die einen Wert in SESSION_CONTEXT haben, der mit dem Eingabeargument übereinstimmt, auf das Unternehmen zugreifen können. Sie können eine beliebige andere Bedingung festlegen, z. B. die Datenbankrolle oder die Datenbank-ID des aktuellen Benutzers usw.

> Der größte Teil des obigen Codes ist eine Vorlage, die Sie kopieren und einfügen. Das einzige, was sich hier ändern wird, sind der Name und die Argumente des Prädikats und der Bedingung in der WHERE-Klausel. Jetzt erstellen Sie eine Sicherheitsrichtlinie, die dieses Prädikat auf einige Tabellen anwendet.

Jetzt können Sie eine Sicherheitsrichtlinie erstellen, die das Prädikat auf einige Tabellen anwendet:

```
CREATE SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany(CompanyID) ON dbo.Company
    WITH (State=ON)
```

Diese Sicherheitsrichtlinie weist der Firmentabelle ein Prädikat zu. Immer wenn jemand versucht, Daten aus der Company-Tabelle zu lesen, wendet die Sicherheitsrichtlinie in jeder Zeile ein Prädikat an, übergibt die Spalte CompanyID als Parameter des Prädikats. Das Prädikat wird ausgewertet, falls diese Zeile im Ergebnis der SELECT-Abfrage zurückgegeben wird.

RLS-Sicherheitsrichtlinie ändern

Die Sicherheitsrichtlinie ist eine Gruppe von Prädikaten, die Tabellen zugeordnet sind, die gemeinsam verwaltet werden können. Sie können Vergleichselemente hinzufügen, entfernen oder die gesamte Richtlinie aktivieren / deaktivieren.

Sie können in der vorhandenen Sicherheitsrichtlinie weitere Vergleichselemente für Tabellen hinzufügen.

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany(CompanyID) ON dbo.Company
```

Sie können einige Prädikate aus der Sicherheitsrichtlinie entfernen:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    DROP FILTER PREDICATE ON dbo.Company
```

Sie können die Sicherheitsrichtlinie deaktivieren

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = OFF );
```

Sie können die deaktivierte Sicherheitsrichtlinie aktivieren:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = ON );
```

Aktualisieren mit RLS-Blockprädikat verhindern

Mithilfe der Sicherheit auf Zeilenebene können Sie einige Prädikate definieren, die steuern, wer Zeilen in der Tabelle aktualisieren kann. Zuerst müssen Sie eine Tabellenwertfunktion definieren, die das Prädikat darstellt, das die Zugriffsrichtlinie steuert.

FUNKTION ERSTELLEN

dbo.pUserCanAccessProduct (@CompanyID int)

```
RETURNS TABLE
WITH SCHEMABINDING
```

AS RETURN (SELECT 1 als canAccess WHERE)

CAST (SESSION_CONTEXT (N'CompanyID ') als int) = @CompanyID

) In diesem Beispiel sagt das Prädikat aus, dass nur Benutzer, die einen Wert in SESSION_CONTEXT haben, der mit dem Eingabeargument übereinstimmt, auf das Unternehmen zugreifen können. Sie können eine beliebige andere Bedingung festlegen, z. B. die Datenbankrolle oder die Datenbank-ID des aktuellen Benutzers usw.

Der größte Teil des obigen Codes ist eine Vorlage, die Sie kopieren und einfügen. Das

einzige, was sich hier ändern wird, sind der Name und die Argumente des Prädikats und der Bedingung in der WHERE-Klausel. Jetzt erstellen Sie eine Sicherheitsrichtlinie, die dieses Prädikat auf einige Tabellen anwendet.

Jetzt können wir Sicherheitsrichtlinien mit dem Prädikat erstellen, die Aktualisierungen der Produkttabelle blockieren, wenn die Spalte CompanyID in der Tabelle das Prädikat nicht erfüllt.

SICHERHEITSPOLITIK ERSTELLEN dbo.ProductAccessPolicy ADD BLOCK PREDICATE dbo.pUserCanAccessProduct (CompanyID) ON dbo.Product

Dieses Prädikat wird auf alle Operationen angewendet. Wenn Sie ein Prädikat auf eine Operation anwenden möchten, können Sie Folgendes schreiben:

SICHERHEITSPOLITIK ERSTELLEN dbo.ProductAccessPolicy ADD BLOCK PREDICATE dbo.pUserCanAccessProduct (CompanyID) ON dbo.Product AFTER INSERT

Mögliche Optionen, die Sie nach der Definition des Blockprädikats hinzufügen können, sind:

[{AFTER {INSERT | UPDATE}}
| {VORNEHMEN {UPDATE | LÖSCHEN}}]

Sicherheit auf Zeilenebene online lesen: https://riptutorial.com/de/sql-server/topic/7045/sicherheit-auf-zeilenebene

Kapitel 86: SORTIEREN NACH

Bemerkungen

Der Zweck der ORDER BY-Klausel besteht darin, die von einer Abfrage zurückgegebenen Daten zu sortieren.

Beachten Sie, dass die **Reihenfolge der von einer Abfrage zurückgegebenen Zeilen undefiniert ist, es sei denn, es gibt eine ORDER BY-Klausel.**

Ausführliche Informationen zur ORDER BY-Klausel finden Sie in der MSDN-Dokumentation: https://msdn.microsoft.com/en-us/library/ms188385.aspx

Examples

Einfache ORDER BY-Klausel

Im Folgenden wird anhand der Employees-Tabelle die Spalte Id, FName und LName in (aufsteigender) LName-Reihenfolge zurückgegeben:

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

Kehrt zurück:

Ich würde	FName	LName
2	John	Johnson
1	James	Schmied
4	Johnathon	Schmied
3	Michael	Williams

Um in absteigender Reihenfolge zu sortieren, fügen Sie das DESC-Schlüsselwort nach dem Feldparameter hinzu, z. B. dieselbe Abfrage in absteigender LName-Reihenfolge:

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

ORDER BY mehrere Felder

Für die `ORDER BY` Klausel können mehrere Felder angegeben werden, entweder in ASCending- oder DESCending-Reihenfolge.

Mithilfe der Tabelle http://stackoverflow.com/documentation/sql/280/example-databases/1207/item-sales-table#t=20160721131406643421 1 können wir beispielsweise eine Abfrage zurückgeben, die nach SaleDate in aufsteigender Reihenfolge sortiert wird, und Menge in absteigender Reihenfolge.

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

Beachten Sie, dass das `ASC` Schlüsselwort optional ist und die Ergebnisse standardmäßig in aufsteigender Reihenfolge eines bestimmten Felds sortiert werden.

ORDER BY mit komplexer Logik

Wenn wir die Daten für jede Gruppe anders bestellen möchten, können wir dem `ORDER BY` eine `CASE` Syntax hinzufügen. In diesem Beispiel möchten wir Mitarbeiter aus Abteilung 1 nach Nachnamen und Mitarbeiter aus Abteilung 2 nach Gehalt bestellen.

Ich würde	FName	LName	Telefonnummer	ManagerId	DepartmentId	Gehalt	Anstellun
1	James	Schmied	1234567890	NULL	1	1000	01-01-200
2	John	Johnson	2468101214	1	1	400	23-03-200
3	Michael	Williams	1357911131	1	2	600	12-05-200
4	Johnathon	Schmied	1212121212	2	1	500	24-07-201
5	Sam	Sächsisch	1372141312	2	2	400	25-03-201

```
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
```

Kundenspezifische Bestellung

Wenn Sie nach einer Spalte in einer anderen Reihenfolge als in alphabetischer / numerischer Reihenfolge bestellen möchten, können Sie mit `case` die gewünschte Reihenfolge angeben.

`order by Group` :

Gruppe	Anzahl
Nicht im Ruhestand	6
Im Ruhestand	4
Gesamt	10

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

Gruppe	Anzahl
Gesamt	10
Im Ruhestand	4
Nicht im Ruhestand	6

SORTIEREN NACH online lesen: https://riptutorial.com/de/sql-server/topic/4149/sortieren-nach

Kapitel 87: Speichern von JSON in SQL-Tabellen

Examples

JSON als Textspalte gespeichert

JSON hat ein Textformat und wird daher in NVARCHAR-Standardspalten gespeichert. Die NoSQL-Sammlung entspricht einer zweispaltigen Tabelle mit Schlüsselwerten:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max)
)
```

Verwenden Sie `nvarchar(max)` da Sie nicht sicher sind, wie groß Ihre JSON-Dokumente sein sollen. `nvarchar(4000)` und `varchar(8000)` eine bessere Leistung, jedoch mit einer Größenbeschränkung von 8 KB.

Stellen Sie sicher, dass JSON mit ISJSON ordnungsgemäß formatiert ist

Da JSON in einer Textspalte gespeichert ist, möchten Sie möglicherweise sicherstellen, dass diese ordnungsgemäß formatiert ist. Sie können eine CHECK-Einschränkung für die JSON-Spalte hinzufügen, die prüft, ob der Text richtig formatiert ist.

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max)
       CONSTRAINT [Data should be formatted as JSON]
       CHECK (ISJSON(Data) > 0)
)
```

Wenn Sie bereits über eine Tabelle verfügen, können Sie mit der Anweisung ALTER TABLE eine Prüfbedingung hinzufügen:

```
ALTER TABLE ProductCollection
    ADD CONSTRAINT [Data should be formatted as JSON]
        CHECK (ISJSON(Data) > 0)
```

Zeigen Sie Werte aus JSON-Text als berechnete Spalten an

Sie können Werte aus der JSON-Spalte als berechnete Spalten verfügbar machen:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max),
  Price AS JSON_VALUE(Data, '$.Price'),
```

```
  Color JSON_VALUE(Data, '$.Color') PERSISTED
)
```

Wenn Sie die berechnete Spalte PERSISTED hinzufügen, wird der Wert aus dem JSON-Text in dieser Spalte angezeigt. Auf diese Weise können Ihre Abfragen Werte schneller aus JSON-Text lesen, da keine Analyse erforderlich ist. Bei jeder Änderung von JSON in dieser Zeile wird der Wert neu berechnet.

Index für JSON-Pfad hinzufügen

Abfragen, die Daten nach einem bestimmten Wert in der JSON-Spalte filtern oder sortieren, verwenden normalerweise die vollständige Tabellensuche.

```
SELECT * FROM ProductCollection
WHERE JSON_VALUE(Data, '$.Color') = 'Black'
```

Um diese Art von Abfragen zu optimieren, können Sie eine nicht persistierte berechnete Spalte hinzufügen, die den in Filter oder Sortierung verwendeten JSON-Ausdruck (in diesem Beispiel JSON_VALUE (Data, '$ .Color')) bereitstellt, und einen Index für diese Spalte erstellen:

```
ALTER TABLE ProductCollection
ADD vColor as JSON_VALUE(Data, '$.Color')

CREATE INDEX idx_JsonColor
ON ProductCollection(vColor)
```

Abfragen verwenden den Index anstelle der einfachen Tabellensuche.

JSON in In-Memory-Tabellen gespeichert

Wenn Sie speicheroptimierte Tabellen verwenden können, können Sie JSON als Text speichern:

```
CREATE TABLE ProductCollection (
  Id int identity primary key nonclustered,
  Data nvarchar(max)
) WITH (MEMORY_OPTIMIZED=ON)
```

Vorteile von JSON im In-Memory:

- JSON-Daten befinden sich immer im Speicher, sodass kein Zugriff auf die Festplatte möglich ist
- Es gibt keine Sperren und Verriegelungen während der Arbeit mit JSON

Speichern von JSON in SQL-Tabellen online lesen: https://riptutorial.com/de/sql-server/topic/5029/speichern-von-json-in-sql-tabellen

Kapitel 88: Split-String-Funktion in SQL Server

Examples

Teilen Sie einen String in SQL Server 2016

In **SQL Server 2016 wurde** schließlich die Split-String-Funktion `STRING_SPLIT`

Parameter: Es akzeptiert zwei Parameter

String :

> Ist ein Ausdruck eines beliebigen Zeichentyps (z. B. nvarchar, varchar, nchar oder char).

Trennzeichen :

> Ein einzelner Zeichenausdruck eines beliebigen Zeichentyps (z. B. nvarchar (1), varchar (1), nchar (1) oder char (1)), der als Trennzeichen für verkettete Zeichenfolgen verwendet wird.

Hinweis: Sie sollten immer überprüfen, ob der Ausdruck eine nicht leere Zeichenfolge ist.

Beispiel:

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

Im obigen Beispiel

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

Ergebnis:

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

Wenn es sich um eine leere Zeichenfolge handelt:

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

Ergebnis:

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

Sie können die obige Situation vermeiden, indem Sie eine `WHERE` Klausel hinzufügen

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

Zeichenfolge in SQL Server 2008/2012/2014 unter Verwendung von XML aufteilen

Da es keine `STRING_SPLIT` Funktion gibt, müssen Sie XML-Hack verwenden, um die Zeichenfolge in Zeilen aufzuteilen:

Beispiel:

```
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);
```

Ergebnis:

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

T-SQL-Tabellenvariable und 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
```

Split-String-Funktion in SQL Server online lesen: https://riptutorial.com/de/sql-server/topic/3713/split-string-funktion-in-sql-server

Kapitel 89: SQL Server Evolution durch verschiedene Versionen (2000 - 2016)

Einführung

Ich benutze SQL Server seit 2004. Ich habe mit 2000 angefangen und jetzt werde ich SQL Server 2016 verwenden. Ich habe Tabellen, Ansichten, Funktionen, Trigger, gespeicherte Prozeduren erstellt und viele SQL-Abfragen geschrieben, aber ich habe nicht viele neue Funktionen aus den nachfolgenden verwendet Versionen. Ich habe gegoogelt, aber leider habe ich nicht alle Funktionen an einem Ort gefunden. Also habe ich diese Informationen aus verschiedenen Quellen gesammelt und validiert und hier eingefügt. Ich füge nur die Informationen auf hoher Ebene für alle Versionen von 2000 bis 20 hinzu

Examples

SQL Server Version 2000 - 2016

Die folgenden Features wurden in SQL Server 2000 aus der vorherigen Version hinzugefügt:

1. Neue Datentypen wurden hinzugefügt (BIGINT, SQL_VARIANT, TABLE)
2. Anstelle von und für Trigger wurden die DDL weiterentwickelt.
3. Kaskadierung der referentiellen Integrität
4. XML-Unterstützung
5. Benutzerdefinierte Funktionen und Partitionsansichten.
6. Indizierte Ansichten (Index für Ansichten mit berechneten Spalten zulassen).

Die folgenden Funktionen wurden in Version 2005 von der vorherigen Version hinzugefügt:

1. Erweiterung in TOP-Klausel mit der Option „WITH TIES“.
2. Datenmanipulationsbefehle (DML) und OUTPUT-Klausel zum Abrufen von INSERTED- und DELETED-Werten
3. Die PIVOT- und UNPIVOT-Operatoren.
4. Ausnahmebehandlung mit TRY / CATCH-Block
5. Ranking-Funktionen
6. Häufige Tabellenausdrücke (CTE)
7. Common Language Runtime (Integration von .NET-Sprachen zum Erstellen von Objekten wie gespeicherte Prozeduren, Trigger, Funktionen usw.)
8. Service Broker (Behandlung von Nachrichten zwischen einem Sender und einem Empfänger auf lose gekoppelte Weise)
9. Datenverschlüsselung (native Funktionen zur Unterstützung der Verschlüsselung von Daten, die in benutzerdefinierten Datenbanken gespeichert sind)
10. SMTP-Mail
11. HTTP-Endpunkte (Erstellung von Endpunkten mit einer einfachen T-SQL-Anweisung, die ein

Objekt verfügbar macht, auf das über das Internet zugegriffen werden kann)
12. Multiple Active Results Sets (MARS). Dadurch kann eine persistente Datenbankverbindung von einem einzelnen Client aus mehr als eine aktive Anforderung pro Verbindung haben.
13. SQL Server Integration Services (Wird als primäres ETL-Tool (Extraktion, Transformation und Laden) verwendet
14. Verbesserungen in Analysis Services und Reporting Services.
15. Partitionierung von Tabellen und Indizes. Ermöglicht die Partitionierung von Tabellen und Indizes basierend auf Partitionsgrenzen, wie durch eine PARTITION-FUNKTION angegeben, wobei einzelne Partitionen über ein PARTITIONEN-SCHEMA Dateigruppen zugeordnet werden.

