EBook Gratuito

APPRENDIMENTO
Microsoft SQL Server

Free unaffiliated eBook created from
Stack Overflow contributors.

#sql-server

Sommario

Di

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

Capitolo 1: Introduzione a Microsoft SQL Server

Osservazioni

Questo è un insieme di esempi che evidenziano l'utilizzo di base di SQL Server.

Versioni

Versione	Data di rilascio
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: le basi del linguaggio di manipolazione dei dati

D ana **M** anipolazione **L** anguage (DML in breve) include operazioni come `INSERT` , `UPDATE` e `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 questo script stiamo **creando una tabella** per dimostrare alcune query di base.

I seguenti esempi mostrano come **interrogare le tabelle:**

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

selezionerà i primi 10 record della tabella `Customer` , ordinati dalla colonna `CompanyName` dal database `Northwind` (che è uno dei database di esempio di Microsoft, può essere scaricato da **qui**):

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

Notare che `Use Northwind;` cambia il database predefinito per tutte le query successive. È ancora possibile fare riferimento al database utilizzando la sintassi completa in formato [Database]. [Schema]. [Table]:

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

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

Questo è utile se stai interrogando dati da diversi database. Si noti che `dbo` , specificato "in between" è chiamato schema e deve essere specificato mentre si utilizza la sintassi completa. Puoi considerarlo come una cartella all'interno del tuo database. `dbo` è lo schema predefinito. Lo schema predefinito può essere omesso. Tutti gli altri schemi definiti dall'utente devono essere specificati.

Se la tabella del database contiene colonne denominate come parole riservate, ad es. `Date` , è necessario racchiudere il nome della colonna tra parentesi, in questo modo:

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

Lo stesso vale se il nome della colonna contiene spazi nel suo nome (che non è raccomandato). Una sintassi alternativa è usare le virgolette anziché le parentesi quadre, ad es .:

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

è equivalente ma non così comunemente usato. Notare la differenza tra virgolette e virgolette singole: le virgolette singole sono usate per le stringhe, es

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

è una sintassi valida. Si noti che T-SQL ha un prefisso `N` per i tipi di dati NChar e NVarchar, ad es

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

restituisce tutte le società che hanno un nome di società che inizia con `AL` (`%` è una wild card, usala come se dovessi usare l'asterisco in una riga di comando DOS, ad esempio `DIR AL*`). Per `LIKE` , ci sono un paio di caratteri jolly disponibili, guarda **qui** per scoprire maggiori dettagli.

Si unisce

I join sono utili se si desidera eseguire query su campi che non esistono in una singola tabella, ma in più tabelle. Per esempio: Si vuole interrogare tutte le colonne della `Region` tabella nel `Northwind` database. Ma si nota che è necessario anche `RegionDescription` , che è memorizzato in una tabella diversa, `Region` . Tuttavia, esiste una chiave comune, `RgionID` che è possibile utilizzare per combinare queste informazioni in una singola query come segue (`Top 5` restituisce solo le prime 5 righe, omettendole per ottenere tutte le righe):

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

mostrerà tutte le colonne da `Territories` più la colonna `RegionDescription` da `Region` . Il risultato è ordinato da `TerritoryDescription` .

Alias di tabella

Quando la tua query richiede un riferimento a due o più tabelle, potresti trovare utile utilizzare un alias di tabella. Gli alias di tabella sono riferimenti abbreviati a tabelle che possono essere utilizzati al posto di un nome di tabella completo e possono ridurre la digitazione e la modifica. La sintassi per l'utilizzo di un alias è:

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

Dove `as` è una parola chiave opzionale. Ad esempio, la query precedente può essere riscritta come:

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

Gli alias devono essere univoci per tutte le tabelle di una query, anche se si utilizza la stessa tabella due volte. Ad esempio, se la tabella Employee include un campo SupervisorId, è possibile utilizzare questa query per restituire un dipendente e il nome del proprio supervisore:

```
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
```

sindacati

Come abbiamo visto prima, un join aggiunge colonne da diverse origini di tabelle. Ma cosa succede se si desidera combinare righe provenienti da fonti diverse? In questo caso puoi usare UNION. Supponiamo che tu stia pianificando una festa e desideri invitare non solo i dipendenti, ma anche i clienti. Quindi puoi eseguire questa query per farlo:

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

```
SELECT ContactName, Address, City FROM Customers
```

Restituirà nomi, indirizzi e città da dipendenti e clienti in un'unica tabella. Tieni presente che le righe duplicate (se ce ne dovrebbero essere) vengono automaticamente eliminate (se non lo desideri, utilizza invece `UNION ALL`). Il numero di colonna, i nomi di colonna, l'ordine e il tipo di dati devono corrispondere a tutte le istruzioni select che fanno parte dell'unione: ecco perché il primo SELECT combina `FirstName` e `LastName` da Employee in `ContactName` .

Variabili di tabella

Può essere utile, se hai bisogno di gestire i dati temporanei (specialmente in una procedura memorizzata), per utilizzare le variabili di tabella: la differenza tra una tabella "reale" e una variabile di tabella è che esiste solo in memoria per l'elaborazione temporanea.

Esempio:

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

crea una tabella in memoria. In questo caso il prefisso `@` è obbligatorio perché è una variabile. Puoi eseguire tutte le operazioni DML sopra menzionate per inserire, eliminare e selezionare righe, ad es

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

Ma normalmente, lo si popolerebbe sulla base di un tavolo reale come

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

che legge i valori filtrati dalla reale tabella `dbo.Region` e lo inserisce nella tabella di memoria `@Region` - dove può essere utilizzato per ulteriori elaborazioni. Ad esempio, potresti usarlo in un join come

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

che in questo caso restituirebbe tutti i territori del `Northern` e del `Southern` . Informazioni più dettagliate possono essere trovate qui . Le tabelle temporanee sono discusse qui , se sei interessato a saperne di più su questo argomento.

NOTA: Microsoft consiglia solo l'uso di variabili di tabella se il numero di righe di dati nella variabile di tabella è inferiore a 100. Se si utilizzano quantità maggiori di dati, utilizzare invece una **tabella temporanea** o una **tabella temporanea** .

STAMPARE

Mostra un messaggio alla console di output. Utilizzando SQL Server Management Studio, questo verrà visualizzato nella scheda dei messaggi, anziché nella scheda dei risultati:

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

SELEZIONA tutte le righe e le colonne da una tabella

Sintassi:

```
SELECT *
FROM table_name
```

L'utilizzo dell'operatore asterisco `*` funge da scorciatoia per selezionare tutte le colonne nella tabella. Verranno inoltre selezionate tutte le righe poiché questa istruzione `SELECT` non ha una clausola `WHERE` , per specificare eventuali criteri di filtro.

Ciò funzionerebbe allo stesso modo se hai aggiunto un alias alla tabella, ad esempio `e` in questo caso:

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

O se vuoi selezionare tutto da una tabella specifica puoi usare l'alias + ". *":

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

È possibile accedere agli oggetti del database anche utilizzando nomi completi:

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

Ciò non è necessariamente raccomandato, in quanto la modifica dei nomi di server e / o database causerebbe l'interruzione delle query utilizzando nomi completi per i nomi di oggetti non validi.

Si noti che i campi prima di `table_name` possono essere omessi in molti casi se le query vengono eseguite su un singolo server, database e schema, rispettivamente. Tuttavia, è comune che un database abbia più schemi e, in questi casi, il nome dello schema non dovrebbe essere omesso quando possibile.

Avvertenza: l' utilizzo di `SELECT *` nel codice di produzione o nelle stored procedure può causare problemi in seguito (quando nuove colonne vengono aggiunte alla tabella o se le colonne vengono ridisposte nella tabella), specialmente se il codice fa presupposti semplici sull'ordine delle colonne, o numero di colonne restituite. Quindi è più sicuro specificare sempre i nomi delle colonne nelle istruzioni SELECT per il codice di produzione.

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

Seleziona le righe che corrispondono a una condizione

Generalmente, la sintassi è:

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

Per esempio:

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

Le condizioni possono essere complesse:

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

UPDATE Specifica riga

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

Il codice sopra riportato aggiorna il valore del campo "HelloWorld" con "CIAO MONDO !!!" per il record in cui "Id = 5" nella tabella HelloWorlds.

Nota: in una dichiarazione di aggiornamento, si consiglia di utilizzare una clausola "where" per evitare di aggiornare l'intera tabella a meno che e fino a quando il requisito non sia diverso.

AGGIORNA Tutte le righe

Una semplice forma di aggiornamento sta incrementando tutti i valori in un dato campo della tabella. Per fare ciò, dobbiamo definire il campo e il valore dell'incremento

Di seguito è riportato un esempio che incrementa il campo `Score` di 1 (in tutte le righe):

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

Questo può essere pericoloso poiché puoi danneggiare i tuoi dati se accidentalmente fai un UPDATE per una **riga specifica** con un UPDATE per **Tutte le righe** nella tabella.

Commenti nel codice

Transact-SQL supporta due forme di scrittura di commenti. I commenti vengono ignorati dal motore del database e sono pensati per essere letti.

I commenti sono preceduti da `--` e vengono ignorati fino a quando non viene rilevata una nuova riga:

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

I commenti barra iniziale iniziano con `/*` e terminano con `*/` . Tutto il testo tra quei delimitatori è considerato un blocco di commenti.

```
/* 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';
```

I commenti barra stella hanno il vantaggio di mantenere il commento utilizzabile se l'istruzione SQL perde caratteri di nuova riga. Questo può accadere quando SQL viene catturato durante la risoluzione dei problemi.

I commenti delle stelle di tipo slash possono essere nidificati e l'inizio `/*` all'interno di un commento a stella barra deve essere terminato con un `*/` per essere valido. Il seguente codice provocherà un errore

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

La stella di taglio anche se all'interno della citazione è considerata come l'inizio di un commento. Quindi deve essere terminato con un altro taglio di chiusura. Il modo corretto sarebbe

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

Recupera informazioni sul server di base

```
SELECT @@VERSION
```

Restituisce la versione di MS SQL Server in esecuzione nell'istanza.

```
SELECT @@SERVERNAME
```

Restituisce il nome dell'istanza MS SQL Server.

```
SELECT @@SERVICENAME
```

Restituisce il nome del servizio Windows MS SQL Server è in esecuzione come.

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

Restituisce il nome fisico della macchina su cui è in esecuzione SQL Server. Utile per identificare il nodo in un cluster di failover.

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

In un cluster di failover restituisce tutti i nodi su cui è possibile eseguire SQL Server. Non restituisce nulla se non un cluster.

Utilizzo delle transazioni per modificare i dati in modo sicuro

Ogni volta che si modificano i dati, in un comando DML (Data Manipulation Language), è possibile includere le modifiche in una transazione. DML include `UPDATE` , `TRUNCATE` , `INSERT` e `DELETE` . Uno dei modi in cui puoi essere sicuro che stai cambiando i dati giusti sarebbe utilizzare una transazione.

Le modifiche DML prenderanno un blocco sulle righe interessate. Quando inizi una transazione, devi terminare la transazione o tutti gli oggetti modificati nel DML rimarranno bloccati da chiunque abbia iniziato la transazione. Puoi terminare la transazione con `ROLLBACK` o `COMMIT` . `ROLLBACK` restituisce tutto nella transazione allo stato originale. `COMMIT` inserisce i dati in uno stato finale in cui non è possibile annullare le modifiche senza un'altra istruzione DML.

Esempio:

```
--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
```

Gli appunti:

- Questo è un **esempio semplificato** che non include la gestione degli errori. Ma qualsiasi operazione di database può fallire e quindi generare un'eccezione. **Ecco un esempio di** come potrebbe essere una tale gestione degli errori richiesta. Non dovresti **mai** usare le transazioni **senza un gestore di errori** , altrimenti potresti lasciare la transazione in uno stato sconosciuto.
- A seconda del **livello di isolamento** , le transazioni stanno mettendo i blocchi sui dati interrogati o modificati. È necessario assicurarsi che le transazioni non siano in esecuzione per un lungo periodo, poiché bloccano i record in un database e possono portare a **deadlock** con altre transazioni parallele in esecuzione. Mantieni le operazioni incapsulate nelle transazioni il più breve possibile e minimizza l'impatto con la quantità di dati che stai bloccando.

CANCELLA Tutte le righe

```
DELETE
FROM Helloworlds
```

Questo cancellerà tutti i dati dalla tabella. La tabella non conterrà righe dopo aver eseguito questo codice. A differenza di `DROP TABLE` , questo conserva la tabella stessa e la sua struttura e puoi continuare a inserire nuove righe in quella tabella.

Un altro modo per eliminare tutte le righe nella tabella è troncarlo, come segue:

```
TRUNCATE TABLE HelloWords
```

Le differenze con l'operazione DELETE sono diverse:

1. L'operazione di troncamento non viene archiviata nel file di registro delle transazioni
2. Se esiste il campo `IDENTITY` , questo sarà resettato
3. TRUNCATE può essere applicato su tutta la tabella e no su parte di esso (invece con il comando `DELETE` è possibile associare una clausola `WHERE`)

Restrizioni di TRUNCATE

1. Impossibile TRONCATARE una tabella se è presente un riferimento `FOREIGN KEY`
2. Se il tavolo è partecipato a una `INDEXED VIEW`
3. Se la tabella viene pubblicata utilizzando `TRANSACTIONAL REPLICATION` o `MERGE REPLICATION`

4. Non verrà generato alcun TRIGGER definito nella tabella

[sic]

TABELLA DEI TRONCATI

```
TRUNCATE TABLE Helloworlds
```

Questo codice cancellerà tutti i dati dalla tabella Helloworlds. La tabella troncata è quasi simile a `Delete from Table` codice `Delete from Table` . La differenza è che non è possibile utilizzare le clausole where con Truncate. La tabella troncata è considerata migliore dell'eliminazione perché utilizza meno spazi del log delle transazioni.

Si noti che se esiste una colonna Identity, viene reimpostata sul valore di inizializzazione iniziale (ad esempio, l'ID con incremento automatico verrà riavviato da 1). Ciò può causare incoerenza se le colonne Identity vengono utilizzate come chiavi esterne in un'altra tabella.

Crea una nuova tabella e inserisci i record dalla vecchia tabella

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

Crea una nuova tabella con la struttura della vecchia tabella e inserisce tutte le righe nella nuova tabella.

Alcune restrizioni

1. Non è possibile specificare una variabile di tabella o un parametro con valori di tabella come nuova tabella.
2. Non è possibile utilizzare SELECT ... INTO per creare una tabella partizionata, anche quando la tabella di origine è partizionata. SELECT ... INTO non usa lo schema di partizione della tabella di origine; al contrario, la nuova tabella viene creata nel filegroup predefinito. Per inserire righe in una tabella partizionata, è necessario prima creare la tabella partizionata e quindi utilizzare l'istruzione INSERT INTO ... SELECT FROM.
3. Gli indici, i vincoli e i trigger definiti nella tabella di origine non vengono trasferiti alla nuova tabella, né possono essere specificati nell'istruzione SELECT ... INTO. Se questi oggetti sono richiesti, è possibile crearli dopo aver eseguito l'istruzione SELECT ... INTO.
4. La specifica di una clausola ORDER BY non garantisce che le righe vengano inserite nell'ordine specificato. Quando una colonna sparsa è inclusa nell'elenco di selezione, la proprietà della colonna sparsa non viene trasferita alla colonna nella nuova tabella. Se questa proprietà è richiesta nella nuova tabella, modificare la definizione della colonna dopo aver eseguito l'istruzione SELECT ... INTO per includere questa proprietà.
5. Quando una colonna calcolata è inclusa nell'elenco di selezione, la colonna corrispondente nella nuova tabella non è una colonna calcolata. I valori nella nuova colonna sono i valori che sono stati calcolati al momento in cui SELECT ...

INTO è stato eseguito.

[*sic*]

Ottenere il conteggio delle righe della tabella

L'esempio seguente può essere utilizzato per trovare il numero totale di righe per una tabella specifica in un database se `table_name` viene sostituito dalla tabella che si desidera interrogare:

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

È anche possibile ottenere il conteggio delle righe per tutte le tabelle tornando alla partizione della tabella in base alle tabelle HEAP (index_id = 0) o cluster cluster index (index_id = 1) utilizzando il seguente script:

```
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;
```

Ciò è possibile in quanto ogni tabella è essenzialmente una singola tabella di partizione, a meno che non vengano aggiunte ulteriori partizioni. Questo script ha anche il vantaggio di non interferire con le operazioni di lettura / scrittura sulle righe delle tabelle '.

Leggi Introduzione a Microsoft SQL Server online: https://riptutorial.com/it/sql-server/topic/236/introduzione-a-microsoft-sql-server

Capitolo 2: Aderire

introduzione

In Structured Query Language (SQL), un JOIN è un metodo per collegare due tabelle di dati in una singola query, consentendo al database di restituire una serie che contiene i dati di entrambe le tabelle contemporaneamente o utilizzando i dati di una tabella per utilizzarli come Filtro sul secondo tavolo. Esistono diversi tipi di JOIN definiti nello standard ANSI SQL.

Examples

Join interno

`Inner join` restituisce solo quei record / righe che corrispondono / esistono in entrambe le tabelle in base a una o più condizioni (specificate usando la parola chiave `ON`). È il tipo più comune di join. La sintassi generale per l' `inner join` è:

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

Può anche essere semplificato semplicemente come `JOIN` :

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

Esempio

```
/* 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
```

Utilizzo di inner join con outer join sinistro (sostituto di non esiste)

Questa query restituirà i dati dalla tabella 1 dove i campi che corrispondono a table2 con una chiave e i dati non nella Tabella 1 quando si confrontano con Table2 con una condizione e una chiave

```
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` è un join cartesiano, che significa un prodotto cartesiano di entrambi i tavoli. Questo join non ha bisogno di alcuna condizione per unire due tabelle. Ogni riga nella tabella di sinistra si unirà a ciascuna riga della tabella di destra. Sintassi per un cross join:

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

Esempio:

```
/* 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;
```

risultati:

```
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
```

Nota che ci sono altri modi in cui un CROSS JOIN può essere applicato. Questo è un join *"vecchio stile"* (deprecato dal ANSI SQL-92) senza condizione, che risulta in un cross / Cartesian join:

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

Questa sintassi funziona anche a causa di una condizione di join "sempre true", ma non è raccomandata e dovrebbe essere evitata, a favore della sintassi `CROSS JOIN` esplicita, per motivi di leggibilità.

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

Outer Join

Left Outer Join

`LEFT JOIN` restituisce tutte le righe dalla tabella sinistra, abbinate alle righe della tabella di destra in cui sono soddisfatte le condizioni della clausola `ON` . Le righe in cui non viene soddisfatta la clausola `ON` hanno `NULL` in tutte le colonne della tabella destra. La sintassi di un `LEFT JOIN` è:

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

Giusto outer join

`RIGHT JOIN` restituisce tutte le righe dalla tabella di destra, abbinate alle righe della tabella sinistra in cui sono soddisfatte le condizioni della clausola `ON` . Le righe in cui non viene soddisfatta la clausola `ON` hanno `NULL` in tutte le colonne della tabella sinistra. La sintassi di `RIGHT JOIN` è:

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

Full Outer Join

`FULL JOIN` combina `LEFT JOIN` e `RIGHT JOIN` . Tutte le righe vengono restituite da entrambe le tabelle, indipendentemente dal fatto che siano soddisfatte le condizioni nella clausola `ON` . Le righe che non soddisfano la clausola `ON` vengono restituite con `NULL` in tutte le colonne della tabella opposta (ovvero, per una riga nella tabella sinistra, tutte le colonne nella tabella destra conterranno `NULL` e viceversa). La sintassi di un `FULL JOIN` è:

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

Esempi

```
/* 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. */
```

SINISTRA ESTERNO

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

Risultati per `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
```

GIUSTO ESTERNO

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

Risultati per `RIGHT JOIN`

```
AnimalId    Animal               AnimalSoundId AnimalId    Sound
----------- -------------------- ------------- ----------- --------------------
1           Dog                  1             1           Barks
2           Cat                  2             2           Meows
3           Elephant             3             3           Trumpet
NULL        NULL                 4             5           Roars
```

FULL OUTER JOIN

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

Risultati per `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
```

Utilizzo Partecipa a un aggiornamento

I join possono anche essere utilizzati in un'istruzione `UPDATE` :

```
CREATE TABLE Users (
    UserId int NOT NULL,
    AccountId int NOT NULL,
    RealName nvarchar(200) NOT NULL
)

CREATE TABLE Preferences (
    UserId int NOT NULL,
    SomeSetting bit NOT NULL
)
```

Aggiorna la colonna `SomeSetting` del filtro delle tabelle delle `Preferences` base a un predicato sulla tabella `Users` come segue:

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

`p` è un alias per le `Preferences` definite nella clausola `FROM` dell'istruzione. Verranno aggiornate solo le righe con un `AccountId` corrispondente dalla tabella `Users` .

Aggiorna con dichiarazioni di outer outer a sinistra

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

Aggiorna tabelle con join interno e funzione di aggregazione

```
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
```

Partecipa a una sottoquery

Unirsi a una sottoquery viene spesso utilizzato quando si desidera ottenere dati aggregati (come Count, Avg, Max o Min) da una tabella child / details e visualizzarli insieme ai record dalla tabella parent / header. Ad esempio, potresti voler recuperare la riga superiore / primo figlio in base a Data o Id o forse vuoi un Conteggio di tutte le Righe figlio o una Media.

Questo esempio utilizza alias che facilita la lettura delle query quando sono coinvolte più tabelle. In questo caso, recuperiamo tutte le righe dalla tabella padre Ordini d'acquisto e recuperiamo solo l'ultima (o più recente) riga secondaria dalla tabella figlio PurchaseOrderLineItems. Questo esempio presuppone che la tabella figlio utilizza ID numerici incrementali.

```
SELECT po.Id, po.PODate, po.VendorName, po.Status, item.ItemNo,
  item.Description, item.Cost, item.Price
FROM PurchaseOrders po
LEFT JOIN
     (
       SELECT l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price, Max(l.id) as Id
       FROM PurchaseOrderLineItems l
       GROUP BY l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost, l.Price
     ) AS item ON item.PurchaseOrderId = po.Id
```

Self Join

Una tabella può essere unita a se stessa in quello che è noto come self join, combinando i record nella tabella con altri record nella stessa tabella. I join automatici vengono in genere utilizzati nelle query in cui è definita una gerarchia nelle colonne della tabella.

Considera i dati di esempio in una tabella denominata `Employees` :

ID	Nome	Boss_ID
1	peso	3
2	Jim	1
3	Sam	2

Ogni dipendente `Boss_ID` mappato di un altro dipendente `ID` . Per recuperare un elenco di dipendenti con il nome del rispettivo capo, la tabella può essere unita su se stessa utilizzando questa mappatura. Si noti che unirsi a una tabella in questo modo richiede l'uso di un alias (`Bosses` in questo caso) sul secondo riferimento alla tabella per distinguersi dalla tabella originale.

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

L'esecuzione di questa query produrrà i seguenti risultati:

Nome	Capo
peso	Sam
Jim	peso
Sam	Jim

Elimina usando Join

I join possono anche essere utilizzati in una istruzione `DELETE` . Dato uno schema come segue:

```
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
```

```
)
```

Possiamo eliminare le righe dalla tabella delle `Preferences` , filtrandole secondo un predicato sulla tabella `Users` come segue:

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

Qui `p` è un alias per le `Preferences` definite nella clausola `FROM AccountId` e cancelliamo solo le righe che hanno un `AccountId` corrispondente dalla tabella `Users` .

Girando accidentalmente un'unione esterna in un'unione interna

Join esterni restituiscono tutte le righe da una o entrambe le tabelle, più le righe corrispondenti.

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

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

A sinistra si uniscono ai tavoli:

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

Ritorna:

```
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
```

Se si desidera restituire tutte le persone, con qualsiasi punteggio matematico applicabile, un errore comune è scrivere:

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

Ciò eliminerebbe Eva dai tuoi risultati, oltre a rimuovere il punteggio scientifico di Bob, poiché

`Subject` è `NULL` per lei.

La sintassi corretta per rimuovere i record non matematici mantenendo tutti gli individui nella tabella `People` potrebbe essere:

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

Leggi Aderire online: https://riptutorial.com/it/sql-server/topic/1008/aderire

Capitolo 3: Analizzando una query

Examples

Scan vs Seek

Quando si visualizza un piano di esecuzione, è possibile che SQL Server abbia deciso di eseguire una ricerca o una scansione.

Una ricerca si verifica quando SQL Server sa dove deve andare e acquisisce solo elementi specifici. Ciò si verifica in genere quando i buoni filtri vengono inseriti in una query, ad esempio `where name = 'Foo'`.

Una scansione è quando SQL Server non sa esattamente dove sono tutti i dati di cui ha bisogno, o ha deciso che la scansione sarebbe stata più efficiente di una ricerca se è stato selezionato un numero sufficiente di dati.

Le ricerche sono in genere più veloci poiché stanno acquisendo solo una sottosezione dei dati, mentre le scansioni stanno selezionando la maggior parte dei dati.

Leggi Analizzando una query online: https://riptutorial.com/it/sql-server/topic/7713/analizzando-una-query

Capitolo 4: Attività pianificata o lavoro

introduzione

SQL Server Agent utilizza SQL Server per archiviare le informazioni sul lavoro. I lavori contengono uno o più passaggi di lavoro. Ogni fase contiene il proprio compito, ovvero: eseguire il backup di un database. SQL Server Agent può eseguire un lavoro in base a una pianificazione, in risposta a un evento specifico oa richiesta.

Examples

Crea un lavoro programmato

Crea un lavoro

- Per aggiungere un lavoro prima dobbiamo usare una stored procedure chiamata sp_add_job

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

- Quindi dobbiamo aggiungere una fase del processo utilizzando una stored procedure denominata sp_add_jobStep

```
EXEC sp_add_jobstep
@job_name = N'Weekly Job',  -- Job name to add a step
@step_name = N'Set database to read only',  -- step name
@subsystem = N'TSQL',  -- Step type
@command = N'ALTER DATABASE SALES SET READ_ONLY',   -- Command
@retry_attempts = 5,  --Number of attempts
@retry_interval = 5 ; -- in minutes
```

- Indirizzare il lavoro a un server

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

Crea una pianificazione usando SQL

Per creare un programma dobbiamo usare una stored procedure di sistema chiamata sp_add_schedule

```
USE msdb
GO
```

```
EXEC sp_add_schedule
    @schedule_name = N'NightlyJobs' ,  -- specify the schedule name
    @freq_type = 4,   -- A value indicating when a job is to be executed (4) means Daily
    @freq_interval = 1,  -- The days that a job is executed and depends on the value of
`freq_type`.
    @active_start_time = 010000 ;   -- The time on which execution of a job can begin
GO
```

Ci sono più parametri che possono essere usati con `sp_add_schedule` cui puoi leggere di più nel link fornito sopra.

Allegare pianificazione a un lavoro

Per allegare una pianificazione a un lavoro dell'agent SQL, è necessario utilizzare una stored procedure denominata sp_attach_schedule

```
-- attaches the schedule to the job BackupDatabase
EXEC sp_attach_schedule
   @job_name = N'BackupDatabase',  -- The job name to attach with
   @schedule_name = N'NightlyJobs' ;  -- The schedule name
GO
```

Leggi Attività pianificata o lavoro online: https://riptutorial.com/it/sql-server/topic/5329/attivita-pianificata-o-lavoro

Capitolo 5: Autorizzazioni del database

Osservazioni

Sintassi di base:

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

- {GRANT | REVOCA | DENY} - Quello che stai cercando di realizzare
  - Grant: "Dare questa autorizzazione al preside dichiarato"
  - Revoca: "Togliti questo permesso dal preside dichiarato"
  - Rifiuta: "Assicurati che l'entità dichiarata non abbia mai questa autorizzazione (es." `DENY SELECT` "significa che indipendentemente da qualsiasi altra autorizzazione, `SELECT` fallirà per questo principale)
- PERMISSION_NAME: l'operazione che stai tentando di modificare. Questo dipenderà dalla sicurezza. Ad esempio, non ha senso `GRANT SELECT` su una stored procedure.
- SICURO - Il nome della cosa su cui stai cercando di influenzare le autorizzazioni. Questo è *opzionale* . Dire `GRANT SELECT TO [aUser];` è perfettamente accettabile; significa "per qualsiasi cosa sicura per cui l'autorizzazione `SELECT` abbia senso, `GRANT` that that permission".
- PRINCIPALE - Per chi stai provando ad influenzare le autorizzazioni. Ad un livello di database, questo può essere un ruolo (applicazione o database) o utente (mappato a un accesso o meno) per esempio.

Examples

Modifica delle autorizzazioni

```
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];
```

CREARE UN UTENTE

```
--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];
```

CREA IL RUOLO

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

Modifica dell'appartenenza al ruolo