Die folgenden Funktionen wurden in Version 2008 von der vorherigen Version hinzugefügt:

1. Erweiterung bestehender Datentypen DATE und TIME
2. Neue Funktionen wie - SYSUTCDATETIME () und SYSDATETIMEOFFSET ()
3. Ersatzspalten - Zum Speichern von Speicherplatz auf der Festplatte.
4. Große benutzerdefinierte Typen (bis zu 2 GB)
5. Einführung einer neuen Funktion zum Übergeben eines Tabellendatentyps an gespeicherte Prozeduren und Funktionen
6. Neuer Befehl MERGE für die Operationen INSERT, UPDATE und DELETE
7. Neuer HierarchyID-Datentyp
8. Räumliche Datentypen - Zur Darstellung der physischen Position und Form jedes geometrischen Objekts.
9. Schnellere Abfragen und Berichterstellung mit GROUPING SETS - Eine Erweiterung der GROUP BY-Klausel.
10. Erweiterung der Speicheroption FILESTREAM

Die folgenden Funktionen wurden in Version 2008 R2 aus der vorherigen Version hinzugefügt:

1. PowerPivot - Für die Verarbeitung großer Datenmengen.
2. Berichts-Generator 3.0
3. Wolke bereit
4. StreamInsight
5. Stammdatendienste
6. SharePoint-Integration
7. DACPAC (Datenschicht-Anwendungskomponentenpakete)
8. Verbesserung anderer Features von SQL Server 2008

Die folgenden Funktionen wurden in Version 2012 aus der vorherigen Version hinzugefügt:

1. Spaltenspeicherindizes - reduziert die E / A- und Speicherauslastung bei großen Abfragen.
2. Paginierung - Die Paginierung kann mit den Befehlen "OFFSET" und "FETCH" durchgeführt werden.
3. Enthaltene Datenbank - Große Funktion für periodische Datenmigrationen.
4. AlwaysOn-Verfügbarkeitsgruppen
5. Windows Server Core-Unterstützung
6. Benutzerdefinierte Serverrollen

7. Big Data-Unterstützung
8. PowerView
9. SQL Azure-Verbesserungen
10. Tabellarisches Modell (SSAS)
11. DQS-Datenqualitätsdienste
12. File Table - eine Erweiterung der FILESTREAM-Funktion, die 2008 eingeführt wurde.
13. Verbesserung der Fehlerbehandlung einschließlich der THROW-Anweisung
14. Verbesserung des SQL Server Management Studio-Debugs a. SQL Server 2012 führt weitere Optionen zum Steuern von Haltepunkten ein. b. Verbesserungen an Debug-Modusfenstern
    c. Verbesserung in IntelliSense - wie das Einfügen von Code-Snippets.

Die folgenden Funktionen wurden in Version 2014 aus der vorherigen Version hinzugefügt:

1. In-Memory-OLTP-Engine - Verbessert die Leistung um das 20-fache.
2. AlwaysOn-Verbesserungen
3. Pufferpool-Erweiterung
4. Hybrid Cloud-Funktionen
5. Erweiterung in Spaltenspeicherindizes (wie etwa aktualisierbare Spaltenspeicherindizes)
6. Verbesserte Abfrageverarbeitung (wie paralleles SELECT INTO)
7. Power BI für Office 365-Integration
8. Verzögerte Haltbarkeit
9. Verbesserungen für Datenbanksicherungen

Die folgenden Funktionen wurden in Version 2016 aus der vorherigen Version hinzugefügt:

1. Always Encrypted (Immer verschlüsselt) - Always Encrypted (Immer verschlüsselt) schützt Daten in Ruhe oder in Bewegung.
2. Echtzeit-Betriebsanalysen
3. PolyBase in SQL Server
4. Native JSON-Unterstützung
5. Abfragespeicher
6. Verbesserungen für AlwaysOn
7. Erweitertes In-Memory-OLTP
8. Mehrere TempDB-Datenbankdateien
9. Stretch-Datenbank
10. Sicherheit auf Zeilenebene
11. In-Memory-Verbesserungen

    T-SQL-Verbesserungen oder neue Ergänzungen in SQL Server 2016

1. TRUNCATE TABLE mit PARTITION

2. DROP WENN EXISTS

3. STRING_SPLIT- und STRING_ESCAPE-Funktionen

4. ALTER TABLE kann jetzt viele Spalten ändern, während die Tabelle online bleibt, wobei

WITH (ONLINE = ON | OFF) verwendet wird.

5. MAXDOP für DBCC CHECKDB, DBCC CHECKTABLE und DBCC CHECKFILEGROUP
6. ALTER DATABASE SET AUTOGROW_SINGLE_FILE
7. ALTER DATABASE SET AUTOGROW_ALL_FILES
8. COMPRESS- und DECOMPRESS-Funktionen
9. FORMATMESSAGE-Anweisung
10. 2016 führt 8 weitere Eigenschaften mit SERVERPROPERTY ein

ein. InstanceDefaultDataPath

b. InstanceDefaultLogPath

c. ProductBuild

d. ProductBuildType

e. ProductMajorVersion

f. ProductMinorVersion

G. ProductUpdateLevel

h. ProductUpdateReference

SQL Server Evolution durch verschiedene Versionen (2000 - 2016) online lesen: https://riptutorial.com/de/sql-server/topic/10129/sql-server-evolution-durch-verschiedene-versionen--2000---2016-

Kapitel 90: SQL Server Management Studio (SSMS)

Einführung

SQL Server Management Studio (SSMS) ist ein Tool zum Verwalten und Verwalten von SQL Server und der SQL-Datenbank.

SSMS wird von Microsoft kostenlos angeboten.

SSMS-Dokumentation ist verfügbar.

Examples

Den IntelliSense-Cache aktualisieren

Wenn Objekte erstellt oder geändert werden, sind sie nicht automatisch für IntelliSense verfügbar. Um sie für IntelliSense verfügbar zu machen, muss der lokale Cache aktualisiert werden.

Drücken Sie innerhalb eines Abfrage-Editor-Fensters entweder `Ctrl + Shift + R` oder wählen Sie `Edit | IntelliSense | Refresh Local Cache` über das Menü.

Danach sind alle Änderungen seit der letzten Aktualisierung für IntelliSense verfügbar.

SQL Server Management Studio (SSMS) online lesen: https://riptutorial.com/de/sql-server/topic/10642/sql-server-management-studio--ssms-

Kapitel 91: SQL Server unter Windows installieren

Examples

Einführung

Dies sind die verfügbaren Editionen von SQL Server, wie aus der Editionsmatrix hervorgeht :

- Express: Kostenlose Einstiegsdatenbank. Enthält Core-RDBMS-Funktionalität. Begrenzt auf 10G Festplattengröße. Ideal zum Entwickeln und Testen.
- Standard Edition: Lizenzierte Standard Edition. Enthält Kernfunktionen und Business Intelligence-Funktionen.
- Enterprise Edition: Voll funktionsfähige SQL Server Edition. Enthält erweiterte Sicherheits- und Data Warehousing-Funktionen.
- Developer Edition: Enthält alle Funktionen der Enterprise Edition und keine Einschränkungen. Der Download und die Verwendung ist nur für Entwicklungszwecke kostenlos .

Nach dem Download / Erwerb von SQL Server wird die Installation mit SQLSetup.exe ausgeführt, das als GUI oder Befehlszeilenprogramm verfügbar ist.

Bei der Installation über eines dieser beiden Programme müssen Sie einen Product Key angeben und eine Erstkonfiguration durchführen, die die Aktivierung von Funktionen und separaten Services sowie das Festlegen der Anfangsparameter für jeden von ihnen umfasst. Zusätzliche Dienste und Funktionen können jederzeit aktiviert werden, indem Sie das Programm SQLSetup.exe in der Befehlszeilen- oder GUI-Version ausführen.

SQL Server unter Windows installieren online lesen: https://riptutorial.com/de/sql-server/topic/5801/sql-server-unter-windows-installieren

Kapitel 92: SQLCMD

Bemerkungen

Sie müssen sich entweder in dem Pfad befinden, in dem SQLCMD.exe vorhanden ist, oder es Ihrer Umgebungsvariablen PATH hinzufügen.

Examples

SQLCMD.exe wurde über eine Batchdatei oder Befehlszeile aufgerufen

```
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"
```

Batchdateien wie diese können zum Automatisieren von Aufgaben verwendet werden, z. B. zum Erstellen von Sicherungen von Datenbanken zu einem bestimmten Zeitpunkt (kann mit dem Taskplaner geplant werden) für eine SQL Server Express-Version, für die Agentenaufträge nicht verwendet werden können.

SQLCMD online lesen: https://riptutorial.com/de/sql-server/topic/5396/sqlcmd

Kapitel 93: String Aggregatfunktionen in SQL Server

Examples

Verwendung von STUFF für die String-Aggregation

Wir haben einen Studententisch mit SubjectId. Hier ist die Anforderung die Verkettung basierend auf subjectId.

Alle SQL Server-Versionen

```
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 für die String-Aggregation

Im Falle von SQL Server 2017 oder vnext können wir für diese Aggregation den eingebauten STRING_AGG verwenden. Für denselben Schülertisch

```
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
```

String Aggregatfunktionen in SQL Server online lesen: https://riptutorial.com/de/sql-server/topic/9892/string-aggregatfunktionen-in-sql-server

Kapitel 94: String-Funktionen

Bemerkungen

Liste der Stringfunktionen (alphabetisch sortiert):

- ASCII
- Verkohlen
- Charindex
- Concat
- Unterschied
- Format
- Links
- Len
- Niedriger
- Ltrim
- Nchar
- Patindex
- Quotename
- Ersetzen
- Replizieren
- Umkehren
- Recht
- Rtrim
- Soundex
- Platz
- Str
- String_escape

- String_split
- Zeug
- Unterstring
- Unicode
- Oberer, höher

Examples

Links

Gibt eine untergeordnete Zeichenfolge zurück, die mit dem ganz linken Zeichen einer Zeichenfolge beginnt und bis zur angegebenen maximalen Länge reicht.

Parameter:

1. Zeichenausdruck. Der Zeichenausdruck kann einen beliebigen Datentyp haben, der implizit in `varchar` oder `nvarchar` konvertiert werden `nvarchar` , mit Ausnahme von `text` oder `ntext`
2. maximale Länge. Eine ganze Zahl zwischen 0 und `bigint` Maximalwert (9.223.372.036.854.775.807).
   Wenn der Parameter für die maximale Länge negativ ist, wird ein Fehler ausgegeben.

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

Wenn die maximale Länge mehr als die Anzahl der Zeichen in der Zeichenfolge ist, wird die Entier-Zeichenfolge zurückgegeben.

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

Recht

Gibt eine Unterzeichenfolge mit der angegebenen maximalen Länge zurück, die den rechten Teil der Zeichenfolge darstellt.

Parameter:

1. Zeichenausdruck. Der Zeichenausdruck kann einen beliebigen Datentyp haben, der implizit in `varchar` oder `nvarchar` konvertiert werden `nvarchar` , mit Ausnahme von `text` oder `ntext`
2. maximale Länge. Eine ganze Zahl zwischen 0 und `bigint` Maximalwert (9.223.372.036.854.775.807). Wenn der Parameter für die maximale Länge negativ ist, wird ein Fehler ausgegeben.

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

Wenn die maximale Länge mehr als die Anzahl der Zeichen in der Zeichenfolge ist, wird die

Entier-Zeichenfolge zurückgegeben.

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

Unterstring

Gibt einen Teilstring zurück, der mit dem Zeichen beginnt, das sich im angegebenen Startindex befindet, und der angegebenen maximalen Länge.

Parameter:

1. Zeichenausdruck Der Zeichenausdruck kann einen beliebigen Datentyp haben, der implizit in `varchar` oder `nvarchar` konvertiert werden `nvarchar` , mit Ausnahme von `text` oder `ntext` .
2. Startindex. Eine Zahl (`int` oder `bigint`), die den `bigint` des angeforderten Teilstrings angibt. (**Hinweis:** Zeichenfolgen in SQL Server sind Basis-1-Index, dh das erste Zeichen der Zeichenfolge ist Index 1). Diese Anzahl kann kleiner als 1 sein. Wenn die Summe aus Startindex und maximaler Länge größer als 0 ist, wäre der Rückgabestring ein String, der mit dem ersten Zeichen des Zeichenausdrucks beginnt und die Länge von (Startindex) hat + maximale Länge - 1). Wenn es weniger als 0 ist, wird eine leere Zeichenfolge zurückgegeben.
3. Maximale Länge. Eine ganze Zahl zwischen 0 und `bigint` Maximalwert (9.223.372.036.854.775.807). Wenn der Parameter für die maximale Länge negativ ist, wird ein Fehler ausgegeben.

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

Wenn die maximale Länge + Startindex mehr als die Anzahl der Zeichen in der Zeichenfolge ist, wird die Entierzeichenfolge zurückgegeben.

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

Wenn der Startindex größer als die Anzahl der Zeichen in der Zeichenfolge ist, wird eine leere Zeichenfolge zurückgegeben.

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

ASCII

Gibt einen int-Wert zurück, der den ASCII-Code des ganz linken Zeichens einer Zeichenfolge darstellt.

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

Wenn die Zeichenfolge Unicode ist und das äußerste linke Zeichen kein ASCII-Zeichen ist, sondern in der aktuellen Kollatierung darstellbar ist, kann ein Wert größer als 127 zurückgegeben

werden:

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

Wenn die Zeichenfolge Unicode ist und das Zeichen ganz links in der aktuellen Sortierung nicht dargestellt werden kann, wird der int-Wert von 63 zurückgegeben: (der das Fragezeichen in ASCII darstellt):

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

CharIndex

Gibt den Startindex des ersten Vorkommens eines Zeichenfolgenausdrucks in einem anderen Zeichenfolgenausdruck zurück.

Parameterliste:

1. Zeichenfolge zu finden (bis zu 8000 Zeichen)
2. Zeichenfolge für die Suche (beliebiger gültiger Zeichendatentyp und Länge, einschließlich binärer Zeichen)
3. (Optional) Index zum Starten. Eine Anzahl von int oder big int. Bei Auslassung oder weniger als 1 beginnt die Suche am Anfang der Zeichenfolge.

Wenn der zu suchende String `varchar(max)` , `nvarchar(max)` oder `varbinary(max)` , gibt die `CHARINDEX` Funktion einen `bigint` Wert zurück. Andernfalls wird ein `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
```

Verkohlen

Gibt ein Zeichen zurück, das durch einen int-ASCII-Code dargestellt wird.

```
SELECT CHAR(116) -- Returns 't'
SELECT CHAR(84)  -- Returns 'T'
```

Dies kann verwendet werden, um neue Zeilen- / Zeilenvorschub- `CHAR(10)` , Wagenrücklauf- `CHAR(13)` usw. einzuführen . Siehe AsciiTable.com .

Wenn der Argumentwert nicht zwischen 0 und 255 liegt, gibt die CHAR-Funktion `NULL` .
Der Rückgabedatentyp der `CHAR` Funktion ist `char(1)`

Len

Gibt die Anzahl der Zeichen einer Zeichenfolge zurück.

Hinweis: Die `LEN` Funktion ignoriert nachfolgende Leerzeichen:

```
SELECT LEN('My string'), -- returns 9
       LEN('My string   '), -- returns 9
       LEN('   My string') -- returns 12
```

Wenn die Länge einschließlich der nachgestellten Leerzeichen gewünscht wird, gibt es mehrere Techniken, um dies zu erreichen, obwohl jede ihre Nachteile hat. Eine Technik besteht darin, ein einzelnes Zeichen an die Zeichenfolge anzuhängen und dann die `LEN` Zeichen minus eins zu verwenden:

```
DECLARE @str varchar(100) = 'My string   '
SELECT LEN(@str + 'x') - 1 -- returns 12
```

Der Nachteil dabei ist, wenn der Typ der Zeichenfolgenvariablen oder -spalte die maximale Länge hat, das Anfügen des zusätzlichen Zeichens verworfen wird und die resultierende Länge nachstehende Leerzeichen nicht zählt. Um dies zu beheben, löst die folgende modifizierte Version das Problem und gibt in allen Fällen die korrekten Ergebnisse auf Kosten einer geringen zusätzlichen Ausführungszeit und aus diesem Grund (korrekte Ergebnisse einschließlich mit Ersatzpaaren und angemessene Ausführungsgeschwindigkeit). scheint die beste Technik zu sein:

```
SELECT LEN(CONVERT(NVARCHAR(MAX), @str) + 'x') - 1
```

Eine andere Technik ist die Verwendung der `DATALENGTH` Funktion.

```
DECLARE @str varchar(100) = 'My string   '
SELECT DATALENGTH(@str) -- returns 12
```

Beachten Sie jedoch, dass `DATALENGTH` die Länge der Zeichenfolge im Speicher in Byte zurückgibt. Dies wird für `varchar` und `nvarchar` .

```
DECLARE @str nvarchar(100) = 'My string   '
SELECT DATALENGTH(@str) -- returns 24
```

Sie können dies anpassen, indem Sie die Datenlänge der Zeichenfolge durch die Datenlänge eines einzelnen Zeichens (das vom selben Typ sein muss) dividieren. Das folgende Beispiel führt dies aus und behandelt auch den Fall, in dem die Zielzeichenfolge leer ist, wodurch eine Division durch Null vermieden wird.

```
DECLARE @str nvarchar(100) = 'My string   '
SELECT DATALENGTH(@str) / DATALENGTH(LEFT(LEFT(@str, 1) + 'x', 1)) -- returns 12
```

Auch dies hat jedoch ein Problem in SQL Server 2012 und höher. Wenn die Zeichenfolge Ersatzpaare enthält, führt dies zu falschen Ergebnissen (einige Zeichen können mehr Bytes als andere Zeichen in derselben Zeichenfolge belegen).

Eine andere Technik besteht darin, mit `REPLACE` Leerzeichen in ein Zeichen ohne Leerzeichen umzuwandeln und die `LEN` des Ergebnisses zu erhalten. Dies führt in allen Fällen zu korrekten

Ergebnissen, hat aber bei langen Zeichenketten eine sehr schlechte Ausführungsgeschwindigkeit.

Concat

SQL Server 2012

Gibt eine Zeichenfolge zurück, die das Ergebnis von zwei oder mehr miteinander verbundenen Zeichenfolgen ist. `CONCAT` akzeptiert zwei oder mehr Argumente.