```
-- 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];
```

Nota: i membri del ruolo possono essere qualsiasi entità a livello di database. Cioè, puoi aggiungere un ruolo come membro in un altro ruolo. Inoltre, l'aggiunta / eliminazione dei membri del ruolo è idempotente. Cioè, il tentativo di aggiungere / eliminare comporterà la loro presenza / assenza (rispettivamente) nel ruolo indipendentemente dallo stato corrente della loro appartenenza al ruolo.

Leggi Autorizzazioni del database online: https://riptutorial.com/it/sql-server/topic/6788/autorizzazioni-del-database

Capitolo 6: Autorizzazioni e sicurezza

Examples

Assegna permessi oggetto a un utente

In produzione è buona norma proteggere i dati e consentire solo le operazioni su di esso da intraprendere tramite stored procedure. Ciò significa che la tua applicazione non può eseguire direttamente operazioni CRUD sui tuoi dati e potenzialmente causare problemi. Assegnare le autorizzazioni è un compito dispendioso in termini di tempo, maneggevole e generalmente oneroso. Per questo motivo è spesso più facile sfruttare parte della (considerevole) potenza contenuta nello schema INFORMATION_SCHEMA che è contenuto in ogni database SQL Server.

Invece, assegnando individualmente le autorizzazioni a un utente a tutto pasto, basta eseguire lo script di seguito, copiare l'output e quindi eseguirlo in una finestra di Query.

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

Leggi Autorizzazioni e sicurezza online: https://riptutorial.com/it/sql-server/topic/7929/autorizzazioni-e-sicurezza

Capitolo 7: Backup e ripristino del database

Sintassi

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

Parametri

Parametro	Dettagli
Banca dati	Il nome del database per il backup o il ripristino
backup_device	Il dispositivo per il backup o il ripristino del database, come {DISK o TAPE}. Può essere separato da virgole (,)
with_options	Varie opzioni che possono essere utilizzate durante l'esecuzione dell'operazione. Come la formattazione del disco su cui deve essere posizionato il backup o il ripristino del database con l'opzione di sostituzione.

Examples

Backup di base su disco senza opzioni

Il seguente comando esegue il backup del database *"Utenti"* nel file *"D: \ DB_Backup"* . È meglio non dare un'estensione.

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

Ripristino di base dal disco senza opzioni

Il seguente comando ripristina il database *"Utenti"* dal file *"D: \ DB_Backup"* .

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

RESTORE Database con REPLACE

Quando si tenta di ripristinare il database da un altro server, si potrebbe ottenere il seguente errore:

> Errore 3154: il set di backup contiene un backup di un database diverso dal database esistente.

In tal caso, utilizzare l'opzione WITH REPLACE per sostituire il database con il database dal

backup:

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

Anche in questo caso potresti ottenere gli errori dicendo che non è possibile localizzare i file su un percorso:

> Il messaggio 3156, livello 16, stato 3, file di riga 1 "WWI_Primary" non può essere ripristinato in "D: \ Data \ WideWorldImportersDW.mdf". Utilizzare WITH MOVE per identificare un percorso valido per il file.

Questo errore si verifica probabilmente perché i file non sono stati posizionati nello stesso percorso della cartella esistente sul nuovo server. In tal caso, è necessario spostare i singoli file di database in una nuova posizione:

```
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'
```

Con questa dichiarazione puoi sostituire il database con tutti i file di database spostati nella nuova posizione.

Leggi Backup e ripristino del database online: https://riptutorial.com/it/sql-server/topic/5826/backup-e-ripristino-del-database

Capitolo 8: Broker di servizi

Examples

1. Nozioni di base

Il broker di servizi è una tecnologia basata sulla comunicazione asincrona tra due (o più) entità. Il broker di servizi è composto da: tipi di messaggi, contratti, code, servizi, percorsi e almeno endpoint di istanza

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

2. Abilitare il broker di servizi sul database

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

3. Creare la costruzione del broker di servizi di base sul database (comunicazione su database singolo)

```
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
```

Non abbiamo bisogno di una rotta per una comunicazione di database.

4. Come inviare comunicazioni di base tramite il broker di servizi

Per questa dimostrazione useremo la costruzione del broker di servizi creata in un'altra parte di questa documentazione. La parte menzionata è denominata **3. Creare la costruzione del broker di servizi di base sul database (comunicazione di un singolo database)** .

```
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;
```

Dopo questa conversazione sarà il tuo messaggio in TargetQueue

5. Come ricevere automaticamente la conversazione da TargetQueue

Per questa dimostrazione useremo la costruzione del broker di servizi creata in un'altra parte di questa documentazione. La parte citata è denominata **3. Creazione di base del broker di servizi di base sul database (comunicazione di un singolo database)** .

Per prima cosa dobbiamo creare una procedura in grado di leggere ed elaborare i dati dalla coda

```
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
```

Secondo passaggio: consentire a TargetQueue di eseguire automaticamente la procedura:

```
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  )
```

Leggi Broker di servizi online: https://riptutorial.com/it/sql-server/topic/7651/broker-di-servizi

Capitolo 9: Chiavi esterne

Examples

Relazione / vincolo di chiave esterna

Le chiavi esterne consentono di definire la relazione tra due tabelle. Una tabella (principale) deve avere una chiave primaria che identifichi in modo univoco le righe nella tabella. Un'altra tabella (secondaria) può avere il valore della chiave primaria dal genitore in una delle colonne. Il vincolo RIFERIMENTI CHIAVE ESTERA assicura che i valori nella tabella figlio debbano esistere come valore della chiave primaria nella tabella padre.

In questo esempio abbiamo la tabella Società padre con la chiave primaria CompanyId e la tabella Employee figlio con ID della società in cui lavora questo dipendente.

```
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)
)
```

i riferimenti a chiavi esterne assicurano che i valori inseriti nella colonna Employee.CompanyId siano presenti anche nella colonna Company.CompanyId. Inoltre, nessuno può eliminare la società nella tabella della società se è presente almeno un dipendente con una società corrispondente in una tabella figlio.

La relazione FOREIGN KEY assicura che le righe in due tabelle non possano essere "scollegate".

Mantenimento della relazione tra righe padre / figlio

Supponiamo di avere una riga nella tabella Company con companyId 1. Possiamo inserire una riga nella tabella dei dipendenti che ha companyId 1:

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

Tuttavia, non possiamo inserire dipendenti con ID azienda non esistente:

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

Messaggio 547, livello 16, stato 0, riga 12 L'istruzione INSERT era in conflitto con il vincolo FOREIGN KEY "FK__Employee__Compan__1EE485AA". Il conflitto si è verificato nel database "MyDb", nella tabella "dbo.Company", nella colonna "CompanyId". La dichiarazione è stata chiusa.

Inoltre, non è possibile eliminare la riga padre nella tabella della società finché nella tabella dei dipendenti è presente almeno una riga secondaria che la fa riferimento.

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

Messaggio 547, livello 16, stato 0, riga 14 L'istruzione DELETE è in conflitto con il vincolo REFERENCE "FK__Employee__Compan__1EE485AA". Il conflitto si è verificato nel database "MyDb", nella tabella "dbo.Employee", nella colonna "CompanyId". La dichiarazione è stata chiusa.

La relazione tra le chiavi esterne assicura che le righe dell'azienda e dei dipendenti non siano "scollegate".

Aggiunta di una relazione di chiave esterna sulla tabella esistente

Il vincolo **FOREIGN KEY** può essere aggiunto su tabelle esistenti che non sono ancora in relazione. Immagina di avere una tabella Company e Employee in cui la colonna CompanyId della tabella Employee non ha una relazione di chiave esterna. L'istruzione ALTER TABLE consente di aggiungere **un** vincolo di **chiave esterna** su una colonna esistente che fa riferimento ad altre tabelle e chiavi primarie in tale tabella:

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

Aggiungi chiave esterna sul tavolo esistente

Le colonne **FOREIGN KEY** con vincolo possono essere aggiunte su tabelle esistenti che non sono ancora in relazione. Immaginiamo di avere tabelle Company e Employee in cui la tabella Employee non ha la colonna CompanyId. L'istruzione ALTER TABLE consente di aggiungere una nuova colonna con **il** vincolo di **chiave esterna** che fa riferimento ad altre tabelle e chiavi primarie in tale tabella:

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

Ottenere informazioni sui vincoli di chiave esterna

La vista di sistema sys.foreignkeys restituisce informazioni su tutte le relazioni con le chiavi esterne nel database:

```
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
```

Leggi Chiavi esterne online: https://riptutorial.com/it/sql-server/topic/5355/chiavi-esterne

Capitolo 10: Chiavi primarie

Osservazioni

Le chiavi primarie sono utilizzate per identificare univocamente un record in una tabella. Una tabella può avere una sola chiave primaria (anche se la chiave primaria può essere costituita da più colonne) e una chiave primaria è necessaria per determinati tipi di replica.

Le chiavi primarie sono spesso utilizzate come (ma non deve essere) l' indice cluster su una tabella.

Examples

Crea una tabella con colonna Identity come chiave primaria

```
-- 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
)
```

Crea una tabella con la chiave primaria GUID

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

Crea una tabella con la chiave naturale

```
-- 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
)
```

Crea tabella w / chiave composita

```
-- 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)
)
```

Aggiungi la chiave primaria alla tabella esistente

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

Nota, se la colonna della chiave primaria (in questo caso `ssn`) ha più di una riga con la stessa chiave candidata, l'istruzione precedente avrà esito negativo, poiché i valori delle chiavi primarie devono essere univoci.

Elimina la chiave primaria

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

Leggi Chiavi primarie online: https://riptutorial.com/it/sql-server/topic/4543/chiavi-primarie

Capitolo 11: Ciclo WHILE

Osservazioni

L'utilizzo di un ciclo `WHILE` o altro processo iterativo non è in genere il modo più efficiente per elaborare i dati in SQL Server.

Si consiglia di utilizzare una query basata sui set sui dati per ottenere gli stessi risultati, ove possibile

Examples

Utilizzando il ciclo While

Il ciclo `WHILE` può essere utilizzato come alternativa ai `CURSORS` . L'esempio seguente stamperà numeri da 0 a 99.

```
DECLARE @i int = 0;
WHILE(@i < 100)
BEGIN
   PRINT @i;
   SET @i = @i+1
END
```

Ciclo continuo con utilizzo della funzione di aggregazione minima

```
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
```

Leggi Ciclo WHILE online: https://riptutorial.com/it/sql-server/topic/4249/ciclo-while

Capitolo 12: Clausola OVER

Parametri

Parametro	Dettagli
PARTITION BY	Il / i campo / i che segue PARTITION BY è quello su cui si baserà il 'raggruppamento'

Osservazioni

La clausola OVER determina una finestra o un sottoinsieme di righe all'interno di una serie di risultati della query. È possibile applicare una funzione finestra per impostare e calcolare un valore per ogni riga nel set. La clausola OVER può essere utilizzata con:

- Funzioni di classificazione
- Funzioni aggregate

quindi qualcuno può calcolare valori aggregati come medie mobili, aggregati cumulativi, totali parziali o risultati N migliori per gruppo.

In un modo molto astratto possiamo dire che OVER si comporta come GROUP BY. Tuttavia, OVER viene applicato per campo / colonna e non per la query nel suo complesso come fa GROUP BY.

Nota n. 1: in SQL Server 2008 (R2), la clausola ORDER BY non può essere utilizzata con le funzioni della finestra di aggregazione (collegamento).

Examples

Utilizzo delle funzioni di aggregazione con OVER

Usando la tabella Cars , calcoleremo l'importo totale, massimo, minimo e medio di denaro speso da ogni cliente e molte volte (COUNT) ha portato una macchina per ripararla.

Id CustomerId MechanicId Status Status Costo totale

```
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'
```

Fai attenzione che l'utilizzo di OVER in questo modo non aggregherà le righe restituite. La query sopra riportata restituirà quanto segue:

Identificativo del cliente	Totale	Avg	Contare	min	Max
1	430	215	2	200	230
1	430	215	2	200	230

La / e riga / e duplicata / i potrebbe non essere quella utile ai fini della segnalazione.

Se si desidera semplicemente aggregare i dati, sarà meglio utilizzare la clausola GROUP BY insieme alle funzioni aggregate appropriate. Ad esempio:

```
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
```

Somma cumulativa

Usando la Tabella delle vendite degli articoli, cercheremo di scoprire come le vendite dei nostri articoli stanno aumentando attraverso le date. Per fare ciò calcoleremo la *somma cumulativa* delle vendite totali per ordine dell'articolo entro la data di vendita.

```
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
```

Utilizzo delle funzioni di aggregazione per trovare i record più recenti

Utilizzando il database delle biblioteche , cerchiamo di trovare l'ultimo libro aggiunto al database per ogni autore. Per questo semplice esempio, assumiamo un ID sempre crescente per ogni record aggiunto.

```
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
```

Invece di RANK, è possibile utilizzare altre due funzioni per ordinare. Nell'esempio precedente il

risultato sarà lo stesso, ma danno risultati diversi quando l'ordine dà più righe per ogni grado.

- `RANK()` : i duplicati ottengono lo stesso valore, il grado successivo prende in considerazione il numero di duplicati nel grado precedente
- `DENSE_RANK()` : i duplicati ottengono lo stesso valore, il grado successivo è sempre superiore a quello precedente
- `ROW_NUMBER()` : assegnerà a ciascuna riga un 'rank' unico, 'classifica' i duplicati casualmente

Ad esempio, se la tabella aveva una colonna non univoca CreationDate e l'ordine è stato eseguito in base a tale, la seguente query:

```
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
```

Potrebbe comportare:

Autore	Titolo	Data di creazione	RANGO	DENSE_RANK	ROW_NUMBER
Autore 1	Prenota 1	22/07/2016	1	1	1
Autore 1	Prenota 2	22/07/2016	1	1	2
Autore 1	Prenota 3	21/07/2016	3	2	3
Autore 1	Prenota 4	21/07/2016	3	2	4
Autore 1	Prenota 5	21/07/2016	3	2	5
Autore 1	Prenota 6	2016/04/07	6	3	6
Autore 2	Prenota 7	2016/04/07	1	1	1

Dividere i dati in bucket ugualmente partizionati usando NTILE

Supponiamo che tu disponga di punteggi degli esami per diversi esami e che tu voglia dividerli in quartili per esame.

```
-- 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` funziona alla grande quando hai davvero bisogno di un certo numero di secchi e ognuno è riempito all'incirca allo stesso livello. Si noti che sarebbe banale separare questi punteggi in percentili semplicemente usando `ntile(100)` .

Leggi Clausola OVER online: https://riptutorial.com/it/sql-server/topic/353/clausola-over

Capitolo 13: COALESCE

Sintassi

- COALESCE ([Colonna1], [Colonna2] .... [columnn]

Examples

Utilizzo di COALESCE per creare una stringa delimitata da virgola

Possiamo ottenere una stringa delimitata da virgole da più righe usando la coalescenza come mostrato di seguito.

Poiché viene utilizzata la variabile di tabella, è necessario eseguire un'intera query una volta. Per semplificare la comprensione, ho aggiunto il blocco BEGIN e END.

```
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
```

Esempio di base di Coalesce

`COALESCE()` restituisce il primo valore `NON NULL` in un elenco di argomenti. Supponiamo di avere una tabella contenente numeri di telefono e numeri di cellulare e di voler restituire solo una per ciascun utente. Per ottenerne solo uno, possiamo ottenere il primo valore `NON NULL` .

```
DECLARE @Table TABLE (UserID int, PhoneNumber varchar(12), CellNumber varchar(12))
INSERT INTO @Table (UserID, PhoneNumber, CellNumber)
VALUES
(1,'555-869-1123',NULL),
(2,'555-123-7415','555-846-7786'),
(3,NULL,'555-456-8521')
```

```
SELECT
    UserID,
    COALESCE(PhoneNumber, CellNumber)
FROM
    @Table
```

Ottenere il primo non null da un elenco di valori di colonna

```
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
```

Leggi COALESCE online: https://riptutorial.com/it/sql-server/topic/3234/coalesce

Capitolo 14: Colonne calcolate

Examples

Una colonna è calcolata da un'espressione

Una colonna calcolata viene calcolata da un'espressione che può utilizzare altre colonne nella stessa tabella. L'espressione può essere un nome di colonna non convertito, una costante, una funzione e qualsiasi combinazione di questi collegati da uno o più operatori.

Crea una tabella con una colonna calcolata

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

)
```

Il valore viene calcolato e memorizzato automaticamente nella colonna calcolata sull'inserimento di altri valori.

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

Semplice esempio che usiamo normalmente nelle tabelle di log

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

Questo dà una differenza di run in minuti per il runtime che sarà molto utile ..

Leggi Colonne calcolate online: https://riptutorial.com/it/sql-server/topic/5561/colonne-calcolate

Capitolo 15: COLONNERO CLUSTER

Examples

Tabella con indice CLUSTERED COLUMNSTORE

Se si desidera organizzare una tabella in formato column-store anziché in row store, aggiungere INDEX cci CLUSTERED COLUMNSTORE nella definizione di tabella:

```
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
)
```

Le tabelle COLUMSTORE sono migliori per le tabelle in cui prevedi scansioni e rapporti completi, mentre le tabelle di archivio righe sono migliori per le tabelle in cui leggere o aggiornare serie di righe più piccole.

Aggiunta dell'indice columnstore in cluster sulla tabella esistente

CREATE CLUSTERED COLUMNSTORE INDEX consente di organizzare una tabella in formato colonna:

```
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
```

Ricostruisci indice CLUSTERED COLUMNSTORE

L'indice dell'archivio di colonne in cluster può essere ricostruito se si dispone di molte righe eliminate:

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

La ricostruzione di CLUSTERED COLUMNSTORE "ricarica" i dati dalla tabella corrente in una nuova e applica nuovamente la compressione, rimuove le righe eliminate, ecc.

È possibile ricostruire una o più partizioni.

Leggi COLONNERO CLUSTER online: https://riptutorial.com/it/sql-server/topic/5774/colonnero-cluster

Capitolo 16: Common Language Runtime Integration

Examples

Abilita CLR sul database

Le procedure CLR non sono abilitate per impostazione predefinita. È necessario eseguire le seguenti query per abilitare CLR:

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

Inoltre, se alcuni moduli CLR necessitano di un accesso esterno, è necessario impostare la proprietà TRUSTWORTHY su ON nel proprio database:

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

Aggiunta di .dll che contiene moduli CLR Sql

Procedure, funzioni, trigger e tipi scritti nelle lingue .Net sono memorizzati in file .dll. Una volta creato il file .dll contenente le procedure CLR, è necessario importarlo in SQL Server:

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

PERMISSION_SET è sicuro per impostazione predefinita, il che significa che il codice in .dll non ha bisogno dell'autorizzazione per accedere a risorse esterne (ad esempio file, siti Web, altri server) e che non utilizzerà il codice nativo che può accedere alla memoria.

PERMISSION_SET = EXTERNAL_ACCESS viene utilizzato per contrassegnare gli assembly che contengono codice che accederà a risorse esterne.

puoi trovare informazioni sui file di assemblaggio CLR correnti nella vista sys.assemblies:

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

Creare la funzione CLR in SQL Server

Se hai creato la funzione .Net, l'hai compilata in .dll e l'hai importata nel server SQL come un assembly, puoi creare una funzione definita dall'utente che fa riferimento alla funzione in quell'assembly:

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

È necessario specificare il nome della funzione e della firma con i parametri di input e i valori di ritorno che corrispondono alla funzione .Net. Nella clausola AS EXTERNAL NAME è necessario specificare il nome assembly, il nome spazio / classe in cui viene inserita questa funzione e il nome del metodo nella classe che contiene il codice che verrà esposto come funzione.

È possibile trovare informazioni sulle funzioni CLR utilizzando la seguente query:

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

Crea CLR Tipo definito dall'utente in SQL Server

Se hai creato la classe .Net che rappresenta un tipo definito dall'utente, lo hai compilato in .dll e lo hai importato nel server SQL come un assembly, puoi creare una funzione definita dall'utente che fa riferimento a questa classe:

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

È necessario specificare il nome del tipo che verrà utilizzato nelle query T-SQL. Nella clausola EXTERNAL NAME è necessario specificare nome assembly, spazio dei nomi e nome classe.

Creare la procedura CLR in SQL Server

Se hai creato il metodo .Net in una classe, lo hai compilato in .dll e lo hai importato nel server SQL come un assembly, puoi creare una stored procedure definita dall'utente che fa riferimento al metodo in quell'assembly:

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

È necessario specificare il nome della procedura e la firma con i parametri di input che corrispondono al metodo .Net. Nella clausola AS EXTERNAL NAME è necessario specificare il nome assembly, il nome spazio / classe in cui viene inserita questa procedura e il nome del metodo nella classe che contiene il codice che verrà esposto come procedura.

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

Capitolo 17: Con l'opzione Ties

Examples

Dati di test

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

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

Di seguito è riportato l'output della tabella sopra, Come puoi vedere Id Column viene ripetuta tre volte ..

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

Ora controlliamo l'output usando l'ordine semplice di ..

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

Uscita: (l'output della query precedente non è garantito per essere lo stesso ogni volta)

```
Id   Name
1    B
```

Consente di eseguire la stessa query con l'opzione Ties ..

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

Produzione :

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

Come puoi vedere, SQL Server restituisce tutte le righe **vincolate con** Ordine per colonna. Vediamo un altro esempio per capire meglio

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

Produzione:

```
Id   Name
1    A
```

In Riepilogo, quando utilizziamo l'opzione Ties, SQL Server emette tutte le righe legate indipendentemente dal limite che imponiamo

Leggi Con l'opzione Ties online: https://riptutorial.com/it/sql-server/topic/2546/con-l-opzione-ties

Capitolo 18: Conversione di tipi di dati

Examples

PROVA PARSE

SQL Server 2012

Converte il tipo di dati stringa nel tipo di dati di destinazione (Data o Numerico).

Ad esempio, i dati di origine sono di tipo stringa e dobbiamo specificare il tipo di data. Se il tentativo di conversione fallisce, restituisce il valore NULL.

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

Valore_stringa: questo argomento è il valore sorgente che è il tipo NVARCHAR (4000).
Data_type: questo argomento è il tipo di dati di destinazione data o numerico.
Cultura - È un argomento facoltativo che aiuta a convertire il valore in formato cultura.
Supponiamo di voler visualizzare la data in francese, quindi devi passare il tipo di cultura come "Fr-FR". Se non passerai alcun nome di cultura valido, allora PARSE genererà un errore.

```
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
```

PROVA CONVERTITO

SQL Server 2012

Converte valore nel tipo di dati specificato e se la conversione fallisce, restituisce NULL. Ad esempio, il valore di origine in formato stringa e abbiamo bisogno di formato data / intero. Allora questo ci aiuterà a raggiungere lo stesso risultato.

Sintassi: TRY_CONVERT (data_type [(lunghezza)], espressione [, stile])

TRY_CONVERT () restituisce un cast di valori al tipo di dati specificato se il cast ha esito positivo; in caso contrario, restituisce null.

Data_type: il tipo di dati in cui convertire. Qui la lunghezza è un parametro facoltativo che aiuta a ottenere risultati nella lunghezza specificata.

Espressione: il valore da convertire
Stile: è un parametro opzionale che determina la formattazione. Supponiamo che tu voglia un formato data come "Maggio 18 2013", quindi devi passare lo stile come 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
```

PROVA CAST

SQL Server 2012

Converte valore nel tipo di dati specificato e se la conversione fallisce, restituisce NULL. Ad esempio, il valore sorgente in formato stringa e ne abbiamo bisogno in formato doppio / intero. Allora questo ci aiuterà a raggiungerlo.

Sintassi: TRY_CAST (espressione AS data_type [(lunghezza)])

TRY_CAST () restituisce un cast di valori al tipo di dati specificato se il cast ha esito positivo; in caso contrario, restituisce null.

Espressione: il valore sorgente che andrà a trasmettere.
Data_type: il tipo di dati di destinazione che verrà lanciato il valore di origine.
Lunghezza: è un parametro facoltativo che specifica la lunghezza del tipo di dati di destinazione.

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

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

lanciare

La funzione Cast () viene utilizzata per convertire una variabile di tipo dati o dati da un tipo di dati a un altro tipo di dati.

Sintassi

CAST ([Espressione] AS Datatype)

Il tipo di dati a cui stai trasmettendo un'espressione è il tipo di destinazione. Il tipo di dati dell'espressione da cui stai trasmettendo è il tipo di origine.

```
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.
```

Genera errore se fallisce

Convertire

Quando si convertono le espressioni da un tipo a un altro, in molti casi è necessario eseguire una stored procedure o un'altra routine per convertire i dati da un tipo datetime a un tipo varchar. La funzione Converti è usata per cose del genere. La funzione CONVERT () può essere utilizzata per visualizzare dati di data / ora in vari formati. Sintassi

CONVERT (data_type (length), expression, style)

Stile: valori di stile per la conversione datetime o smalldatetime ai dati carattere. Aggiungi 100 a un valore di stile per ottenere un anno a quattro cifre che include il secolo (aaaa).

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

13:27:16
```

Leggi Conversione di tipi di dati online: https://riptutorial.com/it/sql-server/topic/5034/conversione-di-tipi-di-dati

Capitolo 19: CREA VISTA

Examples

CREA VISTA

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

Le righe delle viste possono essere selezionate in modo simile alle tabelle:

```
SELECT FirstName
FROM view_EmployeeInfo
```

Puoi anche creare una vista con una colonna calcolata. Possiamo modificare la vista sopra come segue aggiungendo una colonna calcolata:

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

Questo punto di vista aggiunge una colonna aggiuntiva che verrà visualizzato quando si `SELECT` righe da esso. I valori in questa colonna aggiuntiva dipenderanno dai campi `FirstName` e `LastName` nella tabella `Employee` e aggiorneranno automaticamente dietro le quinte quando tali campi vengono aggiornati.

CREATE VIEW con la crittografia

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

CREATE VIEW con 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
```

Le viste possono utilizzare i join per selezionare i dati da numerose fonti come tabelle, funzioni di tabella o anche altre viste. Questo esempio utilizza le colonne FirstName e LastName dalla tabella Person e la colonna JobTitle dalla tabella Employee.

Ora questa vista può essere utilizzata per vedere tutte le righe corrispondenti per i Manager nel database:

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

CREA VISTA indicizzata

Per creare una vista con un indice, la vista deve essere creata utilizzando le parole chiave `WITH SCHEMABINDING` :

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

Ora è possibile creare qualsiasi cluster o indici non cluster:

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

Ci sono alcune limitazioni alle viste indicizzate:

- La definizione della vista può fare riferimento a una o più tabelle nello stesso database.
- Una volta creato l'indice cluster univoco, è possibile creare ulteriori indici non cluster sulla vista.
- È possibile aggiornare i dati nelle tabelle sottostanti, inclusi gli inserimenti, gli aggiornamenti,

le eliminazioni e persino i troncati.

- Non è possibile modificare le tabelle e le colonne sottostanti. La vista viene creata con l'opzione WITH SCHEMABINDING.
- Non può contenere COUNT, MIN, MAX, TOP, join esterni o poche altre parole chiave o elementi.

Per ulteriori informazioni sulla creazione di viste indicizzate è possibile leggere questo articolo MSDN

VISTE raggruppate

Una VISTA raggruppata si basa su una query con una clausola GROUP BY. Poiché ciascuno dei gruppi può avere più di una riga nella base da cui è stato creato, questi sono necessariamente VISTE di sola lettura. Tali VISTE di solito hanno una o più funzioni aggregate e sono utilizzati a fini di reporting. Sono anche utili per aggirare i punti deboli in SQL. Considera una VISTA che mostra la più grande vendita in ogni stato. La query è semplice:

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;
```

VISIONI UNION-ED

Le VISUALIZZA basate su un'operazione UNION o UNION ALL sono di sola lettura perché non esiste un unico modo per mappare una modifica su una sola riga in una delle tabelle di base. L'operatore UNION rimuoverà le righe duplicate dai risultati. Sia gli operatori UNION che UNION ALL nascondono da quale tabella provengono le righe. Tali VISUALIZZAZIONI devono utilizzare a, poiché le colonne in UNION [ALL] non hanno nomi propri. In teoria, un UNION di due tabelle disgiunte, nessuna delle quali ha righe duplicate in sé dovrebbe essere aggiornabile.

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));
```

Leggi CREA VISTA online: https://riptutorial.com/it/sql-server/topic/3815/crea-vista

Capitolo 20: crittografia

Parametri

Parametri opzionali	Dettagli
`WITH PRIVATE KEY`	Per CREATE CERTIFICATE, è possibile specificare una chiave privata: `(FILE='D:\Temp\CertTest\private.pvk', DECRYPTION BY PASSWORD = 'password');`

Osservazioni

La creazione di un certificato DER funzionerà correttamente. Tuttavia, quando viene utilizzato un certificato Base64, il server SQL si lamenterà con il messaggio criptico:

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

Importa il tuo certificato Base64 nell'archivio certificati del tuo sistema operativo per poterlo riesportare in formato binario DER.

Un'altra cosa importante da fare è avere una gerarchia di crittografia in modo che uno protegga l'altro, fino al livello del sistema operativo. Vedi l'articolo su 'Crittografia del database / TDE'

Per ulteriori informazioni sulla creazione di certificati, visitare: https://msdn.microsoft.com/en-us/library/ms187798.aspx

Per ulteriori informazioni sulla crittografia del database / TDE, consultare: https://msdn.microsoft.com/en-us/library/bb934049.aspx

Per ulteriori informazioni sulla crittografia dei dati, consultare: https://msdn.microsoft.com/en-us/library/ms188061.aspx

Examples

Crittografia per certificato

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

Crea il certificato

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

Di solito, si cripterebbe con una chiave simmetrica, quella chiave sarebbe crittografata dalla chiave asimmetrica (chiave pubblica) dal certificato.