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

Hinweis: Im Gegensatz zum Verknüpfen von Zeichenfolgen mit dem Zeichenfolgenverkettungsoperator (`+`) wird dieser beim Übergeben eines Nullwerts an die `concat` Funktion implizit in einen leeren String konvertiert:

```
SELECT CONCAT('This', NULL, ' is', ' my', ' string'), -- returns 'This is my string'
       'This' + NULL + ' is' + ' my' + ' string' -- returns NULL.
```

Auch Argumente eines Nicht-String-Typs werden implizit in einen String umgewandelt:

```
SELECT CONCAT('This', ' is my ', 3, 'rd string') -- returns 'This is my 3rd string'
```

Nicht-String-Typ-Variablen werden ebenfalls in das String-Format konvertiert. Sie müssen sie nicht manuell konvertieren oder in String umwandeln:

```
DECLARE @Age INT=23;
SELECT CONCAT('Ram is ', @Age,' years old');  -- returns 'Ram is 23 years old'
```

SQL Server 2012

Ältere Versionen unterstützen die `CONCAT` Funktion nicht und müssen stattdessen den Zeichenfolgenverkettungsoperator (`+`) verwenden. Nicht-String-Typen müssen in String-Typen umgewandelt oder konvertiert werden, um sie auf diese Weise zu verketten.

```
SELECT 'This is the number ' + CAST(42 AS VARCHAR(5)) --returns 'This is the number 42'
```

Niedriger

Gibt einen Zeichenausdruck (`varchar` oder `nvarchar`) zurück, nachdem alle Großbuchstaben in Kleinbuchstaben umgewandelt wurden.

Parameter:

1. Zeichenausdruck Jeder Ausdruck von Zeichen- oder Binärdaten, der implizit in `varchar` konvertiert werden `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'
```

Oberer, höher

Gibt einen Zeichenausdruck (`varchar` oder `nvarchar`) zurück, nachdem alle Kleinbuchstaben in Großbuchstaben umgewandelt wurden.

Parameter:

1. Zeichenausdruck Jeder Ausdruck von Zeichen- oder Binärdaten, der implizit in `varchar` konvertiert werden `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

Gibt einen Zeichenausdruck (`varchar` oder `nvarchar`) zurück, nachdem alle führenden Leerzeichen entfernt wurden, dh Leerzeichen von links bis zum ersten nicht-weißen Leerzeichen.

Parameter:

1. Zeichenausdruck. Jeder Ausdruck von Zeichen- oder Binärdaten, der implizit in `varcher` konvertiert werden `varcher` , mit Ausnahme von `text` , `ntext` und `image` .

```
SELECT LTRIM('    This is my string') -- Returns 'This is my string'
```

RTrim

Gibt einen Zeichenausdruck (`varchar` oder `nvarchar`) zurück, nachdem alle `nvarchar` Leerzeichen entfernt wurden, dh Leerzeichen vom rechten Ende der Zeichenfolge bis zum ersten nicht-weißen Leerzeichen links.

Parameter:

1. Zeichenausdruck. Jeder Ausdruck von Zeichen- oder Binärdaten, der implizit in `varcher` konvertiert werden `varcher` , mit Ausnahme von `text` , `ntext` und `image` .

```
SELECT RTRIM('This is my string     ') -- Returns 'This is my string'
```

Unicode

Gibt den ganzzahligen Wert zurück, der den Unicode-Wert des ersten Zeichens des Eingabeausdrucks darstellt.

Parameter:

1. Unicode-Zeichenausdruck Jeder gültige `nchar` oder `nvarchar` Ausdruck.

```
SELECT UNICODE(N'Ɛ') -- Returns 400

DECLARE @Unicode nvarchar(11) = N'Ɛ is a char'
SELECT UNICODE(@Unicode) -- Returns 400
```

NChar

Gibt das (die) Unicode-Zeichen (`nchar(1)` oder `nvarchar(2)`) zurück, die dem ganzzahligen Argument entsprechen, das es empfängt, wie vom Unicode-Standard definiert.

Parameter:

1. ganzzahliger Ausdruck. Jeder ganzzahlige Ausdruck, bei dem es sich um eine positive Zahl zwischen 0 und 65535 handelt. Wenn die Sortierung der Datenbank das Zusatzzeichen (CS) -Flag unterstützt, liegt der unterstützte Bereich zwischen 0 und 1114111. Wenn der Ganzzahlausdruck nicht in diesen Bereich fällt, ist `null` ist zurückgekommen.

```
SELECT NCHAR(257) -- Returns 'ā'
SELECT NCHAR(400) -- Returns 'Ɛ'
```

Umkehren

Gibt einen Stringwert in umgekehrter Reihenfolge zurück.

Parameter:

1. Zeichenfolgenausdruck Alle Zeichenfolgen oder Binärdaten, die implizit in `varchar` konvertiert werden `varchar` .

```
Select REVERSE('Sql Server') -- Returns 'revreS lqS'
```

PatIndex

Gibt die Startposition des ersten Vorkommens eines angegebenen Musters im angegebenen Ausdruck zurück.

Parameter:

1. Muster. Ein Zeichenausdruck enthält die zu findende Sequenz. Begrenzt auf eine maximale Länge von 8000 Zeichen. Platzhalter (`%` , `_`) können im Muster verwendet werden. Wenn das Muster nicht mit einem Platzhalter beginnt, stimmt es möglicherweise nur mit dem Anfang des Ausdrucks überein. Wenn es nicht mit einem Platzhalter endet, stimmt es möglicherweise nur mit dem Ende des Ausdrucks überein.

2. Ausdruck. Beliebiger String-Datentyp

```
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.
```

Platz

Gibt eine Zeichenfolge (`varchar`) aus wiederholten Leerzeichen zurück.

Parameter:

1. ganzzahliger Ausdruck. Ein ganzzahliger Ausdruck bis 8000. Bei einem negativen `null` wird `null` zurückgegeben. Bei 0 wird eine leere Zeichenfolge zurückgegeben. (Um eine Zeichenfolge mit mehr als 8000 Leerzeichen zurückzugeben, verwenden Sie Replicate.

```
SELECT SPACE(-1) -- Returns NULL
SELECT SPACE(0)  -- Returns an empty string
SELECT SPACE(3)  -- Returns '   ' (a string containing 3 spaces)
```

Replizieren

Wiederholt einen String-Wert eine bestimmte Anzahl von Malen.

Parameter:

1. Zeichenfolgenausdruck Der Zeichenfolgenausdruck kann eine Zeichenfolge oder binäre Daten sein.
2. ganzzahliger Ausdruck. Jeder ganzzahlige Typ, einschließlich `bigint` . Wenn negativ, wird `null` zurückgegeben. Bei 0 wird eine leere Zeichenfolge zurückgegeben.

```
SELECT REPLICATE('a', -1)  -- Returns NULL

SELECT REPLICATE('a', 0)  -- Returns ''

SELECT REPLICATE('a', 5)  -- Returns 'aaaaa'

SELECT REPLICATE('Abc', 3) -- Returns 'AbcAbcAbc'
```

Anmerkung: Wenn der Zeichenfolgenausdruck nicht vom Typ `varchar(max)` oder `nvarchar(max)` , überschreitet der Rückgabewert 8000 Zeichen nicht. Replicate stoppt vor dem Hinzufügen der Zeichenfolge, durch die der Rückgabewert diesen Grenzwert überschreitet:

```
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
```

Ersetzen

Gibt eine Zeichenfolge (`varchar` oder `nvarchar`) zurück, bei der alle Vorkommen einer angegebenen Unterzeichenfolge durch eine andere Unterzeichenfolge ersetzt werden.

Parameter:

1. Zeichenfolgenausdruck Dies ist die Zeichenfolge, die durchsucht werden würde. Es kann ein Zeichentyp oder ein binärer Datentyp sein.
2. Muster. Dies ist die Unterzeichenfolge, die ersetzt werden würde. Es kann ein Zeichentyp oder ein binärer Datentyp sein. Das Musterargument darf keine leere Zeichenfolge sein.
3. Ersatz. Dies ist die Unterzeichenfolge, die die Musterunterkette ersetzen würde. Es können Zeichen- oder Binärdaten sein.

```
SELECT REPLACE('This is my string', 'is', 'XX') -- Returns 'ThXX XX my string'.
```

Anmerkungen:

- Wenn der Zeichenfolgenausdruck nicht vom Typ `varchar(max)` oder `nvarchar(max)` , `nvarchar(max) replace` Funktion `replace` den Rückgabewert auf 8000 Zeichen ab.
- Der Rückgabedatentyp hängt von den Eingabedatentypen ab. `nvarchar` `nvarchar` wenn einer der Eingabewerte `nvarchar` oder ansonsten `varchar` .
- `NULL` wenn einer der Eingangsparameter `NULL`

String_Split

SQL Server 2016

Teilt einen Zeichenfolgenausdruck mit einem Trennzeichen. Beachten Sie, dass `STRING_SPLIT()` eine Tabellenwertfunktion ist und daher innerhalb der `FROM` Klausel verwendet werden muss.

Parameter:

1. Schnur Beliebiger Zeichentypausdruck (`char` , `nchar` , `varchar` oder `nvarchar`)
2. Separator. Ein einzelner Zeichenausdruck eines beliebigen Typs (`char(1)` , `nchar(1)` , `varchar(1)` oder `nvarchar(1)`).

Gibt eine einzelne Spaltentabelle zurück, in der jede Zeile ein Fragment der Zeichenfolge enthält. Der Name der Spalten lautet `value` und der Datentyp ist `nvarchar` wenn einer der Parameter entweder `nchar` oder `nvarchar` , andernfalls `varchar` .

Im folgenden Beispiel wird eine Zeichenfolge mit Leerzeichen als Trennzeichen aufgeteilt:

```
SELECT value FROM STRING_SPLIT('Lorem ipsum dolor sit amet.', ' ');
```

Ergebnis:

```
value
```

```
-----
Lorem
ipsum
dolor
sit
amet.
```

Bemerkungen:

> Die Funktion `STRING_SPLIT` ist nur unter dem Kompatibilitätsgrad **130** verfügbar. Wenn Ihre Datenbankkompatibilitätsstufe niedriger als 130 ist, kann der SQL Server die `STRING_SPLIT` Funktion nicht finden und ausführen. Sie können den Kompatibilitätsgrad einer Datenbank mit dem folgenden Befehl ändern:

```
ALTER DATABASE [database_name] SET COMPATIBILITY_LEVEL = 130
```

SQL Server 2016

Ältere Versionen von SQL Server verfügen nicht über eine integrierte Split-String-Funktion. Es gibt viele benutzerdefinierte Funktionen, die das Problem des Aufteilens einer Zeichenfolge lösen. Sie können Aaron Bertrands Artikel Split Strings richtig lesen - oder den nächstbesten Weg, um einige von ihnen umfassend zu vergleichen.

Str

Gibt Zeichendaten (`varchar`) aus numerischen Daten zurück.

Parameter:

1. Float-Ausdruck. Ein ungefährer numerischer Datentyp mit Dezimalpunkt.
2. Länge. **wahlweise.** Die Gesamtlänge des zurückgegebenen Zeichenfolgenausdrucks, einschließlich Ziffern, Dezimalzeichen und Leerzeichen (falls erforderlich). Der Standardwert ist 10.
3. Dezimal. **wahlweise.** Die Anzahl der Ziffern rechts vom Dezimalpunkt. Bei mehr als 16 würde das Ergebnis auf sechzehn Stellen rechts vom Dezimalpunkt abgeschnitten.

```
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

Gibt eine Unicode-Zeichenfolge zurück, die von Trennzeichen umgeben ist, um sie zu einer gültigen, durch SQL Server getrennten Bezeichner zu machen.

Parameter:

1. Zeichenkette. Eine Zeichenfolge aus Unicode-Daten mit bis zu 128 Zeichen (`sysname`). Wenn eine Eingabezeichenfolge länger als 128 Zeichen ist, gibt die Funktion `null` .
2. Anführungszeichen **Optional** . Ein einzelnes Zeichen, das als Trennzeichen verwendet werden soll. Kann ein einfaches Anführungszeichen (`'` oder ``), eine linke oder rechte Klammer (`{` , `[` , `(` , `<` oder `>` , `)` , `]` , `}`) oder ein doppeltes Anführungszeichen (`"`) sein. Jeder andere Wert gibt null zurück Der Standardwert ist eckige Klammern.

```
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

Gibt einen `varchar` Code (`varchar`) zurück, um die phonetische Ähnlichkeit von zwei Zeichenfolgen zu bewerten.

Parameter:

1. Zeichenausdruck. Ein alphanumerischer Ausdruck von Zeichendaten.

Die Soundex-Funktion erstellt einen vierstelligen Code, der darauf basiert, wie der Zeichenausdruck klingen würde, wenn er gesprochen wird. Das erste Zeichen ist die Großbuchstabe des ersten Zeichens des Parameters, die restlichen 3 Zeichen sind Zahlen, die die Buchstaben im Ausdruck darstellen (außer a, e, i, o, u, h, w und y, die ignoriert werden) .

```
SELECT SOUNDEX ('Smith') -- Returns 'S530'

SELECT SOUNDEX ('Smythe') -- Returns 'S530'
```

Unterschied

Gibt einen Integer-Wert (`int` Wert) zurück, der den Unterschied zwischen den Soundex-Werten von zwei Zeichenausdrücken angibt.

Parameter:

1. Zeichenausdruck 1.
2. Zeichenausdruck 2.

Beide Parameter sind alphanumerische Ausdrücke von Zeichendaten.

Die zurückgegebene Ganzzahl ist die Anzahl der Zeichen in den Soundex-Werten der Parameter, die gleich sind. 4 bedeutet, dass die Ausdrücke sehr ähnlich sind und 0, dass sie sehr unterschiedlich sind.

```
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

Gibt einen `NVARCHAR` Wert zurück, der mit dem angegebenen Format und der angegebenen Kultur formatiert ist (falls angegeben). Dies wird hauptsächlich zum Konvertieren von Datum / Uhrzeit-Typen in Strings verwendet.

Parameter:

1. `value` . Ein Ausdruck eines unterstützten Datentyps, der formatiert werden soll. gültige Typen sind unten aufgeführt.
2. `format` . Ein `NVARCHAR` . Informationen zu Standard- und benutzerdefinierten Formatzeichenfolgen finden Sie in der offiziellen Dokumentation von Microsoft.
3. `culture` . **Optional** . `nvarchar` Argument, das eine Kultur angibt. Der Standardwert ist die Kultur der aktuellen Sitzung.

DATUM

Standardformatzeichenfolgen verwenden:

```
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'
```

Benutzerdefinierte Formatzeichenfolgen verwenden:

```
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` kann auch zur Formatierung von `CURRENCY` , `PERCENTAGE` und `NUMBERS` .

WÄHRUNG

```
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 €'
```

Wir können die Anzahl der Nachkommastellen angeben.

```
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'
```

PROZENTSATZ

```
   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 %'
```

NUMMER

```
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 gültiger Werttypen: (Quelle)

```
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
```

Wichtige Notizen:

- `FORMAT` gibt `NULL` für andere Fehler als eine nicht gültige Kultur zurück. Beispielsweise wird `NULL` zurückgegeben, wenn der im Format angegebene Wert nicht gültig ist.
- `FORMAT` setzt das Vorhandensein der Common Language Runtime (CLR) von .NET `FORMAT` .
- `FORMAT` stützt sich auf CLR-Formatierungsregeln, nach denen Doppelpunkte und Perioden mit `FORMAT` werden müssen. Wenn der Formatstring (zweiter Parameter) einen Doppelpunkt oder eine Periode enthält, muss daher der Doppelpunkt oder die Periode mit einem umgekehrten Schrägstrich (Escape-Zeichen) versehen werden, wenn ein Eingabewert (erster Parameter) den Zeitdatentyp hat.

Siehe auch Formatierung von Datum und Uhrzeit unter Verwendung des FORMAT-Dokumentationsbeispiels.

String_escape

SQL Server 2016

Escape Sonderzeichen in Texten und gibt Text (`nvarchar(max)`) mit `nvarchar(max)`

Parameter:

1. Text. ist ein `nvarchar` Ausdruck, der die Zeichenfolge darstellt, die mit `nvarchar` werden soll.
2. Art. Escape-Regeln, die angewendet werden. Momentan ist der einzige unterstützte Wert `'json'` .

```
SELECT STRING_ESCAPE('\   /
\\    "     ', 'json') -- returns '\\\t\/\n\\\\\t\"\t'
```

Liste der Zeichen, die maskiert werden:

```
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
```

String-Funktionen online lesen: https://riptutorial.com/de/sql-server/topic/4113/string-funktionen

Kapitel 95: Systemdatenbank - TempDb

Examples

Identifizieren Sie die TempDb-Nutzung

Die folgende Abfrage enthält Informationen zur Verwendung von TempDb. Durch die Analyse der Zählungen können Sie feststellen, welche Sache TempDb beeinflusst

```
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

TempDB-Datenbankdetails

Die folgende Abfrage kann verwendet werden, um die Details der TempDB-Datenbank abzurufen:

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

ODER

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

Mit Hilfe von DMV können Sie überprüfen, wie viel TempDb-Speicherplatz Ihre Sitzung belegt. Diese Abfrage ist beim Debuggen von TempDb-Problemen hilfreich

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

Systemdatenbank - TempDb online lesen: https://riptutorial.com/de/sql-server/topic/4427/systemdatenbank---tempdb

Kapitel 96: Tabellenwertparameter

Bemerkungen

Tabellenwertparameter (kurz TVP) sind Parameter, die an eine gespeicherte Prozedur oder Funktion übergeben werden, die Daten enthält, die in Tabellenstruktur vorliegen. Die Verwendung von Tabellenwertparametern erfordert das Erstellen eines benutzerdefinierten Tabellentyps für den verwendeten Parameter.

Tabellenwerte sind gelesene Parameter.

Examples

Verwenden eines Tabellenwertparameters, um mehrere Zeilen in eine Tabelle einzufügen

Definieren Sie zunächst einen verwendeten definierten Tabellentyp :

```
CREATE TYPE names as TABLE
(
    FirstName varchar(10),
    LastName varchar(10)
)
GO
```

Erstellen Sie die gespeicherte Prozedur:

```
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
```

Ausführen der gespeicherten Prozedur:

```
DECLARE @names dbo.Names
INSERT INTO @Names VALUES
('Zohar', 'Peled'),
('First', 'Last')

EXEC dbo.prInsertNames @Names
```

Tabellenwertparameter online lesen: https://riptutorial.com/de/sql-server/topic/5285/tabellenwertparameter

Kapitel 97: Termine

Syntax

- EOMONTH (*start_date* [, month_to_add])

Bemerkungen

Wie in https://msdn.microsoft.com/en-us/library/ms187819.aspx angegeben , sind `DateTime` s nur auf 3 `DateTime` genau.

Rundung von datetime Bruchzeit Sekunden Die Präzisions-datetime-Werte werden auf Inkremente von 0,000, 0,003 oder 0,007 Sekunden gerundet, wie in der folgenden Tabelle gezeigt.

Vom Benutzer angegebener Wert	Vom System gespeicherter Wert
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	
------	------

Wenn mehr Genauigkeit erforderlich ist, sollten `time` , `datetime2` oder `datetimeoffset` verwendet werden.

Examples

Datums- und Uhrzeitformatierung mit CONVERT

Sie können die CONVERT-Funktion verwenden, um einen Datetime-Datentyp in eine formatierte Zeichenfolge umzuwandeln.