Inoltre, tieni presente che la crittografia è limitata a determinate lunghezze in base alla lunghezza della chiave e restituisce NULL altrimenti. Microsoft scrive: "I limiti sono: una chiave RSA a 512 bit può crittografare fino a 53 byte, una chiave a 1024 bit può crittografare fino a 117 byte e una chiave a 2048 bit può crittografare fino a 245 byte."

EncryptByAsymKey ha gli stessi limiti. Per UNICODE questo sarebbe diviso per 2 (16 bit per carattere), quindi 58 caratteri per una chiave a 1024 bit.

Crittografia del database

```
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
```

Questo utilizza 'Transparent Data Encryption' (TDE)

Crittografia con chiave simmetrica

```
-- 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');
```

Crittografia per passphrase

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

Questo verrà anche crittografato, ma poi per passphrase invece di chiave asimmetrica (certificato) o con una chiave simmetrica esplicita.

Leggi crittografia online: https://riptutorial.com/it/sql-server/topic/7096/crittografia

Capitolo 21: croce si applicano

Examples

Unisci le righe della tabella con le righe generate dinamicamente da una cella

CROSS APPLY consente di "unire" le righe da una tabella con le righe generate dinamicamente restituite da una funzione valore di tabella.

Immagina di avere una tabella Company con una colonna che contiene una serie di prodotti (colonna ProductList) e una funzione che analizza questi valori e restituisce un set di prodotti. È possibile selezionare tutte le righe da una tabella Società, applicare questa funzione su una colonna ProductList e "unire" i risultati generati con la riga della società madre:

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

Per ciascuna riga, il valore della cella *ProductList* verrà fornito alla funzione e la funzione restituirà tali prodotti come un insieme di righe che possono essere uniti alla riga padre.

Unisci le righe della tabella con l'array JSON memorizzato nella cella

CROSS APPLY consente di "unire" le righe da una tabella con la raccolta di oggetti JSON archiviati in una colonna.

Immagina di avere una tabella Company con una colonna che contiene una serie di prodotti (colonna ProductList) formattata come array JSON. La funzione del valore di tabella OPENJSON può analizzare questi valori e restituire l'insieme di prodotti. È possibile selezionare tutte le righe da una tabella Società, analizzare i prodotti JSON con OPENJSON e "unire" i risultati generati con la riga della società madre:

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

Per ciascuna riga, il valore della cella *ProductList* verrà fornito alla funzione OPENJSON che trasformerà gli oggetti JSON in righe con lo schema definito nella clausola WITH.

Filtra le righe per valori di array

Se si memorizza un elenco di tag in una riga come valori separati da virgola, la funzione *STRING_SPLIT* consente di trasformare l'elenco di tag in una tabella di valori. **CROSS APPLY** consente di "unire" i valori analizzati dalla funzione *STRING_SPLIT* con una riga padre.

Immagina di avere una tabella di prodotti con una colonna che contiene una serie di tag separati

da virgole (ad esempio promo, vendite, nuovi). STRING_SPLIT e CROSS APPLY ti consentono di unire le righe di prodotto con i loro tag in modo da poter filtrare i prodotti in base ai tag:

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

Per ogni riga, il valore della cella *Tag* verrà fornito alla funzione STRING_SPLIT che restituirà i valori dei tag. Quindi puoi filtrare le righe con questi valori.

Nota: la funzione *STRING_SPLIT* non è disponibile prima di **SQL Server 2016**

Leggi croce si applicano online: https://riptutorial.com/it/sql-server/topic/5462/croce-si-applicano

Capitolo 22: Cursori

Sintassi

- DECLARE cursor_name CURSOR [LOCAL | GLOBALE]
  - [FORWARD_ONLY | SCORRERE]
    [STATICO | KEYSET | DINAMICO | AVANTI VELOCE]
    [READ_ONLY | SCROLL_LOCKS | OTTIMISTA]
    [TYPE_WARNING]
  - PER select_statement
  - [FOR UPDATE [OF column_name [, ... n]]]

Osservazioni

Normalmente si vorrebbe evitare di utilizzare i cursori in quanto possono avere un impatto negativo sulle prestazioni. Tuttavia, in alcuni casi speciali potrebbe essere necessario eseguire il loop del record di dati tramite record ed eseguire alcune azioni.

Examples

Cursore di avanzamento di base

Normalmente si vorrebbe evitare di utilizzare i cursori in quanto possono avere un impatto negativo sulle prestazioni. Tuttavia, in alcuni casi speciali potrebbe essere necessario eseguire il loop del record di dati tramite record ed eseguire alcune azioni.

```
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
```

Sintassi del cursore rudimentale

Una semplice sintassi del cursore, che funziona su alcune righe di prova di esempio:

```
/* 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;
```

Risultati da SSMS. Si noti che queste sono tutte query separate, non sono in alcun modo unificate. Si noti come il motore di query elabora ciascuna iterazione una alla volta anziché come

una serie.

Id	Val
1	foo
(1 riga (e) interessata)	

Id	Val
2	Bar
(1 riga (e) interessata)	

Id	Val
3	Baz
(1 riga (e) interessata)	

Leggi Cursori online: https://riptutorial.com/it/sql-server/topic/870/cursori

Capitolo 23: Database di sistema - TempDb

Examples

Identificare l'utilizzo di TempDb

La seguente query fornirà informazioni sull'utilizzo di TempDb. Analizzando i conteggi è possibile identificare quale cosa sta influenzando TempDb

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

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

Dettagli del database TempDB

La query sottostante può essere utilizzata per ottenere i dettagli del database TempDB:

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

O

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

Con l'aiuto di DMV sotto, è possibile verificare la quantità di spazio TempDb utilizzato dalla sessione. Questa query è molto utile durante il debug di problemi TempDb

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

Leggi Database di sistema - TempDb online: https://riptutorial.com/it/sql-server/topic/4427/database-di-sistema---tempdb

Capitolo 24: Date

Sintassi

- EOMONTH (*start_date* [, month_to_add])

Osservazioni

come per https://msdn.microsoft.com/en-us/library/ms187819.aspx , `DateTime` s sono solo precisi a 3 ms.

Arrotondamento dei valori datetime Fractional Second Precision datetime sono arrotondati a incrementi di .000, .003 o .007 secondi, come mostrato nella seguente tabella.

Valore specificato dall'utente	Sistema memorizzato
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	
------	------

Se è richiesta maggiore precisione, è necessario utilizzare `time` , `datetime2` o `datetimeoffset` .

Examples

Formattazione di data e ora tramite CONVERT

È possibile utilizzare la funzione CONVERT per eseguire il cast di un datatype datetime su una stringa formattata.

```
SELECT GETDATE() AS [Result] -- 2016-07-21 07:56:10.927
```

È inoltre possibile utilizzare alcuni codici incorporati per convertire in un formato specifico. Ecco le opzioni integrate in SQL Server:

```
DECLARE @convert_code INT = 100 -- See Table Below
SELECT CONVERT(VARCHAR(30), GETDATE(), @convert_code) AS [Result]
```

@convert_code	Risultato
100	"21 lug 2016 7:56 AM"
101	"2016/07/21"
102	"2016/07/21"
103	"21/07/2016"
104	"2016/07/21"
105	"21-07-2016"
106	"21 lug 2016"
107	"21 lug 2016"
108	"07:57:05"
109	"21 luglio 2016 7: 57: 45: 707AM"
110	"2016/07/21"
111	"2016/07/21"
112	"20160721"
113	"21 luglio 2016 07: 57: 59: 553"
114	"07: 57: 59: 553"
120	"21-07-2017 07:57:59"

@convert_code	Risultato
121	"07-07-2017 07: 57: 59,553"
126	"2016-07-21T07: 58: 34,340"
127	"2016-07-21T07: 58: 34,340"
130	"16 ???? 1437 7: 58: 34: 340AM"
131	"16/10/1437 7: 58: 34: 340AM"

```
SELECT GETDATE() AS [Result]                                -- 2016-07-21 07:56:10.927
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),100) AS [Result] -- Jul 21 2016  7:56AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),101) AS [Result] -- 07/21/2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),102) AS [Result] -- 2016.07.21
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),103) AS [Result] -- 21/07/2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),104) AS [Result] -- 21.07.2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),105) AS [Result] -- 21-07-2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),106) AS [Result] -- 21 Jul 2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),107) AS [Result] -- Jul 21, 2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),108) AS [Result] -- 07:57:05
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),109) AS [Result] -- Jul 21 2016  7:57:45:707AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),110) AS [Result] -- 07-21-2016
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),111) AS [Result] -- 2016/07/21
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),112) AS [Result] -- 20160721
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),113) AS [Result] -- 21 Jul 2016 07:57:59:553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),114) AS [Result] -- 07:57:59:553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),120) AS [Result] -- 2016-07-21 07:57:59
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),121) AS [Result] -- 2016-07-21 07:57:59.553
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),126) AS [Result] -- 2016-07-21T07:58:34.340
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),127) AS [Result] -- 2016-07-21T07:58:34.340
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),130) AS [Result] -- 16 ???? 1437  7:58:34:340AM
UNION SELECT CONVERT(VARCHAR(30),GETDATE(),131) AS [Result] -- 16/10/1437  7:58:34:340AM
```

Formattazione di data e ora tramite FORMAT

SQL Server 2012

È possibile utilizzare la nuova funzione: `FORMAT()` .

Usando questo puoi trasformare i tuoi campi `DATETIME` nel tuo formato `VARCHAR` personalizzato.

Esempio

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'dddd, MMMM dd, yyyy hh:mm:ss tt')
```

Lunedì 5 settembre 2016 12:01:02

argomenti

Dato che `DATETIME` stato formattato nel `2016-09-05 00:01:02.333` , il seguente grafico mostra quale

sarebbe il loro output per l'argomento fornito.

Discussione	Produzione
aaaa	2016
aa	16
MMMM	settembre
MM	09
M	9
dddd	Lunedi
ddd	Mon
dd	05
d	5
HH	00
H	0
hh	12
h	12
mm	01
m	1
ss	02
S	2
TT	AM
t	UN
F F F	333
ff	33
f	3

È anche possibile fornire un singolo argomento alla funzione `FORMAT()` per generare un output preformattato:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'U')
```

Lunedì 5 settembre 2016 04:01:02

Argomento singolo	Produzione
D	Lunedì 5 settembre 2016
d	2016/09/05
F	Lunedì 5 settembre 2016 12:01:02
f	Lunedì 5 settembre 2016 alle 12:01
sol	9/5/2016 12:01:02 AM
g	9/5/2016 alle 12:01
M	5 settembre
O	2016-09-05T00: 01: 02,3330000
R	Lun, 05 Set 2016 00:01:02 GMT
S	2016-09-05T00: 01: 02
T	12:01:02 AM
t	Alle 12:01
U	Lunedì 5 settembre 2016 04:01:02
u	2016-09-05 00: 01: 02Z
Y	Settembre 2016

Nota: l'elenco precedente utilizza la cultura `en-US` . Una diversa cultura può essere specificata per il `FORMAT()` tramite il terzo parametro:

```
DECLARE @Date DATETIME = '2016-09-05 00:01:02.333'

SELECT FORMAT(@Date, N'U', 'zh-cn')
```

2016 9 5 4:01:02

Ottieni l'ora corrente

Le funzioni incorporate `GETDATE` e `GETUTCDATE` restituiscono ciascuna la data e l'ora correnti senza un

offset del fuso orario.

Il valore di ritorno di entrambe le funzioni è basato sul sistema operativo del computer su cui è in esecuzione l'istanza di SQL Server.

Il valore restituito di GETDATE rappresenta l'ora corrente nello stesso fuso orario del sistema operativo. Il valore di ritorno di GETUTCDATE rappresenta l'ora UTC corrente.

Entrambe le funzioni possono essere incluse nella clausola `SELECT` di una query o come parte dell'espressione booleana nella clausola `WHERE` .

Esempi:

```
-- 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()
```

Ci sono alcune altre funzioni incorporate che restituiscono diverse variazioni della data-ora corrente:

```
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 per l'aggiunta e la sottrazione dei periodi di tempo

Sintassi generale:

```
DATEADD (datepart , number , datetime_expr)
```

Per aggiungere una misura temporale, il `number` deve essere positivo. Per sottrarre una misura del tempo, il `number` deve essere negativo.

Esempi

```
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
```

NOTA: `DATEADD` accetta anche le abbreviazioni nel parametro `datepart` . L'uso di queste abbreviazioni è generalmente scoraggiato in quanto possono essere fonte di confusione (`m` vs `mi` , `ww` vs `w` , ecc.).

Data di riferimento delle parti

Questi sono i valori di `datepart` disponibili per le funzioni di data e ora:

datepart	Abbreviazioni
anno	yy, yyyy
trimestre	qq, q
mese	mm, m
dayofyear	dy, y
giorno	dd, d
settimana	wk, ww
giorno feriale	dw, w
ora	hh
minuto	mi, n
secondo	ss, s
millisecondo	Signorina
microsecondo	mcs
nanosecondo	ns

NOTA : l'uso delle abbreviazioni è generalmente scoraggiato in quanto possono essere fonte di confusione (`m` vs `mi` , `ww` vs `w` , ecc.). La versione lunga della rappresentazione di `datepart` promuove la chiarezza e la leggibilità e dovrebbe essere utilizzata ogni qualvolta possibile (`month` , `minute` , `week` , `weekday` , ecc.).

DATEDIFF per il calcolo delle differenze di periodo

Sintassi generale:

```
DATEDIFF (datepart, datetime_expr1, datetime_expr2)
```

Restituirà un numero positivo se `datetime_expr` è nel passato relativo a `datetime_expr2` e un numero negativo in caso contrario.

Esempi

```
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
```

NOTA: `DATEDIFF` accetta anche le abbreviazioni nel parametro `datepart` . L'uso di queste abbreviazioni è generalmente scoraggiato in quanto possono essere fonte di confusione (`m` vs `mi` , `ww` vs `w` , ecc.).

`DATEDIFF` può anche essere utilizzato per determinare l'offset tra UTC e l'ora locale di SQL Server. La seguente istruzione può essere utilizzata per calcolare l'offset tra UTC e l'ora locale (incluso il fuso orario).