```
SELECT GETDATE() AS [Result] -- 2016-07-21 07:56:10.927
```

Sie können auch einige integrierte Codes verwenden, um in ein bestimmtes Format zu konvertieren. Hier sind die in SQL Server integrierten Optionen:

```
DECLARE @convert_code INT = 100 -- See Table Below
SELECT CONVERT(VARCHAR(30), GETDATE(), @convert_code) AS [Result]
```

@convert_code	Ergebnis
100	"21. Juli 2016 07:56"
101	"21.07.2016"
102	2016.07.21
103	"21/07/2016"
104	"21.07.2016"
105	21-07-2016
106	"21. Juli 2016"
107	"21. Juli 2016"
108	"07:57:05"
109	"21. Juli 2016 7: 57: 45: 707 Uhr"
110	21.07.2016
111	"2016/07/21"
112	"20160721"
113	"21. Juli 2016 07: 57: 59: 553"
114	07: 57: 59: 553
120	"2016-07-21 07:57:59"

@convert_code	Ergebnis
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: 340 Uhr
131	16/10/1437 7: 58: 34: 340 Uhr

```
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
```

Datums- und Zeitformatierung mit FORMAT

SQL Server 2012

Sie können die neue Funktion nutzen: `FORMAT()` .

Damit können Sie Ihre `DATETIME` Felder in Ihr eigenes `VARCHAR` Format umwandeln.

Beispiel

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'dddd, MMMM dd, yyyy hh:mm:ss tt')
```

Montag, 05. September 2016 00:01:02 Uhr

Argumente

Da das `DATETIME` Format `2016-09-05 00:01:02.333` formatiert ist, zeigt das folgende Diagramm,

welche Ausgabe es für das angegebene Argument geben würde.

Streit	Ausgabe
yyyy	2016
yy	16
MMMM	September
MM	09
M	9
dddd	Montag
ddd	Mo
dd	05
d	5
HH	00
H	0
hh	12
h	12
mm	01
m	1
ss	02
s	2
tt	AM
t	EIN
F f f	333
ff	33
f	3

Sie können der Funktion `FORMAT()` auch ein einzelnes Argument `FORMAT()` , um eine `FORMAT()` Ausgabe zu generieren:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'U')
```

Montag, 05. September 2016 04:01:02 Uhr

Einzelargument	Ausgabe
D	Montag, 05. September 2016
d	9/5/2016
F	Montag, 05. September 2016 00:01:02 Uhr
f	Montag, 05. September 2016, 12:01 Uhr
G	05.09.2016 12:01:02 Uhr
G	05.09.2016 12:01 Uhr
M	September 05
O	2016-09-05T00: 01: 02.3330000
R	Mo, 05 Sep 2016 00:01:02 GMT
s	2016-09-05T00: 01: 02
T	12:01:02 Uhr
t	12:01 Uhr
U	Montag, 05. September 2016 04:01:02 Uhr
u	2016-09-05 00: 01: 02Z
Y	September 2016

Hinweis: Die obige Liste verwendet die `en-US` Kultur. Über den dritten Parameter kann eine andere Kultur für `FORMAT()` angegeben werden:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'U', 'zh-cn')
```

2016 9 5 4:01:02

Holen Sie sich die aktuelle DateTime

Die eingebauten Funktionen `GETDATE` und `GETUTCDATE` jeweils das aktuelle Datum und die aktuelle

Uhrzeit ohne Zeitzonenversatz zurück.

Der Rückgabewert beider Funktionen basiert auf dem Betriebssystem des Computers, auf dem die Instanz von SQL Server ausgeführt wird.

Der Rückgabewert von GETDATE repräsentiert die aktuelle Uhrzeit in derselben Zeitzone wie das Betriebssystem. Der Rückgabewert von GETUTCDATE repräsentiert die aktuelle UTC-Zeit.

Jede Funktion kann in der `SELECT` Klausel einer Abfrage oder als Teil eines booleschen Ausdrucks in der `WHERE` Klausel enthalten sein.

Beispiele:

```
-- 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()
```

Es gibt ein paar andere integrierte Funktionen, die verschiedene Variationen des aktuellen Datums und der Uhrzeit anzeigen:

```
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 zum Hinzufügen und Entfernen von Zeiträumen

Allgemeine Syntax:

```
DATEADD (datepart , number , datetime_expr)
```

Um ein Zeitmaß hinzuzufügen, muss die `number` positiv sein. Um ein Zeitmaß abzuziehen, muss die `number` negativ sein.

Beispiele

```
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
```

`DATEADD` : `DATEADD` akzeptiert auch Abkürzungen im Parameter `datepart` . Die Verwendung dieser Abkürzungen wird im Allgemeinen nicht empfohlen, da sie verwirrend sein können (`m` vs `mi` , `ww` vs `w` usw.).

Datum Teile Referenz

Dies sind die `datepart` , die für Datums- und `datepart` verfügbar sind:

Datumsteil	Abkürzungen
Jahr	yy, yyyy
Quartal	qq, q
Monat	mm, m
Tag des Jahres	dy, y
Tag	dd, d
Woche	wk, ww
Wochentag	dw, w
Stunde	hh
Minute	Mindest
zweite	ss, s
Millisekunde	Frau
Mikrosekunde	mcs
Nanosekunde	ns

HINWEIS : Die Verwendung von Abkürzungen wird im Allgemeinen nicht empfohlen, da sie verwirrend sein können (`m` vs `mi` , `ww` vs `w` usw.). Die Langversion der `datepart` Darstellung `datepart` für Klarheit und Lesbarkeit und sollte nach Möglichkeit verwendet werden (`month` , `minute` , `week` , `weekday` usw.).

DATEDIFF zur Berechnung von Zeitdifferenzen

Allgemeine Syntax:

```
DATEDIFF (datepart, datetime_expr1, datetime_expr2)
```

Es wird eine positive Zahl zurückgegeben, wenn `datetime_expr` relativ zu `datetime_expr2` in der Vergangenheit `datetime_expr2` , ansonsten eine negative Zahl.

Beispiele

```
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
```

`DATEDIFF` : `DATEDIFF` akzeptiert auch Abkürzungen im Parameter `datepart` . Die Verwendung dieser Abkürzungen wird im Allgemeinen nicht empfohlen, da sie verwirrend sein können (`m` vs `mi` , `ww` vs `w` usw.).

`DATEDIFF` kann auch verwendet werden, um den Versatz zwischen UTC und der lokalen Zeit des SQL-Servers zu bestimmen. Mit der folgenden Anweisung kann der Versatz zwischen UTC und lokaler Zeit (einschließlich Zeitzone) berechnet werden.

```
select  DATEDIFF(hh, getutcdate(), getdate()) as 'CentralTimeOffset'
```

DATEPART & DATENAME

`DATEPART` gibt den angegebenen `datepart` des angegebenen Datetime-Ausdrucks als numerischen Wert zurück.

`DATENAME` gibt eine Zeichenfolge zurück, die den angegebenen `datepart` des angegebenen Datums darstellt. In der Praxis ist `DATENAME` meistens nützlich, um den Namen des Monats oder des Wochentags `DATENAME` .

Es gibt auch einige Abkürzungsfunktionen, um das Jahr, den Monat oder den Tag eines datetime-Ausdrucks abzurufen, die sich wie `DATEPART` mit ihren jeweiligen `datepart` Einheiten verhalten.

Syntax:

```
DATEPART ( datepart , datetime_expr )
DATENAME ( datepart , datetime_expr )
DAY ( datetime_expr )
MONTH ( datetime_expr )
YEAR ( datetime_expr )
```

Beispiele:

```
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
```

`DATEPART` : `DATEPART` und `DATENAME` akzeptieren auch Abkürzungen im Parameter `datepart` . Die Verwendung dieser Abkürzungen wird im Allgemeinen nicht empfohlen, da sie verwirrend sein können (`m` vs `mi` , `ww` vs `w` usw.).

Den letzten Tag eines Monats bekommen

Mit den Funktionen `DATEADD` und `DATEDIFF` kann das letzte Datum eines Monats zurückgegeben werden.

```
SELECT DATEADD(d, -1, DATEADD(m, DATEDIFF(m, 0, '2016-09-23') + 1, 0))
-- 2016-09-30 00:00:00.000
```

SQL Server 2012

Die `EOMONTH` Funktion bietet eine `EOMONTH` Möglichkeit, das letzte Datum eines Monats zurückzugeben, und verfügt über einen optionalen Parameter zum Versetzen des Monats.

```
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
```

Nur Datum aus einer DateTime zurückgeben

Es gibt viele Möglichkeiten, ein Date aus einem DateTime-Objekt zurückzugeben

1. `SELECT CONVERT(Date, GETDATE())`
2. `SELECT DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()))` gibt 2016-07-21 00: 00: 00.000 zurück
3. `SELECT CAST(GETDATE() AS DATE)`
4. `SELECT CONVERT(CHAR(10),GETDATE(),111)`
5. `SELECT FORMAT(GETDATE(), 'yyyy-MM-dd')`

Beachten Sie, dass die Optionen 4 und 5 eine Zeichenfolge und kein Datum zurückgeben.

Funktion erstellen, um das Alter einer Person an einem bestimmten Datum zu berechnen

Diese Funktion benötigt zwei datetime-Parameter, das DOB und ein Datum, um das Alter bei zu überprüfen

```
  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
```

B. um das Alter einer am 01.01.2000 geborenen Person zu überprüfen

```
SELECT  dbo.Calc_Age('2000-01-01',Getdate())
```

CROSS PLATFORM DATE OBJECT

SQL Server 2012

In Transact SQL können Sie ein Objekt als `Date` (oder `DateTime`) definieren, indem Sie die Funktion `[DATEFROMPARTS][1]` (oder `[DATETIMEFROMPARTS][1]`) wie folgt verwenden:

```
 DECLARE @myDate DATE=DATEFROMPARTS(1988,11,28)
 DECLARE @someMoment DATETIME=DATEFROMPARTS(1988,11,28,10,30,50,123)
```

Die von Ihnen angegebenen Parameter sind Jahr, Monat, Tag für die Funktion `DATEFROMPARTS` und für die Funktion `DATETIMEFROMPARTS` müssen Sie Jahr, Monat, Tag, Stunde, Minuten, Sekunden und Millisekunden `DATETIMEFROMPARTS` .

Diese Methoden sind nützlich und sollten verwendet werden, da die Verwendung der einfachen Zeichenfolge zum Erstellen eines Datums (oder einer Datumszeit) je nach Region, Speicherort oder Datumsformat des Host-Computers fehlschlagen kann.

Datumsformat erweitert

Datumsformat	SQL-Anweisung	Beispielausgabe
JJ-MM-TT	WÄHLEN SIE RECHTS (CONVERT (VARCHAR (10), SYSDATETIME (), 20), 8) AS [JJ-MM-TT] SELECT REPLACE (KONVERTIEREN (VARCHAR (8),	11-06-08

Datumsformat	SQL-Anweisung	Beispielausgabe
	SYSDATETIME (), 11), '/', '-') AS [JJ-MM-TT]	
JJJJ-MM-TT	SELECT CONVERT (VARCHAR (10), SYSDATETIME (), 120) ALS [JJJJ-MM-TT] SELECT REPLACE (KONVERTIEREN (VARCHAR (10), SYSDATETIME (), 111), '/', '-') AS [JJJJ-MM-TT]	2011-06-08
JJJJ-MD	WÄHLEN SIE CAST (JAHR (SYSDATETIME ()) AS VARCHAR (4)) + '-' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (TAG (SYSDATETIME ()) AS VARCHAR (2)) AS [JJJJ-MD]	2011-6-8
YY-MD	WÄHLEN SIE RECHT (CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)), 2) + '-' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (TAG (SYSDATETIME ( )) AS VARCHAR (2)) AS [YY-MD]	11-6-8
MD-JJJJ	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (TAG (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (JAHR (SYSDATETIME ()) AS VARCHAR (4)) AS [MD-JJJJ]	6-8-2011
MD-YY	WÄHLEN SIE CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (TAG (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RECHTS (CAST (YEAR (SYSDATETIME () AS) VARCHAR (4)), 2) AS [MD-YY]	6-8-11
DM-JJJJ	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (JAHR (SYSDATETIME ()) AS VARCHAR (4)) AS [DM-JJJJ]	8-6-2011
DM-JJ	WÄHLEN SIE CAST (TAG (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RECHTS (CAST (YEAR (SYSDATETIME () AS)) VARCHAR (4)), 2) AS [DM-YY]	8-6-11
JJ-MM	WÄHLEN SIE RECHTS (CONVERT (VARCHAR (7), SYSDATETIME (), 20), 5) AS [JJ-MM] SUBSTRING WÄHLEN (CONVERT (VARCHAR (10), SYSDATETIME (), 120), 3, 5) AS [JJ-MM]	11-06
JJJJ-MM	SELECT CONVERT (VARCHAR (7), SYSDATETIME (), 120) ALS [JJJJ-MM]	2011-06

Datumsformat	SQL-Anweisung	Beispielausgabe
YY-M	WÄHLEN SIE RECHTS (CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)), 2) + '-' + CAST (MONAT (SYSDATETIME ()) ALS VARCHAR (2)) AS [YY-M]	11-6
JJJJ-M	SELECT CAST (JAHR (SYSDATETIME ()) AS VARCHAR (4)) + '-' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) ALS [JJJJ-M]	2011-6
MM-JJ	WÄHLEN SIE RECHTS (CONVERT (VARCHAR (8), SYSDATETIME (), 5), 5) AS [MM-JJ] SELECT SUBSTRING (CONVERT (VARCHAR (8), SYSDATETIME (), 5), 4, 5) AS [MM-YY]	06-11
MM-JJJJ	WÄHLEN SIE RECHTS (CONVERT (VARCHAR (10), SYSDATETIME (), 105), 7) ALS [MM-JJJJ]	06-2011
M-JJ	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RECHTS (CAST (JAHR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M-YY]	6-11
M-JJJJ	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) ALS [M-JJJJ]	6-2011
MM-DD	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 10) AS [MM-DD]	06-08
DD-MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 5) AS [DD-MM]	08-06
MD	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [MD]	6-8
DM	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) AS [DM]	8-6
M / T / JJJJ	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (TAG (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (JAHR (SYSDATETIME ()) AS VARCHAR (4)) AS [M / T / JJJJ]	08.06.2011
M / T / JJ	WÄHLEN SIE CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (TAG (SYSDATETIME ())	6/8/11

Datumsformat	SQL-Anweisung	Beispielausgabe
	AS VARCHAR (2)) + '/' + RECHTS (CAST (YEAR (SYSDATETIME () AS) VARCHAR (4)), 2) AS [M / T / JJ]	
D / M / JJJJ	SELECT CAST (TAG (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (JAHR (SYSDATETIME ()) AS VARCHAR (4)) AS [D / M / JJJJ]	8/6/2011
D / M / YY	WÄHLEN SIE CAST (TAG (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RECHTS (CAST (YEAR (SYSDATETIME () AS)) VARCHAR (4)), 2) AS [D / M / YY]	8/6/11
JJJJ / M / D	SELECT CAST (JAHR (SYSDATETIME ()) AS VARCHAR (4)) + '/' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (TAG (SYSDATETIME ()) AS VARCHAR (2)) AS [JJJJ / M / D]	2011/6/8
JJ / M / T	WÄHLEN SIE RECHTS (CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)), 2) + '/' + CAST (MONAT (SYSDATETIME ()) ALS VARCHAR (2)) + '/' + CAST (TAG (SYSDATETIME ( )) AS VARCHAR (2)) AS [JJ / M / T]	11/6/8
MM / JJ	WÄHLEN SIE RECHTS (CONVERT (VARCHAR (8), SYSDATETIME (), 3), 5) AS [MM / YY]	06/11
MM / JJJJ	WÄHLEN SIE RECHTS (CONVERT (VARCHAR (10), SYSDATETIME (), 103), 7) AS [MM / JJJJ]	06/2011
M / JJ	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RECHTS (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M / YY]	6/11
M / JJJJ	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) ALS [M / JJJJ]	6/2011
JJ / MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 11) ALS [JJ / MM]	11/06
JJJJ / MM	SELECT CONVERT (VARCHAR (7), SYSDATETIME (), 111) ALS [JJJJ / MM]	2011/06

Datumsformat	SQL-Anweisung	Beispielausgabe
JJ / M	WÄHLEN SIE RECHTS (CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)), 2) + '/' + CAST (MONAT (SYSDATETIME ()) ALS VARCHAR (2)) AS [JJ / M]	11/6
JJJJ / M	SELECT CAST (JAHR (SYSDATETIME ()) AS VARCHAR (4)) + '/' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) ALS [JJJJ / M]	2011/6
MM / DD	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 1) AS [MM / DD]	06/08
DD / MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 3) AS [DD / MM]	08/06
M / D	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [M / D]	6/8
DM	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) AS [D / M]	8/6
MM.DD.YYYY	SELECT REPLACE (KONVERTIEREN (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 (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) ALS [MDYYYY]	6.8.2011
MDYY	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + RECHTS (CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)), 2) ALS [MDYY]	6.8.11
DD / MM / JJJJ	SELECT CONVERT (VARCHAR (10), SYSDATETIME (), 104) AS [DD.MM.YYYY]	08.06.2011
TT.MM.JJ	SELECT CONVERT (VARCHAR (10), SYSDATETIME (), 4) AS [DD.MM.YY]	08.06.11
DMYYYY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONAT (SYSDATETIME	8.6.2011

Datumsformat	SQL-Anweisung	Beispielausgabe
	()) AS VARCHAR (2)) + '.' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) ALS [DMYYYY]	
DMYY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '.' + RECHTS (CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)), 2) ALS [DMYY]	8.6.11
JJJJ.MD	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '.' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) ALS [JJJJ.MD]	2011.6.8
YY.MD	WÄHLEN SIE RECHTS (CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)), 2) + '.' + CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [YY.MD]	11.6.8
MM.JJJJ	WÄHLEN SIE RECHTS (CONVERT (VARCHAR (10), SYSDATETIME (), 104), 7) AS [MM.YYYY]	06.2011
MM.YY	WÄHLEN SIE RECHTS (CONVERT (VARCHAR (8), SYSDATETIME (), 4), 5) AS [MM.YY]	06.11
M.YYYY	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)) ALS [M.YYYY]	6.2011
M.YY	SELECT CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) + '.' + RECHTS (CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)), 2) ALS [M.YY]	6.11
JJJJ.MM	SELECT CONVERT (VARCHAR (7), SYSDATETIME (), 102) ALS [JJJJ.MM]	2011.06
YY.MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 2) AS [YY.MM]	11.06
JJJJ.M	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '.' + CAST (MONAT (SYSDATETIME ()) ALS VARCHAR (2)) ALS [JJJJ.M]	2011.6
YY.M	WÄHLEN SIE RECHTS (CAST (JAHR (SYSDATETIME ()) ALS VARCHAR (4)), 2) + '.' +	11.6

Datumsformat	SQL-Anweisung	Beispielausgabe
	CAST (MONAT (SYSDATETIME ()) AS VARCHAR (2)) AS [JJ.M]	
MM.DD	SELECT RIGHT (CONVERT (VARCHAR (8), SYSDATETIME (), 2), 5) AS [MM.DD]	06.08
DD.MM	SELECT CONVERT (VARCHAR (5), SYSDATETIME (), 4) AS [DD.MM]	08.06
MMDDYYYY	SELECT REPLACE (KONVERTIEREN (VARCHAR (10), SYSDATETIME (), 101), '/', '') AS [MMDDYYYY]	06082011
MMDDYY	SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 1), '/', '') AS [MMDDYY]	060811
DD / MM / JJJJ	SELECT REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 103), '/', '') AS [DDMMYYYY]	08062011
DDMMYY	SELECT REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 3), '/', '') AS [DDMMYY]	080611
MMYYYY	WÄHLEN SIE RECHTS (REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 103), '/', ''), 6) AS [MMYYYY]	062011
MMYY	WÄHLEN SIE RECHTS (REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 3), '/', ''), 4) AS [MMYY]	0611
JJJJMM	SELECT CONVERT (VARCHAR (6), SYSDATETIME (), 112) ALS [JJJJMM]	201106
YYMM	SELECT CONVERT (VARCHAR (4), SYSDATETIME (), 12) ALS [JJMM]	1106
Monat TT, JJJJ	SELECT DATENAME (MONAT, SYSDATETIME ()) + '' + RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + ',' + DATENAME (YEAR, SYSDATETIME ()) AS [Monat DD, YYYY]	8. Juni 2011
Mo JJJJ	SELECT LINKS (DATENAME (MONAT, SYSDATETIME ()), 3) + '' + DATENAME (YEAR, SYSDATETIME ()) AS [Mon YYYY]	Juni 2011
Monat JJJJ	SELECT DATENAME (MONAT, SYSDATETIME ()) + '' + DATENAME (YEAR, SYSDATETIME ()) AS [Monat JJJJ]	Juni 2011

Datumsformat	SQL-Anweisung	Beispielausgabe
DD Monat	SELECT RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + '' + DATENAME (MONTH, SYSDATETIME ()) AS [DD Month]	08. Juni
Monat DD	SELECT DATENAME (MONAT, SYSDATETIME ()) + '' + RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) AS [Month DD]	08. Juni
TT Monat JJ	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '' + DATENAME (MM, SYSDATETIME ()) + '' + RECHTS (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [TT Monat JJ]	08. Juni
TT Monat JJJJ	WÄHLEN SIE RECHTS ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + '' + DATENAME (MONAT, SYSDATETIME ()) + '' + DATENAME (YEAR, SYSDATETIME ()) AS [DD Monat JJJJ]	08. Juni 2011
Mo-JJ	SELECT ERSETZEN (RECHTS (CONVERT (VARCHAR (9), SYSDATETIME (), 6), 6), '', '-') AS [Mon-YY]	08-Jun
Mo-JJJJ	SELECT ERSETZEN (RECHTS (CONVERT (VARCHAR (11), SYSDATETIME (), 106), 8), '', '-') AS [Mon-YYYY]	Jun-2011
DD-Mo-JJ	SELECT REPLACE (KONVERTIEREN (VARCHAR (9), SYSDATETIME (), 6), '', '-') AS [DD-Mo-YY]	08-Jun-11
TT-Mo-JJJJ	SELECT REPLACE (KONVERTIEREN (VARCHAR (11), SYSDATETIME (), 106), '', '-') AS [DD-Mo-JJJJ]	08-Jun-2011

Termine online lesen: https://riptutorial.com/de/sql-server/topic/1471/termine

Kapitel 98: Transaktionsabwicklung

Parameter

Parameter	Einzelheiten
Transaktionsname	zur Benennung Ihrer Transaktion - nützlich mit dem Parameter [*with mark*], der eine sinnvolle Protokollierung ermöglicht - Groß- und Kleinschreibung (!)
mit dem Zeichen ['description']	kann zu [*Transaktionsname*] hinzugefügt *werden* und speichert eine Markierung im Protokoll

Examples

Grundlegendes Transaktionsgerüst mit Fehlerbehandlung

```
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
```

Transaktionsabwicklung online lesen: https://riptutorial.com/de/sql-server/topic/5859/transaktionsabwicklung

Kapitel 99: Transaktionsisolationsstufen

Syntax

- SET TRANSACTION ISOLATION LEVEL {READ UNCOMMITTED | LESEN BESTIMMT | Wiederholbares Lesen | SNAPSHOT | SERIALIZABLE} [;]

Bemerkungen

MSDN-Referenz: SET TRANSACTION ISOLATION LEVEL

Examples

Lesen Sie unverbindlich

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

Dies ist die am meisten zulässige Isolationsstufe, da sie keinerlei Sperren verursacht. Sie gibt an, dass Anweisungen alle Zeilen lesen können, einschließlich Zeilen, die in Transaktionen geschrieben wurden, aber noch nicht festgeschrieben sind (dh sie befinden sich noch in der Transaktion). Diese Isolationsstufe kann "Dirty Reads" unterliegen.

Lesen Sie Committed

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

Diese Isolationsstufe ist die zweithöchste. Es verhindert schmutziges Lesen. Das Verhalten von `READ COMMITTED` hängt von der Einstellung von `READ_COMMITTED_SNAPSHOT` :

- Wenn die Einstellung auf OFF (Standardeinstellung) gesetzt ist, verwendet die Transaktion gemeinsame Sperren, um zu verhindern, dass andere Transaktionen die von der aktuellen Transaktion verwendeten Zeilen ändern. Außerdem wird die aktuelle Transaktion daran gehindert, von anderen Transaktionen modifizierte Zeilen zu lesen.
- Wenn diese `READCOMMITTEDLOCK` auf ON gesetzt ist, kann der `READCOMMITTEDLOCK` Tabellenhinweis verwendet werden, um anstelle der Zeilenversionierung für Transaktionen, die im Modus `READ COMMITTED` werden, gemeinsames Sperren anzufordern.

Hinweis: `READ COMMITTED` ist das Standardverhalten von SQL Server.

Was sind "Dirty Reads"?

Schmutzige Lesevorgänge (oder nicht festgeschriebene Lesevorgänge) sind Lesevorgänge von Zeilen, die von einer offenen Transaktion geändert werden.

Dieses Verhalten kann mithilfe von zwei separaten Abfragen repliziert werden: eine, um eine Transaktion zu öffnen und einige Daten in eine Tabelle zu schreiben, ohne zu verpflichten.

Abfrage 1 - Bereiten Sie eine Transaktion vor, aber beenden Sie sie nicht:

```
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
```

Abfrage 2 - Lesen Sie die Zeilen einschließlich der offenen Transaktion:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM dbo.demo;
```

Kehrt zurück:

```
col1        col2
----------- ---------------------------------------
99          Normal transaction
42          Dirty read
```

PS: Vergessen Sie nicht, diese Demo-Daten aufzuräumen:

```
COMMIT TRANSACTION;
DROP TABLE dbo.demo;
GO
```

Wiederholbares Lesen

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

Diese Transaktionsisolationsstufe ist etwas weniger zulässig als `READ COMMITTED` , da gemeinsam genutzte Sperren für alle von jeder Anweisung in der Transaktion gelesenen Daten gesetzt und **bis zum Abschluss der Transaktion** gehalten werden, anstatt nach jeder Anweisung freigegeben zu werden.

Anmerkung: Verwenden Sie diese Option nur, wenn dies erforderlich ist, da dies eher zu einer Beeinträchtigung der Datenbankleistung und zu Deadlocks als zu `READ COMMITTED` .

Schnappschuss

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
```

Gibt an, dass Daten, die von einer Anweisung in einer Transaktion gelesen werden, die transaktionskonsistente Version der Daten sind, die zu Beginn der Transaktion vorhanden waren, dh, dass nur Daten gelesen werden, die vor dem Beginn der Transaktion festgeschrieben wurden.

`SNAPSHOT` Transaktionen fordern keine Sperren für die gelesenen Daten an und verursachen keine Sperren, da sie nur die Version (oder Momentaufnahme) der Daten lesen, die zum Zeitpunkt des Transaktionsbeginns vorhanden waren.

Eine Transaktion, die in der `SNAPSHOT` Isolationsstufe ausgeführt wird, liest nur ihre eigenen Datenänderungen, während sie ausgeführt wird. Eine Transaktion könnte beispielsweise einige Zeilen aktualisieren und dann die aktualisierten Zeilen lesen. Diese Änderung ist jedoch nur für die aktuelle Transaktion sichtbar, bis sie festgeschrieben wird.

Anmerkung: Die Datenbankoption `ALLOW_SNAPSHOT_ISOLATION` muss auf ON gesetzt sein, bevor die Isolationsstufe `SNAPSHOT` verwendet werden kann.

Serialisierbar

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SERIALIZEABLE
```

Diese Isolationsstufe ist am restriktivsten. Er fordert **Bereichssperren an, die** den Bereich der Schlüsselwerte sperren, die von jeder Anweisung in der Transaktion gelesen werden. Dies bedeutet auch, dass `INSERT` Anweisungen aus anderen Transaktionen gesperrt werden, wenn die einzufügenden Zeilen in dem von der aktuellen Transaktion gesperrten Bereich liegen.

Diese Option hat den gleichen Effekt wie das Festlegen von `HOLDLOCK` für alle Tabellen in allen `SELECT` Anweisungen in einer Transaktion.

Hinweis: Diese Transaktionsisolation hat die geringste Parallelität und sollte nur bei Bedarf verwendet werden.

Transaktionsisolationsstufen online lesen: https://riptutorial.com/de/sql-server/topic/5114/transaktionsisolationsstufen

Kapitel 100: ÜBER Klausel

Parameter

Parameter	Einzelheiten
PARTITION VON	Das Feld bzw. die Felder, die auf PARTITION BY folgen, sind die, auf denen die 'Gruppierung' basiert

Bemerkungen

Die OVER-Klausel bestimmt ein Fenster oder eine Teilmenge einer Zeile innerhalb einer Abfrageergebnismenge. Eine Fensterfunktion kann angewendet werden, um einen Wert für jede Zeile in der Gruppe festzulegen und zu berechnen. Die OVER-Klausel kann verwendet werden mit:

- Ranking-Funktionen
- Aggregatfunktionen

So kann jemand aggregierte Werte berechnen, z. B. gleitende Durchschnitte, kumulierte Aggregate, laufende Summen oder ein Top N pro Gruppe.

In sehr abstrakter Weise können wir sagen, dass sich OVER wie GROUP BY verhält. OVER wird jedoch pro Feld / Spalte angewendet und nicht als Ganzes auf die Abfrage wie bei GROUP BY.

Hinweis 1: In SQL Server 2008 (R2) kann die ORDER BY-Klausel nicht mit Aggregatfensterfunktionen (Verknüpfung) verwendet werden.

Examples

Aggregationsfunktionen mit OVER verwenden

Anhand der Autos-Tabelle berechnen wir den Gesamtbetrag, die Höchst-, Mindest- und Durchschnittsbeträge, die jeder Kunde ausgegeben hat, und er hat (COUNT) viele Male (COUNT), die er mit einem Auto zur Reparatur gebracht hat.

Id CustomerId MechanicId-Modellstatus Gesamtkosten

```
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'
```

Beachten Sie, dass bei Verwendung von OVER auf diese Weise die zurückgegebenen Zeilen nicht zusammengefasst werden. Die obige Abfrage gibt Folgendes zurück:

Kundennummer	Gesamt	Durchschn	Anzahl	Mindest	Max
1	430	215	2	200	230
1	430	215	2	200	230

Die duplizierten Zeilen sind möglicherweise nicht für Berichtszwecke nützlich.

Wenn Sie einfach Daten zusammenfassen möchten, können Sie die GROUP BY-Klausel zusammen mit den entsprechenden Aggregatfunktionen verwenden. ZB:

```
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
```

Kumulierte Summe

Anhand der Artikelverkaufstabelle versuchen wir herauszufinden, wie der Verkauf unserer Artikel im Laufe der Zeit steigt. Dazu berechnen wir die *kumulierte Summe* des Gesamtumsatzes pro Artikelbestellung bis zum Verkaufsdatum.

```
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
```

Aggregationsfunktionen verwenden, um die neuesten Datensätze zu finden

Mit der Bibliotheksdatenbank versuchen wir, das letzte der Datenbank hinzugefügte Buch für jeden Autor zu finden. Für dieses einfache Beispiel nehmen wir für jeden hinzugefügten Datensatz eine immer inkrementierende ID an.

```
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
```

Anstelle von RANK können zwei andere Funktionen zum Bestellen verwendet werden. Im

vorherigen Beispiel ist das Ergebnis dasselbe, aber sie ergeben unterschiedliche Ergebnisse, wenn die Reihenfolge mehrere Reihen für jeden Rang ergibt.

- `RANK()` : Duplikate erhalten denselben Rang, der nächste Rang berücksichtigt die Anzahl der Duplikate im vorherigen Rang
- `DENSE_RANK()` : Duplikate erhalten denselben Rang, der nächste Rang ist immer um einen höheren als der vorherige
- `ROW_NUMBER()` : Jede Zeile erhält einen eindeutigen 'Rang', der die Duplikate zufällig `ROW_NUMBER()`

Wenn die Tabelle beispielsweise über eine nicht eindeutige Spalte CreationDate verfügt und darauf basierend die Reihenfolge festgelegt wurde, wird die folgende Abfrage angezeigt:

```
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
```

Könnte dazu führen:

Autor	Titel	Erstellungsdatum	RANG	DENSE_RANK	ZEILENNUMMER
Autor 1	Buch 1	22/07/2016	1	1	1
Autor 1	Buch 2	22/07/2016	1	1	2
Autor 1	Buch 3	21/07/2016	3	2	3
Autor 1	Buch 4	21/07/2016	3	2	4
Autor 1	Buch 5	21/07/2016	3	2	5
Autor 1	Buch 6	07.04.2016	6	3	6
Autor 2	Buch 7	07.04.2016	1	1	1

Teilen von Daten in gleich partitionierte Buckets mit NTILE

Nehmen wir an, Sie haben Prüfungsergebnisse für mehrere Prüfungen und möchten diese pro Quartal in Quartile unterteilen.

```
-- 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` funktioniert `ntile` wenn Sie wirklich eine bestimmte Anzahl von Eimern benötigen und jedes auf ungefähr das gleiche Niveau gefüllt ist. Beachten Sie, dass es trivial wäre, diese Werte durch einfaches Verwenden von `ntile(100)` in Perzentile zu trennen.

ÜBER Klausel online lesen: https://riptutorial.com/de/sql-server/topic/353/uber-klausel

Kapitel 101: UNION

Examples

Union und Union alle

Union- Vorgang kombiniert die Ergebnisse von zwei oder mehr Abfragen in einer einzigen Ergebnismenge, die alle Zeilen enthält, die zu allen Abfragen in der Union gehören, und ignoriert alle vorhandenen Duplikate. **Union all** macht dasselbe, schließt aber auch die doppelten Werte ein. Das Konzept der Gewerkschaftsoperation wird aus dem nachstehenden Beispiel deutlich. Bei der Verwendung von union sind nur wenige Punkte zu beachten:

1. Die Anzahl und die Reihenfolge der Spalten müssen in allen Abfragen gleich sein.

2.Die Datentypen müssen kompatibel sein.

Beispiel:

Wir haben drei Tabellen: Marksheet1, Marksheet2 und Marksheet3. Marksheet3 ist die doppelte Tabelle von Marksheet2, die dieselben Werte wie die von Marksheet2 enthält.

Table1 : Marksheet1

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

Table2 : Marksheet2

CourseCode	CourseName	MarksObtained
201	PhysicsII	82
202	ChemistryII	86
203	MathsII	95
204	EnglishII	70
205	ComputerII	86

Table3 : Marksheet3

SubjectCode	SubjectName	MarksObtained
201	PhysicsII	82
202	ChemistryII	86
203	MathsII	95
204	EnglishII	70
205	ComputerII	86

Union auf Tabellen Marksheet1 und Marksheet2

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
```

Hinweis: Die Ausgabe für die Vereinigung der drei Tabellen entspricht der Vereinigung in Marksheet1 und Marksheet2, da die Vereinigungsoperation keine doppelten Werte erfordert.