```
select  DATEDIFF(hh, getutcdate(), getdate()) as 'CentralTimeOffset'
```

DATEPART & DATENAME

`DATEPART` restituisce la `datepart` specificata dell'espressione datetime specificata come valore numerico.

`DATENAME` restituisce una stringa di caratteri che rappresenta la `datepart` specificata della data specificata. In pratica, `DATENAME` è utile soprattutto per ottenere il nome del mese o il giorno della settimana.

Esistono anche alcune funzioni abbreviate per ottenere l'anno, il mese o il giorno di un'espressione datetime, che si comportano come `DATEPART` con le rispettive unità di `datepart` .

Sintassi:

```
DATEPART ( datepart , datetime_expr )
DATENAME ( datepart , datetime_expr )
DAY ( datetime_expr )
MONTH ( datetime_expr )
YEAR ( datetime_expr )
```

Esempi:

```
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
```

NOTA: `DATEPART` e `DATENAME` accettano anche le abbreviazioni nel parametro `datepart` . L'uso di queste abbreviazioni è generalmente scoraggiato in quanto possono essere fonte di confusione (`m` vs `mi` , `ww` vs `w` , ecc.).

Ottenere l'ultimo giorno di un mese

Utilizzando le funzioni `DATEADD` e `DATEDIFF` , è possibile restituire l'ultima data del mese.

```
SELECT DATEADD(d, -1, DATEADD(m, DATEDIFF(m, 0, '2016-09-23') + 1, 0))
-- 2016-09-30 00:00:00.000
```

SQL Server 2012

La funzione `EOMONTH` fornisce un modo più conciso per restituire l'ultima data di un mese e ha un parametro facoltativo per compensare il mese.

```
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
```

Restituisce solo la data da un DateTime

Esistono molti modi per restituire una data da un oggetto DateTime

1. `SELECT CONVERT(Date, GETDATE())`
2. `SELECT DATEADD(dd, 0, DATEDIFF(dd, 0, GETDATE()))` restituisce 2016-07-21 00: 00: 00.000
3. `SELECT CAST(GETDATE() AS DATE)`
4. `SELECT CONVERT(CHAR(10),GETDATE(),111)`
5. `SELECT FORMAT(GETDATE(), 'yyyy-MM-dd')`

Nota che le opzioni 4 e 5 restituiscono una stringa, non una data.

Crea una funzione per calcolare l'età di una persona in una data specifica

Questa funzione richiede 2 parametri datetime, il DOB e una data per verificare l'età in

```
  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
```

ad esempio per verificare l'età di oggi di qualcuno nato il 1 ° gennaio 2000

```
SELECT  dbo.Calc_Age('2000-01-01',Getdate())
```

OGGETTO DELLA DATA DELLA PIATTAFORMA CROSS

SQL Server 2012

In Transact SQL, è possibile definire un oggetto come `Date` (o `DateTime`) utilizzando la funzione `[DATEFROMPARTS][1]` (o `[DATETIMEFROMPARTS][1]`) come segue:

```
 DECLARE @myDate DATE=DATEFROMPARTS(1988,11,28)
 DECLARE @someMoment DATETIME=DATEFROMPARTS(1988,11,28,10,30,50,123)
```

I parametri forniti sono Anno, Mese, Giorno per la funzione `DATEFROMPARTS` e, per la funzione `DATETIMEFROMPARTS` , è necessario fornire anno, mese, giorno, ora, minuti, secondi e millisecondi.

Questi metodi sono utili e meritano di essere utilizzati perché l'utilizzo della stringa semplice per creare una data (o datetime) potrebbe non riuscire a seconda delle impostazioni della regione, del percorso o della data della macchina host.

Formato data esteso

Formato data	Dichiarazione SQL	Uscita di esempio
AA-MM-DD	SELEZIONA DESTRA (CONVERT (VARCHAR (10), SYSDATETIME (), 20), 8) AS [YY-MM-DD] SELEZIONA REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 11), '/', '-') AS [YY-MM-DD]	11-06-08
AAAA-MM-DD	SELECT CONVERT (VARCHAR (10), SYSDATETIME (),	2011-06-08

Formato data	Dichiarazione SQL	Uscita di esempio
	120) AS [YYYY-MM-DD] SELEZIONA REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 111), '/', '-') AS [YYYY-MM-DD]	
AAAA-MD	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY-MD]	2011/06/08
YY-MD	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ( )) AS VARCHAR (2)) AS [YY-MD]	11-6-8
MD-AAAA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [MD-YYYY]	2011/06/08
MD-YY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [MD-YY]	6-8-11
DM-AAAA	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)) AS [DM-YYYY]	2011/08/06
DM-YY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RIGHT (CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [DM-YY]	8-6-11
AA-MM	SELEZIONA DESTRA (CONVERT (VARCHAR (7), SYSDATETIME (), 20), 5) AS [YY-MM] SELEZIONARE SUBSTRING (CONVERT (VARCHAR (10), SYSDATETIME (), 120), 3, 5) AS [YY-MM]	11-06
AAAA-MM	SELEZIONA CONVERT (VARCHAR (7), SYSDATETIME (), 120) AS [YYYY-MM]	2011-06
YY-M	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YY-M]	11-6

Formato data	Dichiarazione SQL	Uscita di esempio
AAAA-M	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY-M]	2011-6
MM-AA	SELEZIONA DESTRA (CONVERT (VARCHAR (8), SYSDATETIME (), 5), 5) AS [MM-YY] SELEZIONARE SUBSTRING (CONVERT (VARCHAR (8), SYSDATETIME (), 5), 4, 5) AS [MM-YY]	06-11
MM-AAAA	SELEZIONA DESTRA (CONVERT (VARCHAR (10), SYSDATETIME (), 105), 7) AS [MM-YYYY]	06-2011
M-YY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M-YY]	6-11
M-AAAA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M-YYYY]	6-2011
MM-DD	SELEZIONA CONVERT (VARCHAR (5), SYSDATETIME (), 10) AS [MM-DD]	06-08
GG-MM	SELEZIONA CONVERT (VARCHAR (5), SYSDATETIME (), 5) AS [DD-MM]	08-06
MD	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [MD]	6-8
DM	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '-' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [DM]	8-6
M / D / AAAA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)) AS [M / D / YYYY]	2011/06/08
M / D / AA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M / D / YY]	6/8/11
D / M / AAAA	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONTH (SYSDATETIME ()) AS	8/6/2011

Formato data	Dichiarazione SQL	Uscita di esempio
	VARCHAR (2)) + '/' + CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)) AS [D / M / YYYY]	
D / M / YY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RIGHT (CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [D / M / YY]	8/6/11
AAAA / M / D	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY / M / D]	2011/06/08
YY / M / D	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ( )) AS VARCHAR (2)) AS [YY / M / D]	11/6/8
MM / AA	SELEZIONA DESTRA (CONVERT (VARCHAR (8), SYSDATETIME (), 3), 5) AS [MM / AA]	06/11
MM / AAAA	SELEZIONA DESTRA (CONVERT (VARCHAR (10), SYSDATETIME (), 103), 7) AS [MM / YYYY]	06/2011
M / AA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M / YY]	6/11
M / AAAA	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) AS [M / YYYY]	6/2011
AA / MM	SELEZIONA CONVERT (VARCHAR (5), SYSDATETIME (), 11) AS [YY / MM]	11/06
AAAA / MM	SELEZIONA CONVERT (VARCHAR (7), SYSDATETIME (), 111) AS [YYYY / MM]	2011/06
YY / M	SELECT RIGHT (CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)), 2) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YY / M]	11/6
AAAA / M	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY / M]	2011/6
MM / DD	SELEZIONA CONVERT (VARCHAR (5), SYSDATETIME (),	06/08

Formato data	Dichiarazione SQL	Uscita di esempio
	1) AS [MM / DD]	
GG / MM	SELEZIONA CONVERT (VARCHAR (5), SYSDATETIME (), 3) AS [DD / MM]	08/06
M / D	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [M / D]	6/8
D / M	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '/' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [D / M]	8/6
MM.GG.AAAA	SELEZIONA REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 101), '/', '.') AS [MM.DD.YYYY]	06.08.2011
MM.GG.AA	SELEZIONA REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 1), '/', '.') AS [MM.DD.YY]	06.08.11
MDYYYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)) COME [MDYYYY]	6.8.2011
MDYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + DESTRA (CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [MDYY]	6.8.11
GG.MM.AAAA	SELEZIONA CONVERT (VARCHAR (10), SYSDATETIME (), 104) AS [GG.MM.AAAA]	08.06.2011
GG.MM.AA	SELEZIONA CONVERT (VARCHAR (10), SYSDATETIME (), 4) AS [GG.MM.AA]	08.06.11
DMYYYY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)) COME [DMYYYY]	8.6.2011
DMYY	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + DESTRA (CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [DMYY]	8.6.11
YYYY.MD	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '.' + CAST (MONTH (SYSDATETIME ()) AS	2011/06/08

Formato data	Dichiarazione SQL	Uscita di esempio
	VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY.MD]	
YY.MD	SELEZIONA DESTRA (CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)), 2) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) AS [YY.MD]	11.6.8
mm.aaaa	SELEZIONA DESTRA (CONVERT (VARCHAR (10), SYSDATETIME (), 104), 7) AS [MM.YYYY]	06.2011
MM.YY	SELEZIONA DESTRA (CONVERT (VARCHAR (8), SYSDATETIME (), 4), 5) AS [MM.YY]	06.11
M.YYYY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)) COME [M.YYYY]	6,2011
M.YY	SELECT CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) + '.' + DESTRA (CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [M.YY]	6.11
YYYY.MM	SELEZIONA CONVERT (VARCHAR (7), SYSDATETIME (), 102) AS [YYYY.MM]	2011.06
YY.MM	SELEZIONA CONVERT (VARCHAR (5), SYSDATETIME (), 2) AS [YY.MM]	11.06
YYYY.M	SELECT CAST (YEAR (SYSDATETIME ()) AS VARCHAR (4)) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YYYY.M]	2.011,6
YY.M	SELEZIONA DESTRA (CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)), 2) + '.' + CAST (MONTH (SYSDATETIME ()) AS VARCHAR (2)) AS [YY.M]	11.6
mm.dd	SELEZIONA DESTRA (CONVERT (VARCHAR (8), SYSDATETIME (), 2), 5) AS [MM.DD]	06.08
DD.MM	SELEZIONA CONVERT (VARCHAR (5), SYSDATETIME (), 4) AS [DD.MM]	08.06
MMGGAAAA	SELEZIONA REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 101), '/', '') AS [MMDDYYYY]	06082011
MMDDYY	SELEZIONA REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 1), '/', '') AS [MMDDYY]	060.811

Formato data	Dichiarazione SQL	Uscita di esempio
DDMMYYYY	SELEZIONA SOSTITUISCI (CONVERT (VARCHAR (10), SYSDATETIME (), 103), '/', '') AS [GGMMAAAA]	08062011
GGMMAA	SELEZIONA SOSTITUISCI (CONVERT (VARCHAR (8), SYSDATETIME (), 3), '/', '') AS [DDMMYY]	080.611
MMYYYY	SELEZIONA DESTRA (REPLACE (CONVERT (VARCHAR (10), SYSDATETIME (), 103), '/', ''), 6) AS [MMYYYY]	062011
MMAA	SELEZIONA DESTRA (REPLACE (CONVERT (VARCHAR (8), SYSDATETIME (), 3), '/', ''), 4) AS [MMYY]	0611
AAAAMM	SELEZIONA CONVERT (VARCHAR (6), SYSDATETIME (), 112) AS [YYYYMM]	201106
AAMM	SELEZIONA CONVERT (VARCHAR (4), SYSDATETIME (), 12) AS [YYMM]	1106
Mese GG, AAAA	SELEZIONA DATENAME (MESE, SYSDATETIME ()) + '' + DESTRA ('0' + DATENAME (GIORNO, SYSDATETIME ()), 2) + ',' + DATENAME (ANNO, SYSDATETIME ()) AS [Mese DD, YYYY]	8 giugno 2011
Lun YYYY	SELEZIONA A SINISTRA (DATENAME (MONTH, SYSDATETIME ()), 3) + '' + DATENAME (ANNO, SYSDATETIME ()) AS [Mon YYYY]	Giugno 2011
Mese AAAA	SELEZIONA DATENAME (MESE, SYSDATETIME ()) + '' + DATENAME (ANNO, SYSDATETIME ()) AS [Mese YYYY]	Giugno 2011
DD Month	SELECT RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + '' + DATENAME (MONTH, SYSDATETIME ()) AS [DD Month]	08 giugno
Mese DD	SELEZIONA DATENAME (MONTH, SYSDATETIME ()) + '' + RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) AS [Month DD]	8 giugno
DD Mese AA	SELECT CAST (DAY (SYSDATETIME ()) AS VARCHAR (2)) + '' + DATENAME (MM, SYSDATETIME ()) + '' + RIGHT (CAST (ANNO (SYSDATETIME ()) AS VARCHAR (4)), 2) AS [DD Mese YY]	08 giugno
DD Mese AAAA	SELECT RIGHT ('0' + DATENAME (DAY, SYSDATETIME ()), 2) + '' + DATENAME (MONTH, SYSDATETIME ()) + '' + DATENAME (ANNO, SYSDATETIME ()) AS [DD Month YYYY]	08 giugno 2011

Formato data	Dichiarazione SQL	Uscita di esempio
Mon-YY	SELEZIONA REPLACE (DESTRA (CONVERT (VARCHAR (9), SYSDATETIME (), 6), 6), '', '-') AS [Mon-YY]	Jun-08
Mon-AAAA	SELEZIONA SOSTITUISCI (DESTRA (CONVERSO (VARCHAR (11), SYSDATETIME (), 106), 8), '', '-') AS [Mon-YYYY]	Giu-2011
DD-Mon-YY	SELEZIONA SOSTITUISCI (CONVERT (VARCHAR (9), SYSDATETIME (), 6), '', '-') AS [DD-Mon-YY]	08-giu-11
DD-Mon-AAAA	SELEZIONA REPLACE (CONVERT (VARCHAR (11), SYSDATETIME (), 106), '', '-') AS [DD-Mon-YYYY]	08-Giu-2011

Leggi Date online: https://riptutorial.com/it/sql-server/topic/1471/date

Capitolo 25: Dati spaziali

introduzione

Esistono 2 tipi di dati spaziali

Geometria X / Y sistema di coordinate per una superficie piana

Geografia Sistema di coordinate di latitudine / longitudine per una superficie curva (la terra). Esistono diverse proiezioni di superfici curve, pertanto ogni spazio geografico deve consentire a SQL Server di sapere quale proiezione utilizzare. Il solito Spatial Reference ID (SRID) è 4326, che misura le distanze in chilometri. Questo è lo SRID predefinito utilizzato nella maggior parte delle mappe web

Examples

PUNTO

Crea un singolo punto. Questo sarà un punto geografico o geografico in base alla classe utilizzata.

Parametro	Dettaglio
Lat o X	Espressione float che rappresenta la coordinata x del punto generato
Lungo o Y	Espressione float che rappresenta la coordinata y del punto generato
Stringa	Testo ben noto (WKB) di una forma geometrica / geografica
Binario	Well Known Binary (WKB) di una forma geometrica / geografica
SRID	Espressione int che rappresenta l'ID di riferimento spaziale (SRID) dell'istanza di geometria / geografia che si desidera restituire

```
--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)
```

Leggi Dati spaziali online: https://riptutorial.com/it/sql-server/topic/6816/dati-spaziali

Capitolo 26: DBCC

Examples

Comandi di manutenzione DBCC

I comandi DBCC consentono all'utente di mantenere spazio nel database, pulire le cache, ridurre i database e le tabelle.

Gli esempi sono:

```
DBCC DROPCLEANBUFFERS
```

Rimuove tutti i buffer puliti dal pool buffer e gli oggetti columnstore dal pool di oggetti columnstore.

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

Rimuove tutte le query SQL nella cache di piano. Ogni nuovo piano verrà ricompilato: è possibile specificare l'handle del piano, l'handle di query per pulire i piani per il piano di query specifico o l'istruzione SQL.

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

Pulisce tutte le voci memorizzate nella cache create dal sistema. Può **cancellare** le voci o = in tutto o in alcuni pool di risorse specificate (**myresourcepool** nell'esempio sopra)

```
DBCC FLUSHAUTHCACHE
```

Elimina la cache di autenticazione del database contenente informazioni sugli accessi e le regole del firewall.

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

Riduce il database MyDB al 10%. Il secondo parametro è facoltativo. Puoi usare l'id del database invece del nome.

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

Restringe il file di dati denominato DataFile1 nel database corrente. La dimensione del target è 7 MB (il parametro tis è facoltativo).

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

Recupera uno spazio dalla tabella specificata

Dichiarazioni di convalida DBCC

I comandi DBCC consentono all'utente di convalidare lo stato del database.

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

Il vincolo di controllo viene aggiunto con opzioni nocheck, quindi non verrà verificato sui dati esistenti. DBCC attiverà il controllo dei vincoli.

I seguenti comandi DBCC verificano l'integrità del database, della tabella o del catalogo:

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

Dichiarazioni informative DBCC

I comandi DBCC possono mostrare informazioni sugli oggetti del database.