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

AUSGABE

	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 All

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION ALL
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION ALL
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

AUSGABE

	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

Sie werden hier feststellen, dass die doppelten Werte aus Marksheet3 ebenfalls mit union all angezeigt werden.

UNION online lesen: https://riptutorial.com/de/sql-server/topic/5590/union

Kapitel 102: Unterabfragen

Examples

Unterabfragen

Eine Unterabfrage ist eine Abfrage in einer anderen SQL-Abfrage. Eine Unterabfrage wird auch als innere Abfrage oder innere Auswahl bezeichnet. Die Anweisung, die eine Unterabfrage enthält, wird als äußere Abfrage oder äußere Auswahl bezeichnet.

Hinweis

1. Unterabfragen müssen in Klammern stehen.
2. Ein ORDER BY kann nicht in einer Unterabfrage verwendet werden.
3. Der Bildtyp wie BLOB, Array, Textdatentypen ist in Unterabfragen nicht zulässig.

Unterabfragen können mit select-, insert-, update- und delete-Anweisungen verwendet werden, in denen die select-Klausel zusammen mit IN, Vergleichsoperatoren usw. enthalten ist.

Wir haben eine Tabelle namens ITCompanyInNepal, in der Abfragen ausgeführt werden, um Beispiele für Unterabfragen anzuzeigen:

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

Beispiele: **SubQueries mit Select-Anweisung**

mit **In** Operator und **Where-** Klausel:

```
SELECT *
FROM ITCompanyInNepal
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'USA');
```

mit **Vergleichsoperator** und **Where-** Klausel

```
SELECT *
FROM ITCompanyInNepal
WHERE NumberOfEmployee < (SELECT AVG(NumberOfEmployee)
```

```
                        FROM ITCompanyInNepal
                    )
```

mit **Select-** Klausel

```
SELECT   CompanyName,
         CompanyAddress,
         Headquarter,
         (Select SUM(NumberOfEmployee)
         FROM ITCompanyInNepal
         Where Headquarter = 'USA') AS TotalEmployeeHiredByUSAInKathmandu
FROM     ITCompanyInNepal
WHERE    CompanyAddress = 'Kathmandu' AND Headquarter = 'USA'
```

Unterabfragen mit insert-Anweisung

Wir müssen Daten aus der IndianCompany-Tabelle in ITCompanyInNepal einfügen. Die Tabelle für IndianCompany wird unten gezeigt:

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
```

Unterabfragen mit Update-Anweisung

Angenommen, alle Unternehmen mit Hauptsitz in den USA haben sich entschlossen, 50 Angestellte aller US-amerikanischen Unternehmen in Nepal zu entlassen, weil sich die Politik der US-Unternehmen geändert hat.

```
UPDATE ITCompanyInNepal
SET NumberOfEmployee = NumberOfEmployee - 50
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'USA')
```

Unterabfragen mit Delete-Anweisung

Angenommen, alle Unternehmen mit Hauptsitz in Dänemark haben beschlossen, ihre Unternehmen aus Nepal zu schließen.

```
DELETE FROM ITCompanyInNepal
WHERE Headquarter IN (SELECT Headquarter
                     FROM ITCompanyInNepal
                     WHERE Headquarter = 'Denmark')
```

Unterabfragen online lesen: https://riptutorial.com/de/sql-server/topic/5629/unterabfragen

Kapitel 103: Variablen

Syntax

- DECLARE @VariableName DataType [= Wert];
- SET @VariableName = Wert;

Examples

Deklarieren Sie eine Tabellenvariable

```
DECLARE @Employees TABLE
(
    EmployeeID INT NOT NULL PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    ManagerID INT NULL
)
```

Wenn Sie eine normale Tabelle erstellen, verwenden Sie die Syntax `CREATE TABLE Name (Columns)`. Beim Erstellen einer Tabellenvariablen verwenden Sie die Syntax `DECLARE @Name TABLE (Columns)`.

Um auf die Tabellenvariable innerhalb einer `SELECT` Anweisung zu verweisen, müssen Sie in SQL Server einen Alias für die Tabellenvariable angeben. Andernfalls erhalten Sie eine Fehlermeldung:

> Muss die Skalarvariable "@TableVariableName" deklarieren.

dh

```
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
```

Eine Variable mit SET aktualisieren

```
DECLARE @VariableName INT
SET @VariableName = 1
PRINT @VariableName
```

1

Mit `SET` können Sie jeweils nur eine Variable aktualisieren.

Variablen mit SELECT aktualisieren

Mit `SELECT` können Sie mehrere Variablen gleichzeitig aktualisieren.

```
DECLARE @Variable1 INT, @Variable2 VARCHAR(10)
SELECT @Variable1 = 1, @Variable2 = 'Hello'
PRINT @Variable1
PRINT @Variable2
```

1

Hallo

Bei der Verwendung von `SELECT` zum Aktualisieren einer Variablen aus einer Tabellenspalte wird bei mehreren Werten der *letzte* Wert verwendet. (Es gelten die normalen Bestellregeln. Wenn keine Sortierung angegeben wird, ist die Bestellung nicht garantiert.)

```
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

Wenn von der Abfrage keine Zeilen zurückgegeben werden, ändert sich der Wert der Variablen nicht:

```
SELECT TOP 0 @Variable = Example
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

1

Mehrere Variablen gleichzeitig mit Anfangswerten deklarieren

```
DECLARE
  @Var1 INT = 5,
  @Var2 NVARCHAR(50) = N'Hello World',
  @Var3 DATETIME = GETDATE()
```

Zusammengesetzte Zuweisungsoperatoren

SQL Server 2008 R2

Unterstützte Verbundoperatoren:

- `+=` Hinzufügen und Zuweisen
- `-=` Subtrahieren und zuweisen
- `*=` Multiplizieren und zuweisen
- `/=` Teilen und zuweisen
- `%=` Modulo und zuweisen
- `&=` Bitweises UND und Zuweisen
- `^=` Bitweises XOR und zuweisen
- `|=` Bitweises ODER und zuweisen

Verwendungsbeispiel:

```
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
```

Aktualisieren von Variablen durch Auswahl aus einer Tabelle

Abhängig von der Struktur Ihrer Daten können Sie Variablen erstellen, die dynamisch aktualisiert werden.

```
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)
```

In den meisten Fällen sollten Sie sicherstellen, dass Ihre Abfrage bei Verwendung dieser Methode nur einen Wert zurückgibt.

Variablen online lesen: https://riptutorial.com/de/sql-server/topic/2566/variablen

Kapitel 104: Verschieben und Kopieren von Daten um Tabellen herum

Examples

Daten von einer Tabelle in eine andere kopieren

Dieser Code wählt Daten aus einer Tabelle aus und zeigt sie im Abfrage-Tool (normalerweise SSMS)

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

Dieser Code fügt diese Daten in eine Tabelle ein:

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

Kopieren Sie die Daten in eine Tabelle und erstellen Sie diese sofort

Dieser Code wählt Daten aus einer Tabelle aus:

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

Dieser Code erstellt eine neue Tabelle mit dem Namen `MyNewTable` und `MyNewTable` diese Daten hinzu

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

Verschieben von Daten in eine Tabelle (unter Annahme einer eindeutigen Schlüsselmethode)

Um Daten zu *verschieben* , fügen Sie sie zuerst in das Ziel ein und löschen dann alles, was Sie aus der Quelltabelle eingefügt haben. Dies ist keine normale SQL-Operation, kann aber aufschlussreich sein

Was hast du eingefügt? Normalerweise benötigen Sie in Datenbanken eine oder mehrere Spalten, die Sie zur eindeutigen Identifizierung von Zeilen verwenden können, sodass wir davon ausgehen und diese verwenden.

Diese Anweisung wählt einige Zeilen aus

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

Zuerst fügen wir diese in unsere Zieltabelle ein:

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

`Key1` nun *angenommen wird, dass Datensätze in beiden Tabellen* für `Key1` , `Key2` *eindeutig* `Key1` , `Key2` wir damit Daten aus der `Key2` suchen und löschen

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

Dies funktioniert nur korrekt, wenn `Key1` , `Key2` in beiden Tabellen eindeutig sind

Schließlich wollen wir nicht, dass die Arbeit zur Hälfte erledigt ist. Wenn wir dies in einer Transaktion abschließen, werden entweder alle Daten verschoben oder nichts passiert. Dadurch wird sichergestellt, dass wir die Daten nicht einfügen und die Daten nicht aus der Quelle löschen können.

```
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;
```

Verschieben und Kopieren von Daten um Tabellen herum online lesen: https://riptutorial.com/de/sql-server/topic/1467/verschieben-und-kopieren-von-daten-um-tabellen-herum

Kapitel 105: Verschlüsselung

Parameter

Optionale Parameter	Einzelheiten
`WITH PRIVATE KEY`	Für CREATE CERTIFICATE kann ein privater Schlüssel angegeben werden: `(FILE='D:\Temp\CertTest\private.pvk', DECRYPTION BY PASSWORD = 'password');`

Bemerkungen

Die Erstellung eines DER-Zertifikats funktioniert einwandfrei. Wenn ein Base64-Zertifikat verwendet wird, beschwert sich der SQL Server jedoch mit der kryptischen Nachricht:

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

Importieren Sie Ihr Base64-Zertifikat in den Zertifikatspeicher Ihres Betriebssystems, um es erneut in das binäre Format DER exportieren zu können.

Eine weitere wichtige Sache ist, eine Verschlüsselungshierarchie zu haben, damit die eine bis zur Betriebssystemebene geschützt wird. Siehe den Artikel über 'Verschlüsselung der Datenbank / TDE'.

Weitere Informationen zum Erstellen von Zertifikaten finden Sie unter:
https://msdn.microsoft.com/en-us/library/ms187798.aspx

Weitere Informationen zur Verschlüsselung der Datenbank / TDE finden Sie unter:
https://msdn.microsoft.com/de-de/library/bb934049.aspx

Weitere Informationen zur Verschlüsselung von Daten finden Sie unter:
https://msdn.microsoft.com/en-us/library/ms188061.aspx

Examples

Verschlüsselung nach Zertifikat

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

Erstellen Sie das Zertifikat

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

Normalerweise würden Sie mit einem symmetrischen Schlüssel verschlüsseln. Dieser Schlüssel würde durch den asymmetrischen Schlüssel (öffentlicher Schlüssel) Ihres Zertifikats verschlüsselt.

Beachten Sie außerdem, dass die Verschlüsselung abhängig von der Schlüssellänge auf bestimmte Längen beschränkt ist und ansonsten NULL zurückgibt. Microsoft schreibt: "Die Grenzen sind: Ein 512-Bit-RSA-Schlüssel kann bis zu 53 Byte verschlüsseln, ein 1024-Bit-Schlüssel kann bis zu 117 Byte und ein 2048-Bit-Schlüssel kann bis zu 245 Byte verschlüsseln."

EncryptByAsymKey hat die gleichen Grenzwerte. Für UNICODE würde dies durch 2 (16 Bits pro Zeichen) geteilt, also 58 Zeichen für einen 1024-Bit-Schlüssel.

Verschlüsselung der Datenbank

```
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
```

Dies verwendet 'Transparente Datenverschlüsselung' (TDE).

Verschlüsselung durch symmetrischen Schlüssel

```
-- 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');
```

Verschlüsselung nach Passphrase

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

Dies wird auch verschlüsseln, dann aber durch Passphrase anstelle eines asymmetrischen (Zertifikat) Schlüssels oder durch einen expliziten symmetrischen Schlüssel.

Verschlüsselung online lesen: https://riptutorial.com/de/sql-server/topic/7096/verschlusselung

Kapitel 106: VERSCHMELZEN

Syntax

- COALESCE ([Spalte1], [Spalte2] .... [SpalteN])

Examples

Verwenden von COALESCE zum Erstellen einer durch Kommas getrennten Zeichenfolge

Wir können eine durch Kommas getrennte Zeichenfolge aus mehreren Zeilen mit Hilfe von coalesce erhalten, wie unten gezeigt.

Da die Tabellenvariable verwendet wird, müssen Sie die gesamte Abfrage einmal ausführen. Um es leicht verständlich zu machen, habe ich den BEGIN- und END-Block hinzugefügt.

```
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
```

Coalesce-Basisbeispiel

`COALESCE()` gibt den ersten `NON NULL` Wert in einer Liste von Argumenten zurück. Nehmen wir an, wir hätten eine Tabelle mit Telefonnummern und Handynummern und wollten nur eine für jeden Benutzer zurückgeben. Um nur einen zu erhalten, können wir den ersten `NON NULL` Wert erhalten.

```
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
```

Der erste Wert ist nicht null aus einer Liste von Spaltenwerten

```
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
```

VERSCHMELZEN online lesen: https://riptutorial.com/de/sql-server/topic/3234/verschmelzen

Kapitel 107: VERSCHMELZEN

Einführung

Beginnend mit SQL Server 2008 ist es möglich, Einfüge-, Aktualisierungs- oder Löschvorgänge in einer einzelnen Anweisung mithilfe der MERGE-Anweisung auszuführen.

Mit der MERGE-Anweisung können Sie eine Datenquelle mit einer Zieltabelle oder -sicht verknüpfen und dann basierend auf den Ergebnissen dieses Joins mehrere Aktionen für das Ziel ausführen.

Syntax

- Gemäß MSDN - https://msdn.microsoft.com/en-us/library/bb510625.aspx [WITH <common_table_expression> [, ... n]] MERGE [TOP (Ausdruck) [PERCENT]] [INTO] < target_table> [WITH (<merge_hint>)] [[AS] table_alias] USING <table_source> ON <merge_search_condition> [WHEN MATCHED [AND <Klausel_search_condition>] THEN <merge_matched>] [... n] [WENN NICHT ÜBEREINSTIMMT [] TARGET] [AND <clause_search_condition>] THEN <merge_not_matched>] [WENN NICHT MIT QUELLE [UND <clause_search_condition>] DANN <merge_matched>] [... n] [<output_clause>] [OPTION (<query_hint>]). ..n])]; <target_table> :: = {[Datenbankname]. schema_name. | schema_name.] target_table} <merge_hint> :: = {{[<table_hint_limited> [, ... n]] [[,] INDEX (index_val [, ... n])]}} <table_source> :: = {table_or_view_name [[AS] table_alias] [<tablesample_clause>] [WITH (table_hint [[,] ... n])] | rowset_function [[AS] table_alias] [(bulk_column_alias [, ... n])] | user_defined_function [[AS] table_alias] | OPENXML <openxml_clause> | abgeleitete_Tabelle [AS] table_alias [(column_alias [, ... n])] | <join_table> | <pivoted_table> | <unpivoted_table>} <merge_search_condition> :: = <suchbedingung> <merge_matched> :: = {UPDATE SET <set_clause> | DELETE} <set_clause> :: = SET {Spaltenname = {Ausdruck | STANDARD | NULL} | {udt_column_name. {{property_name = Ausdruck | Feldname = Ausdruck} | Methodenname (Argument [, ... n])}} | Spaltenname {.WRITE (Ausdruck, @Offset, @Length)} | @variable = Ausdruck | @variable = Spalte = Ausdruck | Spaltenname {+ = | - = | * = | / = | % = | & = | ^ = | | =} Ausdruck | @variable {+ = | - = | * = | / = | % = | & = | ^ = | | =} Ausdruck | @variable = Spalte {+ = | - = | * = | / = | % = | & = | ^ = | | =} Ausdruck} [, ... n] <merge_not_matched> :: = {INSERT [(Spaltenliste)] {VALUES (Werte_Liste) | DEFAULT VALUES}} <clause_search_condition> :: = <search_condition> :: = {[NOT] | (<Suchbedingung>)} [{AND | ODER} [NICHT] {| (<Suchbedingung>)}] [, ... n] :: = {Ausdruck {= | <> | ! = |

  | > = | ! > | <| <= | ! <} Ausdruck | Zeichenfolgenausdruck [NOT] LIKE Zeichenfolgenausdruck [ESCAPE 'escape_character'] | Ausdruck [NICHT] ZWISCHEN Ausdruck UND Ausdruck | Ausdruck IST [NICHT] NULL | CONTAINS ({column | *}, '<contains_search_condition>') | FREETEXT ({column | *}, 'freetext_string') | Ausdruck [NOT] IN (Unterabfrage | Ausdruck [, ... n]) | Ausdruck {= | <> | ! = | | > = | ! > | <| <= | ! <} {ALL | EINIGE | ANY} (Unterabfrage) | EXISTS (Unterabfrage)} <output_clause> :: = {[OUTPUT <dml_select_list>

INTO {@table_variable | output_table} [(column_list)]] [OUTPUT <dml_select_list>]} <dml_select_list> :: = {<spaltenname> | scalar_expression} [[AS] column_alias_identifier] [, ... n] <spaltenname> :: = {DELETED | INSERTED | from_table_name}. {* | Spaltenname} | $ action

Bemerkungen

Führt Einfüge-, Aktualisierungs- oder Löschvorgänge für eine Zieltabelle aus, basierend auf den Ergebnissen eines Joins mit einer Quelltabelle. Sie können beispielsweise zwei Tabellen synchronisieren, indem Sie Zeilen in einer Tabelle basierend auf den in der anderen Tabelle gefundenen Unterschieden einfügen, aktualisieren oder löschen.

Examples

MERGE zum Einfügen / Aktualisieren / Löschen

```
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
```

Beschreibung:

- `MERGE INTO targetTable` - zu `MERGE INTO targetTable` Tabelle
- `USING sourceTable` - Datenquelle (kann eine Tabelle oder eine Ansicht oder eine Funktion mit Tabellenwert sein)
- `ON ...` - Join-Bedingung zwischen `targetTable` und `sourceTable` .
- `WHEN MATCHED` - Aktionen, die ausgeführt werden sollen, wenn eine Übereinstimmung gefunden wird
- - `AND (targetTable.PKID > 100)` - zusätzliche Bedingung (en), die erfüllt sein müssen, damit die Aktion ausgeführt werden kann
- `THEN DELETE` - Löscht den übereinstimmenden Datensatz aus der `targetTable`
- `THEN UPDATE` - Aktualisieren Sie die Spalten des übereinstimmenden Datensatzes, der mit `SET ....` angegeben wurde `SET ....`
- `WHEN NOT MATCHED` - Aktionen, die ausgeführt werden sollen, wenn in **`targetTable`** keine

Übereinstimmung gefunden **`targetTable`**
- `WHEN NOT MATCHED BY SOURCE` - Aktionen, die ausgeführt werden sollen, wenn keine **`sourceTable`** in **`sourceTable`**

Bemerkungen:

Wenn eine bestimmte Aktion nicht erforderlich ist, lassen Sie die Bedingung aus, z. B. das Entfernen von `WHEN NOT MATCHED THEN INSERT` verhindert, dass Datensätze eingefügt werden

Merge-Anweisung erfordert ein abschließendes Semikolon.

Beschränkungen:

- `WHEN MATCHED` erlaubt keine `INSERT` Aktion
- `UPDATE` Aktion `UPDATE` kann eine Zeile nur einmal aktualisieren. Dies bedeutet, dass die Join-Bedingung eindeutige Übereinstimmungen erzeugen muss.

Mit CTE-Quelle zusammenführen

```
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);
```

MERGE mithilfe der abgeleiteten Quelltabelle

```
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);
```

Zusammenführungsbeispiel - Quelle und Zieltabelle synchronisieren

Beachten Sie zur Veranschaulichung der MERGE-Anweisung die folgenden zwei Tabellen:

1. **dbo.Product** : Diese Tabelle enthält Informationen zu dem Produkt, das das Unternehmen derzeit verkauft

2. **dbo.ProductNew** : Diese Tabelle enthält Informationen über das Produkt, das das Unternehmen zukünftig verkaufen wird.

Das folgende T-SQL erstellt diese beiden Tabellen und füllt sie auf

```
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)
```

Nehmen wir an, wir wollen die Tabelle dbo.Product Target mit der Tabelle dbo.ProductNew synchronisieren. Hier ist das Kriterium für diese Aufgabe:

1. Produkte, die sowohl in der Quelltabelle dbo.ProductNew als auch in der Zieltabelle dbo.Product vorhanden sind, werden in der Zieltabelle dbo.Product mit neuen neuen Produkten aktualisiert.

2. Alle Produkte in der dbo.ProductNew-Quelltabelle, die nicht in der Zieltabelle dob.Product enthalten sind, werden in die Zieltabelle dbo.Product eingefügt.

3. Jedes Produkt in der Zieltabelle dbo.Product, das nicht in der Quelltabelle dbo.ProductNew vorhanden ist, muss aus der Zieltabelle dbo.Product gelöscht werden. Hier ist die MERGE-Anweisung, um diese Aufgabe auszuführen.

```
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.*;
```

Hinweis: Das Semikolon muss am Ende der MERGE-Anweisung stehen.

	$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

Zusammenführen mit EXCEPT

Verwenden Sie EXCEPT, um Aktualisierungen an unveränderten Datensätzen zu verhindern

```
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;
```

VERSCHMELZEN online lesen: https://riptutorial.com/de/sql-server/topic/4550/verschmelzen

Kapitel 108: VERSUCHEN / FANGEN

Bemerkungen

TRY / CATCH ist ein für T SQL von MS SQL Server spezifisches Sprachkonstrukt.

Es ermöglicht die Fehlerbehandlung innerhalb von T-SQL, ähnlich wie in .NET-Code.

Examples

Transaktion in einem TRY / CATCH

Dadurch werden beide Inserts aufgrund einer ungültigen Datumszeit zurückgesetzt:

```
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
```

Dadurch werden beide Einfügungen festgelegt:

```
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
```

Fehler beim Try-Catch-Block

Die RAISERROR-Funktion generiert einen Fehler im TRY CATCH-Block:

```
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 mit einem zweiten Parameter größer als 10 (in diesem Beispiel 11) stoppt die Ausführung in TRY BLOCK und löst einen Fehler aus, der im CATCH-Block behandelt wird. Sie können mit der Funktion ERROR_MESSAGE () auf die Fehlernachricht zugreifen. Ausgabe dieses Beispiels ist:

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

Infomeldungen werden in try catch block ausgelöst

RAISERROR mit einem Schweregrad (zweiter Parameter) kleiner oder gleich 10 löst keine Ausnahme aus.

```
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
```

Nach der RAISERROR-Anweisung wird die dritte Anweisung ausgeführt und der CATCH-Block wird nicht aufgerufen. Ergebnis der Ausführung ist:

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

Erneuter Ausnahmefehler, der von RAISERROR generiert wurde

Sie können Fehler, die Sie im CATCH-Block abfangen, mit der TRHOW-Anweisung erneut werfen:

```
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
```

Beachten Sie, dass in diesem Fall ein Fehler mit formatierten Argumenten (vierter und fünfter Parameter) auftritt. Dies kann nützlich sein, wenn Sie der Nachricht weitere Informationen

hinzufügen möchten. Ergebnis der Ausführung ist:

```
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'
```

Ausnahme in TRY / CATCH-Blöcken

Sie können in try catch block eine Ausnahme auslösen:

```
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
```

Ausnahme mit im CATCH-Block behandelt werden und dann mit THROW ohne Parameter erneut ausgelöst werden.

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

THROW ähnelt RAISERROR mit folgenden Unterschieden:

- Es wird empfohlen, dass bei neuen Anwendungen THROW anstelle von RASIERROR verwendet wird.
- THROW kann eine beliebige Zahl als erstes Argument (Fehlernummer) verwenden, RAISERROR kann nur IDs in der Sicht sys.messages verwenden
- THROW hat Schweregrad 16 (kann nicht geändert werden)
- THROW kann keine Argumente wie RAISERROR formatieren. Verwenden Sie die Funktion FORMATMESSAGE als Argument von RAISERROR, wenn Sie diese Funktion benötigen.

VERSUCHEN / FANGEN online lesen: https://riptutorial.com/de/sql-server/topic/5189/versuchen---fangen

Kapitel 109: Verwalten der Azure SQL-Datenbank

Examples

Suchen Sie nach Service Tier-Informationen für die Azure SQL-Datenbank

Die Azure SQL-Datenbank verfügt über verschiedene Editionen und Leistungsstufen.

Sie können Version, Edition (Basis, Standard oder Premium) und Serviceziel (S0, S1, P4, P11 usw.) der SQL-Datenbank finden, die als Dienst in Azure ausgeführt wird, und zwar mit folgenden Anweisungen:

```
select @@version
SELECT DATABASEPROPERTYEX('Wwi', 'EDITION')
SELECT DATABASEPROPERTYEX('Wwi', 'ServiceObjective')
```

Ändern Sie die Serviceebene der Azure SQL-Datenbank

Sie können die Azure SQL-Datenbank mit der Anweisung ALTER DATABASE skalieren oder verkleinern:

```
ALTER DATABASE WWI
MODIFY (SERVICE_OBJECTIVE = 'P6')
-- or
ALTER DATABASE CURRENT
MODIFY (SERVICE_OBJECTIVE = 'P2')
```

Wenn Sie versuchen, den Service-Level zu ändern, während der Service-Level der aktuellen Datenbank geändert wird, wird der folgende Fehler angezeigt:

> Meldung 40802, Ebene 16, Status 1, Zeile 1 Eine Dienstzielzuweisung auf Server '.....' und Datenbank '.....' ist bereits in Bearbeitung. Bitte warten Sie, bis der Zuordnungsstatus der Service-Ziele für die Datenbank als "Abgeschlossen" markiert ist.

Führen Sie Ihre ALTER DATABASE-Anweisung erneut aus, wenn der Übergangszeitraum abgeschlossen ist.

Replikation der Azure SQL-Datenbank

Sie können ein sekundäres Replikat einer Datenbank mit demselben Namen auf einem anderen Azure SQL Server erstellen, wodurch die lokale Datenbank primär wird, und beginnt mit der asynchronen Replikation von Daten vom primären zum neuen sekundären.

```
ALTER DATABASE <<mydb>>
```

```
ADD SECONDARY ON SERVER <<secondaryserver>>
WITH ( ALLOW_CONNECTIONS = ALL )
```

Der Zielserver kann sich in einem anderen Datencenter befinden (für Geo-Replikation verwendbar). Wenn auf dem Zielserver bereits eine Datenbank mit demselben Namen vorhanden ist, schlägt der Befehl fehl. Der Befehl wird in der master-Datenbank auf dem Server ausgeführt, der die lokale Datenbank hostet, die als primäre Datenbank fungiert. Wenn ALLOW_CONNECTIONS auf ALL gesetzt ist (standardmäßig auf NO gesetzt), handelt es sich bei der sekundären Reproduktion um eine schreibgeschützte Datenbank, die allen Anmeldungen mit den entsprechenden Berechtigungen die Verbindung zulässt.

Sekundäre Datenbankreplikate können mit dem folgenden Befehl zu einer primären Datenbank hochgestuft werden:

```
ALTER DATABASE mydb FAILOVER
```

Sie können die sekundäre Datenbank auf dem sekundären Server entfernen:

```
ALTER DATABASE <<mydb>>
REMOVE SECONDARY ON SERVER <<testsecondaryserver>>
```

Erstellen Sie eine Azure SQL-Datenbank im Elastic-Pool

Sie können Ihre azure SQL-Datenbank in einen elastischen SQL-Pool einfügen:

```
CREATE DATABASE wwi
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

Sie können eine Kopie einer vorhandenen Datenbank erstellen und diese in einem elastischen Pool ablegen:

```
CREATE DATABASE wwi
AS COPY OF myserver.WideWorldImporters
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

Verwalten der Azure SQL-Datenbank online lesen: https://riptutorial.com/de/sql-server/topic/7113/verwalten-der-azure-sql-datenbank

Kapitel 110: Verwendung der TEMP-Tabelle

Bemerkungen

Temporäre Tische sind wirklich sehr hilfreich.

Die Tabelle kann zur Laufzeit erstellt werden und kann alle Operationen ausführen, die in einer normalen Tabelle ausgeführt werden.

Diese Tabellen werden in einer Tempdb-Datenbank erstellt.

Wann verwendet?

1. Wir müssen komplexe Join-Operationen durchführen.
2. Wir führen eine große Anzahl von Zeilenmanipulationen in gespeicherten Prozeduren durch.
3. Kann die Verwendung des Cursors ersetzen.

Dadurch wird die Leistung erhöht.

Examples

Lokale Temp-Tabelle

- Ist verfügbar, bis die aktuelle Verbindung für den Benutzer bestehen bleibt.

  Wird automatisch gelöscht, wenn der Benutzer die Verbindung trennt.

  Der Name sollte mit # beginnen (#temp)

  ```
  CREATE TABLE #LocalTempTable(
                 StudentID      int,
                 StudentName    varchar(50),
                 StudentAddress varchar(150))
  ```

```
insert into #LocalTempTable values ( 1, 'Ram','India');

select * from #LocalTempTable
```

Nachdem Sie alle diese Anweisungen ausgeführt haben, schließen Sie das Abfragefenster und öffnen es erneut

```
“Invalid object name #LocalTempTable”
```

Globale Temp-Tabelle

- Beginnt mit ## (## temp).

  Wird nur gelöscht, wenn der Benutzer alle Verbindungen trennt.

  Es verhält sich wie ein fester Tisch.

```
CREATE TABLE ##NewGlobalTempTable(
               StudentID      int,
               StudentName    varchar(50),
               StudentAddress varchar(150))

Insert Into ##NewGlobalTempTable values ( 1,'Ram','India');
Select * from ##NewGlobalTempTable
```

Hinweis: Diese können von allen Benutzern der Datenbank unabhängig von der Berechtigungsstufe angezeigt werden.

Temp-Tabellen löschen

Temporäre Tabellen müssen eindeutige IDs haben (innerhalb der Sitzung, bei lokalen temporären Tabellen oder innerhalb des Servers für globale temporäre Tabellen). Wenn Sie versuchen, eine Tabelle mit einem bereits vorhandenen Namen zu erstellen, wird der folgende Fehler zurückgegeben:

```
There is already an object named '#tempTable' in the database.
```

Wenn Ihre Abfrage temporäre Tabellen erzeugt und Sie sie mehr als einmal ausführen möchten, müssen Sie die Tabellen löschen, bevor Sie versuchen, sie erneut zu generieren. Die grundlegende Syntax dafür ist:

```
drop table #tempTable
```

Wenn Sie versuchen, diese Syntax auszuführen, bevor die Tabelle existiert (z. B. beim ersten Durchlauf Ihrer Syntax), wird ein anderer Fehler ausgegeben:

```
Cannot drop the table '#tempTable', because it does not exist or you do not have permission.
```

Um dies zu vermeiden, können Sie vor dem Löschen überprüfen, ob die Tabelle bereits vorhanden ist:

```
IF OBJECT_ID ('tempdb..#tempTable', 'U') is not null DROP TABLE #tempTable
```

Verwendung der TEMP-Tabelle online lesen: https://riptutorial.com/de/sql-server/topic/5328/verwendung-der-temp-tabelle

Kapitel 111: Volltextindizierung

Examples

A. Erstellen eines eindeutigen Indexes, eines Volltextkatalogs und eines Volltextindex

Im folgenden Beispiel wird ein eindeutiger Index für die Spalte JobCandidateID der Tabelle HumanResources.JobCandidate der AdventureWorks2012-Beispieldatenbank erstellt. Das Beispiel erstellt dann einen Standard-Volltextkatalog, ft. Schließlich erstellt das Beispiel einen Volltextindex für die Spalte Resume, wobei der ft-Katalog und die Systemstopliste verwendet werden.

```
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/de-de/library/cc879306.aspx

https://msdn.microsoft.com/de-de/library/ms142571.aspx

Erstellen eines Volltextindex für mehrere Tabellenspalten

```
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
```

Erstellen eines Volltextindex mit einer Sucheigenschaftenliste, ohne ihn aufzufüllen

```
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
```

Und später mit bevölkern

```
ALTER FULLTEXT INDEX ON Production.Document SET CHANGE_TRACKING AUTO;
GO
```

Volltextsuche

```
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;
```

Weitere und detaillierte Informationen erhalten Sie unter https://msdn.microsoft.com/de-de/library/ms142571.aspx

Volltextindizierung online lesen: https://riptutorial.com/de/sql-server/topic/4557/volltextindizierung

Kapitel 112: While-Schleife

Bemerkungen

Die Verwendung einer `WHILE` Schleife oder eines anderen iterativen Prozesses ist normalerweise nicht die effizienteste Art, Daten in SQL Server zu verarbeiten.

Sie sollten es vorziehen, eine satzbasierte Abfrage für die Daten zu verwenden, um möglichst dieselben Ergebnisse zu erzielen

Examples

While-Schleife verwenden

Die `WHILE` Schleife kann als Alternative zu `CURSORS` . Im folgenden Beispiel werden Zahlen von 0 bis 99 gedruckt.

```
DECLARE @i int = 0;
WHILE(@i < 100)
BEGIN
   PRINT @i;
   SET @i = @i+1
END
```

While-Schleife mit Min-Aggregatfunktion

```
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
```

While-Schleife online lesen: https://riptutorial.com/de/sql-server/topic/4249/while-schleife

Kapitel 113: Zeittabellen

Bemerkungen

SQL Server 2016 führt die Unterstützung für temporäre Tabellen mit Systemversion als Datenbankfeature ein, die integrierte Informationen zur Verfügung stellt, um Informationen zu in der Tabelle gespeicherten Daten zu einem beliebigen Zeitpunkt bereitzustellen, und nicht nur die zum aktuellen Zeitpunkt richtigen Daten.

Eine systemversionierte temporale Tabelle ist eine neue Art von Benutzertabelle in SQL Server 2016, die einen vollständigen Verlauf der Datenänderungen aufzeichnet und eine einfache Zeitpunktanalyse ermöglicht. Diese Art von temporärer Tabelle wird als systemversionierte temporale Tabelle bezeichnet, da der Gültigkeitszeitraum für jede Zeile vom System (dh der Datenbank-Engine) verwaltet wird. Jede temporale Tabelle hat zwei explizit definierte Spalten mit jeweils einem datetime2-Datentyp. Diese Spalten werden als Periodenspalten bezeichnet. Diese Zeitraumspalten werden ausschließlich vom System verwendet, um den Gültigkeitszeitraum für jede Zeile zu erfassen, wenn eine Zeile geändert wird.

Examples

Zeittabellen erstellen

```
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: Bei einem **INSERT** setzt das System den Wert für die **ValidFrom-** Spalte auf die **Anfangszeit** der aktuellen Transaktion (in der UTC-Zeitzone) basierend auf der Systemuhr und weist den Wert für die **ValidTo-** Spalte dem Maximalwert von 9999- zu. 12-31. Dies kennzeichnet die Reihe als offen.