```
DBCC PROCCACHE
```

Visualizza le informazioni in un formato tabella sulla cache delle procedure.

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

Restituisce il buffer di output corrente in formato esadecimale e ASCII per il session_id specificato (e request_id facoltativo).

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

Visualizza l'ultima istruzione inviata da un client a un'istanza di Microsoft SQL Server.

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

Comandi DBCC Trace

I flag di traccia in SQL Server vengono utilizzati per modificare il comportamento del server SQL, attivare / disattivare alcune funzionalità. I comandi DBCC possono controllare i flag di traccia:

L'esempio seguente attiva il flag di traccia 3205 a livello globale e 3206 per la sessione corrente:

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

Nell'esempio seguente viene disattivato il flag di traccia 3205 a livello globale e 3206 per la sessione corrente:

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

Nell'esempio seguente viene visualizzato lo stato dei flag di traccia 2528 e 3205:

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

Dichiarazione DBCC

Le istruzioni DBCC fungono da comandi della console di database per SQL Server. Per ottenere le informazioni sulla sintassi per il comando DBCC specificato, utilizzare l'istruzione DBCC HELP (...).

L'esempio seguente restituisce tutte le istruzioni DBCC per cui è disponibile la Guida:

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

L'esempio seguente restituisce le opzioni per l'istruzione DBCC CHECKDB:

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

Leggi DBCC online: https://riptutorial.com/it/sql-server/topic/7316/dbcc

Capitolo 27: dbmail

Sintassi

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

Examples

Invia una semplice email

Questo codice invia una semplice e-mail di solo testo a recipient@someaddress.com

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

Invia risultati di una query

Questo allega i risultati della query `SELECT * FROM Users` e lo invia a `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;
```

Invia email HTML

Il contenuto HTML deve essere passato a `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>';
```

Quindi usa la variabile `@html` con l' `@body argument` . La stringa HTML può anche essere passata direttamente a `@body` , sebbene possa rendere il codice più difficile da leggere.

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

Leggi dbmail online: https://riptutorial.com/it/sql-server/topic/4908/dbmail

Capitolo 28: Delimitazione di caratteri speciali e parole riservate

Osservazioni

In generale, è meglio non usare le parole riservate T-SQL come nomi di tabelle, nomi di colonne, nomi di oggetti di programmazione, alias ecc. Quindi il metodo per sfuggire a queste parole chiave dovrebbe essere applicato solo se si eredita un progetto di database che non può essere modificato .

Per le parole riservate, l'utilizzo delle parentesi quadre non è obbligatorio. Quando si utilizza uno strumento come SQL Server Management Studio, le parole riservate verranno evidenziate per richiamare l'attenzione sul fatto che sono riservate.

Examples

Metodo di base

Il metodo di base per evitare le parole riservate per SQL Server è l'uso delle parentesi quadre (`[` e `]`). Ad esempio, *Descrizione* e *Nome* sono parole riservate; tuttavia, se esiste un oggetto che utilizza entrambi come nomi, la sintassi utilizzata è:

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

L'unico carattere speciale per SQL Server è la citazione singola `'` ed è sfuggito raddoppiando il suo utilizzo. Ad esempio, per trovare il nome *O'Shea* nella stessa tabella, si utilizzerà la seguente sintassi:

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

Leggi Delimitazione di caratteri speciali e parole riservate online: https://riptutorial.com/it/sql-server/topic/7156/delimitazione-di-caratteri-speciali-e-parole-riservate

Capitolo 29: Dichiarazione CASE

Osservazioni

Sopra l'esempio è solo per mostrare la sintassi per l'utilizzo di case statement in SQL Server con l'esempio del giorno della settimana. Sebbene lo stesso output possa essere ottenuto usando "SELECT DATENAME (WEEKDAY, GETDATE ())".

Examples

Semplice dichiarazione CASE

In una semplice dichiarazione di un caso, un valore o una variabile viene verificata rispetto a più risposte possibili. Il seguente codice è un esempio di una semplice dichiarazione di un caso:

```
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
```

Dichiarazione CASE ricercata

In un'istruzione Case causata, ciascuna opzione può testare uno o più valori in modo indipendente. Il seguente codice è un esempio di una dichiarazione di caso cercato:

```
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
```

Leggi Dichiarazione CASE online: https://riptutorial.com/it/sql-server/topic/7238/dichiarazione-case

Capitolo 30: Elimina la parola chiave

introduzione

La parola chiave Drop può essere utilizzata con vari oggetti SQL, questo argomento fornisce esempi rapidi di utilizzo diverso con gli oggetti del database.

Osservazioni

Collegamenti a MSDN.

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

Examples

Drop tables

Il comando **DROP TABLE** rimuove le definizioni di tabella e tutti i dati, gli indici, i trigger, i vincoli e le autorizzazioni correlate.

Prima di rilasciare una tabella, è necessario verificare se esistono oggetti (viste, stored procedure, altre tabelle) che fanno riferimento alla tabella.

Non è possibile eliminare una tabella referenziata da un'altra tabella da FOREIGN KEY. È necessario innanzitutto eliminare la chiave esterna facendone riferimento.

È possibile rilasciare una tabella a cui fa riferimento una vista o una stored procedure, ma dopo aver rilasciato la tabella, la vista o la stored procedure non è più utilizzabile.

La sintassi

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

- `IF EXISTS` : rilasciare la tabella solo se esiste
- `database_name` : specifica il nome del database in cui è contenuta la tabella
- `schema_name` : specificare il nome dello schema in cui si trova la tabella
- `table_name` : specifica il nome della tabella da eliminare

Esempi

Rimuovere la tabella con il nome **TABLE_1** dal database corrente e il dbo dello schema

predefinito

```
DROP TABLE Table_1;
```

Rimuovere la tabella con **TABLE_1** dal database **HR** e il dbo dello schema predefinito

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

Rimuovere la tabella con **TABLE_1** dal database **HR** e schema **esterno**

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

Drop Database

Il comando **DROP DATABASE** rimuove un catalogo di database, indipendentemente dal suo stato (non in linea, di sola lettura, sospetto, ecc.), Dall'istanza di SQL Server corrente.

Non è possibile eliminare un database se vi sono associati snapshot del database, poiché è necessario eliminare prima le istantanee del database.

Un drop di database rimuove tutti i file del disco fisico (a meno che non sia offline) utilizzati dal database a meno che non si utilizzi la stored procedure "sp_detach_db".

Una caduta di istantanee del database elimina lo snapshot dall'istanza di SQL Server ed elimina i file fisici utilizzati anche da esso.

Un database abbandonato può essere ricreato solo ripristinando un backup (non da uno snapshot del database).

La sintassi

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

- `IF EXISTS` : rilasciare la tabella solo se esiste
- `database_name` : specifica il nome del database da eliminare
- `database_snapshot_name` : specifica lo snapshot del database da rimuovere
-

Esempi

Rimuovere un singolo database;

```
DROP DATABASE Database1;
```

Rimozione di più database

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

Rimozione di un'istantanea

```
DROP DATABASE Database1_snapshot17;
```

Rimozione se il database esiste

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

Elimina tabelle temporanee

Nel server SQL abbiamo 2 tipi di tabelle temporanee:

1. `##GlobalTempTable` è un tipo di tabella temporanea che viene spostato tra tutte le sessioni dell'utente.
2. `#LocalTempTable` temporanea `#LocalTempTable` - è un tipo di tabella temporanea che esiste solo nell'ambito corrente (solo nel processo effettivo - puoi ottenere l'id del tuo processo corrente da `SELECT @@SPID`)

Il processo di rilascio delle tabelle temporanee è lo stesso della tabella normale:

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

PRIMA di SQL Server 2016:

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

SQL Server 2016:

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

Leggi Elimina la parola chiave online: https://riptutorial.com/it/sql-server/topic/9532/elimina-la-parola-chiave

Capitolo 31: Esportare i dati nel file txt usando SQLCMD

Sintassi

- sqlcmd -S SHERAZM-E7450 \ SQL2008R2 -d Baseline_DB_Aug_2016 -o c: \ employee.txt -Q "seleziona * dal dipendente"

Examples

Usando SQLCMD su Prompt dei comandi

La struttura dei comandi è

sqlcmd -S yourservername \ instancename -d database_name -o outputfilename_withpath -Q "la tua query di selezione"

Gli interruttori sono come segue

-S per nomeserver e nome di istanza

-d per il database di origine

-o per il file di output di destinazione (creerà il file di output)

-Q per la query per recuperare i dati

Leggi Esportare i dati nel file txt usando SQLCMD online: https://riptutorial.com/it/sql-server/topic/7076/esportare-i-dati-nel-file-txt-usando-sqlcmd

Capitolo 32: Espressioni di tabella comuni

Sintassi

- WITH *cte_name* [(*column_name_1* , *column_name_2* , ...)] AS (*cte_expression*)

Osservazioni

È necessario separare un CTE dall'istruzione precedente con un carattere di punto e virgola (;).

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

Lo scopo di una CTE è un singolo batch e solo a valle della sua definizione. Un lotto può contenere più CTE e un CTE può fare riferimento a un altro CTE definito in precedenza nel batch, ma un CTE potrebbe non fare riferimento a un altro CTE definito successivamente nel batch.

Examples

Gerarchia dei dipendenti

Impostazione tabella

```
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)
```

Espressione di tabella comune

```
;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
```

Produzione:

Nome e cognome	EmpLevel	del manager
Ken Sánchez	1	*nullo*
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

Trova l'ultimo stipendio più alto usando CTE

Tabella dei dipendenti:

```
|  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
```

Per trovare il 2o stipendio più alto sostituisci semplicemente N con 2. Analogamente, per trovare il 3o stipendio più alto, sostituisci semplicemente N con 3.

Elimina le righe duplicate usando CTE

Tabella dei dipendenti:

```
|  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
```

Risultato dell'esecuzione:

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

Genera una tabella di date usando 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)
```

Questo esempio restituisce una tabella di date a colonna singola, a partire dalla data specificata nella variabile @startdate e restituendo il valore successivo di @numberDays.

CTE ricorsivo

Questo esempio mostra come ottenere ogni anno da quest'anno al 2011 (2012 - 1).

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

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

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

myYear
2016
2015
2014
2013

myYear
2012
2011

È possibile controllare la ricorsione (si pensi all'overflow dello stack nel codice) con MAXRECURSION come opzione di query che limiterà il numero di chiamate ricorsive.

```
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);
```

> Messaggio 530, livello 16, stato 1, riga 2: istruzione terminata. La massima ricorsione 10 è stata esaurita prima del completamento dell'istruzione.

CTE con più istruzioni AS

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

Con le espressioni di tabella comuni, è possibile creare più query utilizzando istruzioni AS separate da virgole. Una query può quindi fare riferimento a una o tutte quelle query in molti modi diversi, anche unendole.

Leggi Espressioni di tabella comuni online: https://riptutorial.com/it/sql-server/topic/1343/espressioni-di-tabella-comuni

Capitolo 33: Filestream

introduzione

FILESTREAM integra il Motore di database di SQL Server con un file system NTFS memorizzando i dati varbinary (max) binary large object (BLOB) come file sul file system. Le istruzioni Transact-SQL possono inserire, aggiornare, interrogare, cercare e eseguire il backup dei dati FILESTREAM. Le interfacce del file system Win32 forniscono accesso in streaming ai dati.

Examples

Esempio

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

Leggi Filestream online: https://riptutorial.com/it/sql-server/topic/9509/filestream

Capitolo 34: Funzione Split Split in Sql Server

Examples

Dividere una stringa in SQL Server 2016

In **SQL Server 2016,** infine, è stata introdotta la funzione di stringa divisa: `STRING_SPLIT`

Parametri: accetta due parametri

Stringa :

> Espressione di qualsiasi tipo di carattere (es. Nvarchar, varchar, nchar o char).

separatore :

> Espressione di un singolo carattere di qualsiasi tipo di carattere (ad esempio nvarchar (1), varchar (1), nchar (1) o char (1)) utilizzato come separatore per stringhe concatenate.

Nota: dovresti sempre controllare se l'espressione è una stringa non vuota.