UPDATES: Bei einem **UPDATE** speichert das System den vorherigen Wert der Zeile in der **Protokolltabelle** und setzt den Wert für die Spalte **ValidTo** auf den **Anfangszeitpunkt** der aktuellen Transaktion (in der UTC-Zeitzone) basierend auf der Systemuhr. Dies kennzeichnet die Zeile als geschlossen, wobei ein Zeitraum aufgezeichnet wird, für den die Zeile gültig war. In der aktuellen Tabelle wird die Zeile mit ihrem neuen Wert aktualisiert, und das System setzt den Wert für die Spalte **ValidFrom** auf die **Anfangszeit** der Transaktion (in der UTC-Zeitzone) basierend

auf der Systemuhr. Der Wert für die aktualisierte Zeile in der aktuellen Tabelle für die **ValidTo-**Spalte bleibt der Maximalwert von 9999-12-31.

DELETES : Bei einem **DELETE** speichert das System den vorherigen Wert der Zeile in der **Protokolltabelle** und setzt den Wert für die Spalte **ValidTo** auf den **Anfangszeitpunkt** der aktuellen Transaktion (in der UTC-Zeitzone) basierend auf der Systemuhr. Dies kennzeichnet die Zeile als geschlossen, wobei ein Zeitraum aufgezeichnet wird, für den die vorherige Zeile gültig war. In der aktuellen Tabelle wird die Zeile entfernt. Abfragen der aktuellen Tabelle geben diese Zeile nicht zurück. Nur Abfragen, die sich auf Verlaufsdaten beziehen, geben Daten zurück, für die eine Zeile geschlossen ist.

MERGE : Bei einer **MERGE** verhält sich die Operation genau so, als würden bis zu drei Anweisungen (**INSERT** , **UPDATE** und / oder **DELETE**) ausgeführt, je nachdem, was als Aktionen in der **MERGE-** Anweisung angegeben ist.

Tipp: Die in den Systemdatetime2-Spalten aufgezeichneten Zeiten basieren auf der Anfangszeit der Transaktion. Beispielsweise wird für alle Zeilen, die in eine einzige Transaktion eingefügt werden, dieselbe UTC-Zeit in der Spalte aufgezeichnet, die dem Beginn des **SYSTEM_TIME-**Zeitraums entspricht.

Wie frage ich zeitliche Daten ab?

```
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;
```

Gibt den angegebenen Zeitpunkt zurück (FOR SYSTEM_TIME AS of))

Gibt eine Tabelle mit Zeilen zurück, die die Werte enthalten, die zum angegebenen Zeitpunkt in der Vergangenheit aktuell waren (aktuell).

```
SELECT * FROM Employee
    FOR SYSTEM_TIME AS  OF '2016-08-06 08:32:37.91'
```

FÜR SYSTEM_TIME ZWISCHEN UND

Das Gleiche wie oben in der Beschreibung von FOR SYSTEM_TIME FROM <start_date_time> TO <end_date_time> beschrieben, mit der Ausnahme, dass die zurückgegebene Tabelle mit Zeilen Zeilen enthält, die an der vom Endpoint <end_date_time> definierten oberen Grenze aktiv wurden.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME BETWEEN  '2015-01-01' AND '2015-12-31'
```

FÜR SYSTEM_TIME FROM ZU

Gibt eine Tabelle mit den Werten für alle Zeilenversionen zurück, die innerhalb des angegebenen

Zeitbereichs aktiv waren, unabhängig davon, ob sie vor dem Parameterwert <start_date_time> für das Argument FROM aktiv waren oder nach dem Parameterwert <end_date_time> für den Parameter nicht mehr aktiv waren Zu argumentieren. Intern wird eine Vereinigung zwischen der temporalen Tabelle und ihrer Verlaufstabelle durchgeführt, und die Ergebnisse werden gefiltert, um die Werte für alle Zeilenversionen zurückzugeben, die zu einem beliebigen Zeitpunkt im angegebenen Zeitraum aktiv waren. Zeilen, die genau an der vom FROM-Endpunkt definierten unteren Grenze aktiv wurden, werden eingeschlossen, und Datensätze, die genau an der vom TO-Endpunkt definierten oberen Grenze aktiv wurden, sind nicht enthalten.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME FROM '2015-01-01' TO '2015-12-31'
```

FÜR SYSTEM_TIME ENTHALTEN (.)

Gibt eine Tabelle mit den Werten für alle Zeilenversionen zurück, die innerhalb des angegebenen Zeitraums geöffnet und geschlossen wurden, der durch die beiden datetime-Werte für das Argument CONTAINED IN definiert wird. Zeilen, die genau an der unteren Grenze aktiv wurden oder nicht mehr genau an der oberen Grenze aktiv waren, werden eingeschlossen.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME CONTAINED IN ('2015-04-01', '2015-09-25')
```

FOR SYSTEM_TIME ALL

Gibt die Vereinigung von Zeilen zurück, die zur aktuellen und zur Verlaufstabelle gehören.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME ALL
```

Erstellen einer speicheroptimierten Systemversion der temporalen Tabelle und Bereinigen der SQL Server-Verlaufstabelle

Das Erstellen einer temporalen Tabelle mit einer Standardprotokolltabelle ist eine bequeme Option, wenn Sie die Benennung steuern möchten und dennoch das System zur Erstellung der Protokolltabelle mit der Standardkonfiguration benötigen. In dem folgenden Beispiel ist eine neue systemversionierte, speicheroptimierte temporale Tabelle mit einer neuen datenträgerbasierten Verlaufstabelle verknüpft.

```
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 )
    );
```

Bereinigen der SQL Server-Verlaufstabelle Mit der Zeit kann die Verlaufstabelle erheblich anwachsen. Da das Einfügen, Aktualisieren oder Löschen von Daten aus der Verlaufstabelle nicht zulässig ist, können Sie die Verlaufstabelle nur bereinigen, indem Sie die Versionierung des Systems deaktivieren:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = OFF); GEHEN

Löschen Sie nicht benötigte Daten aus der Verlaufstabelle:

```
    DELETE FROM dbo.EmployeeHistory
```

WO Endzeit <= '2017-01-26 14:00:29';

und aktivieren Sie die Systemversion erneut:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = [dbo]. [EmployeeHistory], DATA_CONSISTENCY_CHECK = ON));

Das Bereinigen der Protokolltabelle in Azure SQL-Datenbanken unterscheidet sich ein wenig, da Azure SQL-Datenbanken eine integrierte Unterstützung für das Bereinigen der Protokolltabelle bieten. Erstens muss die Bereinigung der temporären Verlaufsbeibehaltung auf Datenbankebene aktiviert werden:

```
ALTER DATABASE CURRENT
```

SET TEMPORAL_HISTORY_RETENTION ON GO

Legen Sie dann die Aufbewahrungszeit pro Tabelle fest:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = ON (HISTORY_RETENTION_PERIOD = 90 TAGE));

Dadurch werden alle Daten in der Verlaufstabelle gelöscht, die älter als 90 Tage sind. Die lokalen Datenbanken von SQL Server 2016 unterstützen TEMPORAL_HISTORY_RETENTION und HISTORY_RETENTION_PERIOD nicht. Eine der beiden obigen Abfragen wird in den lokalen Datenbanken von SQL Server 2016 ausgeführt.

Für TEMPORAL_HISTORY_RETENTION wird ein Fehler angezeigt:

```
Msg 102, Level 15, State 6, Line 34
```

Falsche Syntax in der Nähe von 'TEMPORAL_HISTORY_RETENTION'.

Für HISTORY_RETENTION_PERIOD wird ein Fehler auftreten:

```
Msg 102, Level 15, State 1, Line 39
```

Falsche Syntax in der Nähe von 'HISTORY_RETENTION_PERIOD'.

Zeittabellen online lesen: https://riptutorial.com/de/sql-server/topic/5296/zeittabellen

Credits

S. No	Kapitel	Contributors
1	Erste Schritte mit 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	Abfragehinweise	cteski, DARKOCEAN, Jovan MSFT, user_0
3	Abfragen der Ergebnisse nach Seite	Pat
4	Abfragen mit JSON-Daten	bakedpatato, James, Jovan MSFT
5	Abfragespeicher	bakedpatato, Jovan MSFT
6	Aggregatfunktionen	Akshay Anand, cnayak, cteski, Jeffrey L Whitledge, Joe Taras, Vexator
7	Aliasnamen in SQL Server	Pரதீ ப
8	Allgemeine Tabellenausdrücke	Arif, bbrown, cteski, DForck42, Jeffrey Van Laethem, Jovan MSFT, kafka, Keith Hall, Monty Wild, SQLMason
9	Ändern Sie den JSON-Text	Jovan MSFT
10	ANSICHT ERSTELLEN	Almir Vuk, cteski, Edathadan Chief aka Arun, Hadi, Josh B, Robert Columbia, Tot Zam
11	Ansichten	Benjamin Hodgson, Daniel Lemke, Max
12	ANSONSTEN	cteski, M.Ali, RamenChef
13	Auslösen	Oluwafemi, The_Outsider, Zohar Peled

14	bcp (Massenkopierprogramm) Dienstprogramm	MarmiK
15	Begrenzung von Sonderzeichen und reservierten Wörtern	bassrek
16	Beitreten	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
17	Benutzerdefinierte Tabellentypen	Jivan, Zohar Peled
18	Berechnete Spalten	cnayak, Kannan Kandasamy
19	Berechtigungen und Sicherheit	5arx
20	BULK Import	Jovan MSFT
21	CASE-Erklärung	Laughing Vergil, RamenChef, Vikas Vaidya
22	CLUSTERED COLUMNSTORE	Jovan MSFT
23	Common Language Runtime Integration	Jovan MSFT
24	Cursor	Kane, Phrancis
25	Dateigruppe	Behzad
26	Datenbank sichern und wiederherstellen	Jones Joseph, Jovan MSFT
27	Datenbankberechtigungen	Ben Thul
28	Datenbank-Snapshots	Akash, Daryl, Jovan MSFT, Wolfgang
29	Datenfluss	Raghu Ariga
30	Datentypen	Laughing Vergil, Matas Vaitkevicius
31	Datentypen konvertieren	Ben O, Edathadan Chief aka Arun
32	Datumsbereich generieren	James, Siyual

33	DBCC	Jovan MSFT
34	DBMAIL	Phrancis
35	Die STUFF-Funktion	Arthur D, bluefeet, Chetan Sanghani, dacohenii, Kiran Ukande, Luis Bosquez, MrE, user1690166
36	Dienstmakler	Ken S., Matej, RamenChef
37	Dynamische Datenmaskierung	Jovan MSFT
38	Dynamisches SQL	Jovan MSFT
39	Dynamisches SQL-Pivot	Jesse
40	Eine Abfrage analysieren	DForck42
41	Einfügen	Randall
42	EINFÜGEN IN	Abubakar Riaz, barcanoj, DVJex, Hari K M, intox, martinshort, Matas Vaitkevicius, Max, Michael Stum, n00b, Piotr Nawrot, Robert Columbia, Tot Zam, woony
43	Ergebnismenge begrenzen	alalp, chrisb, cteski, ErikE
44	Erweiterte Optionen	Ahmad Aghazadeh
45	Exportieren Sie Daten in TXT-Datei mithilfe von SQLCMD	sheraz mirza
46	Fensterfunktionen	andyabel, feetwet, MarmiK
47	Fremde Schlüssel	Jovan MSFT
48	FÜR JSON	James, Jovan MSFT
49	FÜR XML-PFAD	bluefeet, gotqn, Keith Hall, Wolfgang
50	Geplante Aufgabe oder Auftrag	Edathadan Chief aka Arun, Hadi
51	Gespeicherte Prozeduren	Bino Mathew Varghese, cnayak, cteski, Erik Oppedijk, Eugene Niemand, Hari K M, Jayasurya Satheesh, Matas Vaitkevicius, Nathan Skerl, Pirate X, scsimon
52	Grundlegende DDL-Vorgänge in MS SQL Server	Matt

53	GRUPPIERE NACH	Andy, Edathadan Chief aka Arun, Jenism, juergen d, Julien Vavasseur, Kiran Ukande, Matas Vaitkevicius, ShlomiR
54	Index	Ahmad Aghazadeh, Akshay Anand, cteski, Henrik Staun Poulsen, Martin Smith, Tom V
55	In-Memory-OLTP (Hekaton)	Akshay Anand, Behzad, Brandon, Jovan MSFT, Martijn Pieters
56	Isolationsstufen und Verriegelung	RamenChef, sqlandmore.com
57	JSON im SQL Server	Jovan MSFT, Mono
58	Kreuz anwenden	Hamza Rabah, Jovan MSFT, Tom V
59	Letzte eingefügte Identität	Jeffrey Van Laethem, sqluser, Tot Zam
60	Logische Funktionen	dd4711
61	Microsoft SQL Server Management Studio-Tastenkombinationen	Bino Mathew Varghese, cteski, Sibeesh Venu
62	Migration	Matas Vaitkevicius
63	Mit Krawatten Option	TheGameiswar
64	Nativ zusammengestellte Module (Hekaton)	bakedpatato, Jovan MSFT
65	NULL	Amir Pourmand, Hadi, Kannan Kandasamy, Kritner, Laughing Vergil, podiluska, Sean Branchaw, Zohar Peled
66	OPENJSON	James, Jovan MSFT
67	Parsename	Mani
68	Partitionierung	Dan Guzman, James Anderson, John Odom, Susang
69	PHANTOM lesen	Max
70	PIVOT / UNPIVOT	Athafoud, bluefeet, DhruvJoshi, kolunar
71	Primärschlüssel	Kritner
72	Privilegien oder Berechtigungen	Oluwafemi

73	Ranking-Funktionen	cteski, kolunar, New
74	Räumliche Daten	cteski, Neil Kennedy, RamenChef, Vladimir Oselsky
75	Reihen sortieren / sortieren	APH, OzrenTkalcecKrznaric
76	Ressourcen-Gouverneur	Ako, RamenChef
77	Rufen Sie Informationen zu Ihrer Instanz ab	Bino Mathew Varghese, feetwet, James Anderson, Kritner, LowlyDBA, S.Karras, scsimon
78	Rufen Sie Informationen zur Datenbank ab	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
79	Schemas	Merenix
80	Schlüsselwort löschen	Ignas, Jakub Ojmucianski, Justin Rohr, Max, scsimon
81	SCOPE_IDENTITY ()	Dheeraj Kumar
82	Seitennummerierung	cteski, Jovan MSFT, Sender
83	SELECT-Anweisung	cteski, Jovan MSFT
84	Sequenzen	Josh Morel
85	Sicherheit auf Zeilenebene	Carsten Hynne, Jovan MSFT
86	SORTIEREN NACH	APH, beercohol, cteski, Gidil, RamenChef
87	Speichern von JSON in SQL-Tabellen	Ed Harper, Jovan MSFT, RamenChef
88	Split-String-Funktion in SQL Server	Jibin Balachandran, Jovan MSFT, MasterBob, Pரத ப ், RamenChef
89	SQL Server Evolution durch verschiedene Versionen (2000 - 2016)	Dan Guzman, M.Ali
90	SQL Server Management Studio (SSMS)	dd4711
91	SQL Server unter Windows installieren	Luis Bosquez
92	SQLCMD	Eugene Niemand, Techie

93	String Aggregatfunktionen in SQL Server	Kannan Kandasamy
94	String-Funktionen	A_Arnold, anon, cteski, FoxyBOA, Hadi, hatchet, Igor Micev, Jibin Balachandran, Jovan MSFT, mtb, Phrancis, Raidri, Ricardo C, Ross Presser, takrl, Zohar Peled
95	Systemdatenbank - TempDb	Anuj Tripathi, RamenChef
96	Tabellenwertparameter	Zohar Peled
97	Termine	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
98	Transaktionsabwicklung	Metanormal
99	Transaktionsisolationsstufen	Phrancis
100	ÜBER Klausel	Athafoud, bluefeet, Brandon, DVT, gofr1, Lamak, Paul Bambury, RamenChef, Rowland Shaw, Sam
101	UNION	cnayak
102	Unterabfragen	cnayak
103	Variablen	APH, Keith Hall, Phrancis
104	Verschieben und Kopieren von Daten um Tabellen herum	Nick.McDermaid
105	Verschlüsselung	Rubenisme
106	VERSCHMELZEN	Bharat Prasad Satyal, Edathadan Chief aka Arun, Karthikeyan, Matej, scsimon, Tab Alleman
107	VERSUCHEN / FANGEN	Jovan MSFT, ravindra, Uberzen1
108	Verwalten der Azure SQL-Datenbank	Jovan MSFT
109	Verwendung der TEMP-Tabelle	APH, New
110	Volltextindizierung	Edathadan Chief aka Arun
111	While-Schleife	lord5et, Matas Vaitkevicius, podiluska, RamenChef, Wojciech Kazior