Esempio:

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

Nell'esempio sopra

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

Risultato:

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

Se è una stringa vuota:

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

Risultato:

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

Puoi evitare la situazione di cui sopra aggiungendo una clausola `WHERE`

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

Dividi la stringa in Sql Server 2008/2012/2014 usando XML

Poiché non esiste una funzione `STRING_SPLIT` , è necessario utilizzare XML hack per suddividere la stringa in righe:

Esempio:

```
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);
```

Risultato:

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

Tabella T-SQL variabile e 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
```

Leggi Funzione Split Split in Sql Server online: https://riptutorial.com/it/sql-server/topic/3713/funzione-split-split-in-sql-server

Capitolo 35: Funzioni aggregate

introduzione

Le funzioni di aggregazione in SQL Server eseguono calcoli su insiemi di valori, restituendo un singolo valore.

Sintassi

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

Examples

SOMMA()

Restituisce la somma dei valori numerici in una determinata colonna.

Abbiamo una tabella come mostrato in figura che verrà utilizzata per eseguire diverse funzioni di aggregazione. Il nome della tabella è *Marksheet* .

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

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

La funzione `sum` non considera le righe con valore NULL nel campo usato come parametro

Nell'esempio sopra se abbiamo un'altra riga come questa:

```
106    Italian    NULL
```

Questa riga non sarà presa in considerazione nel calcolo della somma

AVG ()

Restituisce la media dei valori numerici in una determinata colonna.

Abbiamo una tabella come mostrato in figura che verrà utilizzata per eseguire diverse funzioni di aggregazione. Il nome della tabella è *Marksheet* .

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

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

La funzione `average` non considera le righe con valore NULL nel campo utilizzato come parametro

Nell'esempio sopra se abbiamo un'altra riga come questa:

```
106    Italian    NULL
```

Questa riga non sarà considerata nel calcolo medio

MAX ()

Restituisce il valore più grande in una determinata colonna.

Abbiamo una tabella come mostrato in figura che verrà utilizzata per eseguire diverse funzioni di aggregazione. Il nome della tabella è *Marksheet* .

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

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

MIN ()

Restituisce il valore più piccolo in una determinata colonna.

Abbiamo una tabella come mostrato in figura che verrà utilizzata per eseguire diverse funzioni di aggregazione. Il nome della tabella è *Marksheet* .

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

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

CONTARE()

Restituisce il numero totale di valori in una data colonna.

Abbiamo una tabella come mostrato in figura che verrà utilizzata per eseguire diverse funzioni di aggregazione. Il nome della tabella è *Marksheet* .

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

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

La funzione `count` non considera le righe con valore NULL nel campo usato come parametro. Di solito il parametro count è * (tutti i campi) quindi solo se tutti i campi della riga sono NULL questa riga non sarà considerata

Nell'esempio sopra se abbiamo un'altra riga come questa:

```
106    Italian    NULL
```

Questa riga non verrà presa in considerazione nel calcolo del conteggio

NOTA

La funzione `COUNT(*)` restituisce il numero di righe in una tabella. Questo valore può anche essere ottenuto utilizzando un'espressione costante non nulla che non contiene riferimenti di colonna, ad esempio `COUNT(1)` .

Esempio

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

COUNT (Column_Name) con GROUP BY Column_Name

La maggior parte delle volte ci piace ottenere il numero totale di occorrenza di un valore di

colonna in una tabella, ad esempio:

NOME DELLA TABELLA: RAPPORTI

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

Segnala il nome	CONTARE
Test	4
Test 2	1
Test 3	2
Test 4	1

Leggi Funzioni aggregate online: https://riptutorial.com/it/sql-server/topic/5802/funzioni-aggregate

Capitolo 36: Funzioni della finestra

Examples

Media mobile centrata

Calcola una media mobile a 6 mesi (126 giorni lavorativi) centrata di un prezzo:

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

Tieni presente che, poiché saranno necessarie *fino a* 63 righe prima e dopo ogni riga restituita, all'inizio e alla fine dell'intervallo TradeDate non sarà centrato: quando raggiunge il TradeDate più grande sarà in grado di trovare solo 63 valori precedenti includere nella media.

Trova l'elemento più recente in un elenco di eventi con data e ora

Nelle tabelle che registrano eventi c'è spesso un campo datetime che registra l'ora in cui si è verificato un evento. Trovare il singolo evento più recente può essere difficile perché è sempre possibile che due eventi siano stati registrati con timestamp esattamente identici. Puoi usare row_number () over (order by ...) per assicurarti che tutti i record siano classificati in modo univoco e selezionare quello superiore (dove my_ranking = 1)

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

Questa stessa tecnica può essere utilizzata per restituire una singola riga da qualsiasi set di dati con valori potenzialmente duplicati.

Media mobile degli ultimi 30 articoli

Media mobile degli ultimi 30 articoli venduti

```
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
```

Leggi Funzioni della finestra online: https://riptutorial.com/it/sql-server/topic/3209/funzioni-della-finestra

Capitolo 37: Funzioni di classifica

Sintassi

- DENSE_RANK () OVER ([<partition_by_clause>] <order_by_clause>)
- RANK () OVER ([partition_by_clause] order_by_clause)

Parametri

argomenti	Dettagli
`<partition_by_clause>`	Divide il set di risultati prodotto dalla clausola FROM in partizioni a cui viene applicata la funzione `DENSE_RANK` . Per la sintassi `PARTITION BY` , vedere la clausola OVER (Transact-SQL) .
`<order_by_clause>`	Determina l'ordine in cui la funzione `DENSE_RANK` viene applicata alle righe in una partizione.
`OVER ( [ partition_by_clause ] order_by_clause)`	`partition_by_clause` divide il set di risultati prodotto dalla clausola `FROM` in partizioni a cui viene applicata la funzione. Se non specificato, la funzione considera tutte le righe del set di risultati della query come un singolo gruppo. order_by_clause determina l'ordine dei dati prima che la funzione venga applicata. È richiesto order_by_clause. La `<rows or range clause>` delle `<rows or range clause>` della clausola `OVER` non può essere specificata per la funzione `RANK` . Per ulteriori informazioni, vedere Clausola OVER (Transact-SQL) .

Osservazioni

Se due o più righe si uniscono per un grado nella stessa partizione, ogni fila legata riceve lo stesso valore. Ad esempio, se i due migliori venditori hanno lo stesso valore SalesYTD, sono entrambi al primo posto. Il venditore con la successiva SalesYTD successiva è al secondo posto. Questo è uno in più del numero di righe distinte che precedono questa riga. Pertanto, i numeri restituiti dalla funzione `DENSE_RANK` non hanno spazi vuoti e hanno sempre ranghi consecutivi.

L'ordinamento utilizzato per l'intera query determina l'ordine in cui le righe vengono visualizzate in un risultato. Ciò implica che una riga classificata come numero uno non deve essere la prima riga nella partizione.

`DENSE_RANK` non è deterministico. Per ulteriori informazioni, vedere Funzioni deterministiche e non deterministiche .

Examples

RANGO()

A RANK () Restituisce il rango di ogni riga nel set di risultati della colonna partizionata.

Per esempio :

```
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 ()

Come quello di RANK (). Restituisce il rango senza spazi vuoti:

```
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
```

Leggi Funzioni di classifica online: https://riptutorial.com/it/sql-server/topic/5031/funzioni-di-classifica

Capitolo 38: Funzioni di stringa

Osservazioni

Elenco delle funzioni stringa (in ordine alfabetico):

- Ascii
- carbonizzare
- charIndex
- concat
- Differenza
- Formato
- Sinistra
- Len
- Inferiore
- ltrim
- nchar
- Patindex
- QUOTENAME
- Sostituire
- Replicare
- Inverso
- Destra
- rtrim
- Soundex
- Spazio
- Str
- String_escape

- String_split
- Cose
- substring
- Unicode
- Superiore

Examples

Sinistra

Restituisce una sottostringa che inizia con il carattere più a sinistra di una stringa e fino alla lunghezza massima specificata.

parametri:

1. espressione del personaggio. L'espressione di carattere può essere di qualsiasi tipo di dati che può essere convertita implicitamente in `varchar` o `nvarchar` , ad eccezione di `text` o `ntext`
2. lunghezza massima. Un numero intero compreso tra 0 e `bigint` max `bigint` (9.223.372.036.854.775.807).
   Se il parametro lunghezza massima è negativo, verrà generato un errore.

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

Se la lunghezza massima è maggiore del numero di caratteri nella stringa, viene restituita la stringa dell'intero.

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

Destra

Restituisce una sottostringa che rappresenta la parte più corretta della stringa, con la lunghezza massima specificata.

parametri:

1. espressione del personaggio. L'espressione di carattere può essere di qualsiasi tipo di dati che può essere convertita implicitamente in `varchar` o `nvarchar` , ad eccezione di `text` o `ntext`
2. lunghezza massima. Un numero intero compreso tra 0 e `bigint` max `bigint` (9.223.372.036.854.775.807). Se il parametro lunghezza massima è negativo, verrà generato un errore.

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

Se la lunghezza massima è maggiore del numero di caratteri nella stringa, viene restituita la

stringa dell'intero.

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

substring

Restituisce una sottostringa che inizia con il carattere che si trova nell'indice iniziale specificato e nella lunghezza massima specificata.

parametri:

1. Espressione di carattere L'espressione di carattere può essere di qualsiasi tipo di dati che può essere convertita implicitamente in `varchar` o `nvarchar` , ad eccezione di `text` o `ntext` .
2. Inizia indice. Un numero (`int` o `bigint`) che specifica l'indice iniziale della sottostringa richiesta. (**Nota: le** stringhe nel server SQL sono indice di base 1, il che significa che il primo carattere della stringa è l'indice 1). Questo numero può essere inferiore a 1. In questo caso, se la somma dell'indice iniziale e della lunghezza massima è maggiore di 0, la stringa di ritorno sarà una stringa che inizia dal primo carattere dell'espressione di carattere e con la lunghezza di (indice iniziale + lunghezza massima - 1). Se è inferiore a 0, verrà restituita una stringa vuota.
3. Lunghezza massima. Un numero intero compreso tra 0 e `bigint` max `bigint` (9.223.372.036.854.775.807). Se il parametro lunghezza massima è negativo, verrà generato un errore.

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

Se la lunghezza massima + indice iniziale è maggiore del numero di caratteri nella stringa, viene restituita la stringa dell'intero.

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

Se l'indice iniziale è più grande del numero di caratteri nella stringa, viene restituita una stringa vuota.

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

ASCII

Restituisce un valore int che rappresenta il codice ASCII del carattere più a sinistra di una stringa.

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

Se la stringa è Unicode e il carattere più a sinistra non è ASCII ma è rappresentabile nelle regole di confronto correnti, è possibile restituire un valore superiore a 127:

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

Se la stringa è Unicode e il carattere più a sinistra non può essere rappresentato nelle regole di confronto correnti, viene restituito il valore int di 63: (che rappresenta il punto interrogativo in ASCII):

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

charIndex

Restituisce l'indice iniziale di una prima occorrenza di un'espressione stringa all'interno di un'altra espressione di stringa.

Lista dei parametri:

1. Stringa per trovare (fino a 8000 caratteri)
2. Stringa da cercare (qualsiasi tipo di dati di carattere valido e lunghezza, incluso binario)
3. (Opzionale) indice per iniziare. Un numero di tipo int o big int. Se omesso o meno di 1, la ricerca inizia all'inizio della stringa.

Se la stringa da cercare è `varchar(max)` , `nvarchar(max)` o `varbinary(max)` , la funzione `CHARINDEX` restituirà un valore `bigint` . Altrimenti, restituirà un `int` .

```
SELECT CHARINDEX('is', 'this is my string') -- returns 3
SELECT CHARINDEX('is', 'this is my string', 4) -- returns 6
SELECT CHARINDEX(' is', 'this is my string') -- returns 5
```

carbonizzare

Restituisce un carattere rappresentato da un codice ASCII int.

```
SELECT CHAR(116) -- Returns 't'
SELECT CHAR(84)  -- Returns 'T'
```

Questo può essere usato per introdurre nuovi line / line feed `CHAR(10)` , carriage return `CHAR(13)` , ecc. Vedi AsciiTable.com come riferimento.

Se il valore dell'argomento non è compreso tra 0 e 255, la funzione CHAR restituisce `NULL` .
Il tipo di dati di ritorno della funzione `CHAR` è `char(1)`

Len

Restituisce il numero di caratteri di una stringa.
Nota: la funzione `LEN` ignora gli spazi finali:

```
SELECT LEN('My string'), -- returns 9
```

```
       LEN('My string   '), -- returns 9
       LEN('   My string') -- returns 12
```

Se la lunghezza, compresi gli spazi finali, è desiderata, ci sono diverse tecniche per raggiungere questo obiettivo, anche se ognuno ha i suoi svantaggi. Una tecnica è quella di aggiungere un singolo carattere alla stringa, e quindi utilizzare il `LEN` meno uno:

```
DECLARE @str varchar(100) = 'My string   '
SELECT LEN(@str + 'x') - 1 -- returns 12
```

Lo svantaggio è se il tipo della variabile o colonna stringa è della lunghezza massima, l'append del carattere extra viene scartata e la lunghezza risultante non conterà ancora gli spazi finali. Per risolvere questo problema, la seguente versione modificata risolve il problema e fornisce i risultati corretti in tutti i casi a scapito di una piccola quantità di tempo di esecuzione aggiuntivo, e per questo (risultati corretti, anche con coppie surrogate e una ragionevole velocità di esecuzione) sembra essere la migliore tecnica da usare:

```
SELECT LEN(CONVERT(NVARCHAR(MAX), @str) + 'x') - 1
```

Un'altra tecnica consiste nell'utilizzare la funzione `DATALENGTH` .

```
DECLARE @str varchar(100) = 'My string   '
SELECT DATALENGTH(@str) -- returns 12
```

È importante notare che `DATALENGTH` restituisce la lunghezza in byte della stringa in memoria. Questo sarà diverso per `varchar` vs `nvarchar` .

```
DECLARE @str nvarchar(100) = 'My string   '
SELECT DATALENGTH(@str) -- returns 24
```

È possibile regolare per questo dividendo la lunghezza dei dati della stringa per la lunghezza dei dati di un singolo carattere (che deve essere dello stesso tipo). L'esempio seguente esegue questa operazione e gestisce anche il caso in cui la stringa di destinazione risulta vuota, evitando così una divisione per zero.

```
DECLARE @str nvarchar(100) = 'My string   '
SELECT DATALENGTH(@str) / DATALENGTH(LEFT(LEFT(@str, 1) + 'x', 1)) -- returns 12
```

Anche questo, tuttavia, ha un problema in SQL Server 2012 e versioni successive. Produrrà risultati errati quando la stringa contiene coppie surrogate (alcuni caratteri possono occupare più byte di altri caratteri nella stessa stringa).

Un'altra tecnica consiste nell'usare `REPLACE` per convertire gli spazi in un carattere non spaziale e prendere la `LEN` del risultato. Ciò fornisce risultati corretti in tutti i casi, ma ha una velocità di esecuzione molto scarsa con stringhe lunghe.

concat

SQL Server 2012

Restituisce una stringa che è il risultato di due o più stringhe unite insieme. `CONCAT` accetta due o più argomenti.

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

Nota: Diversamente dalla concatenazione di stringhe usando l'operatore di concatenazione di stringhe (`+`), quando si passa un valore nullo alla funzione `concat` lo si converte implicitamente in una stringa vuota:

```
SELECT CONCAT('This', NULL, ' is', ' my', ' string'), -- returns 'This is my string'
       'This' + NULL + ' is' + ' my' + ' string' -- returns NULL.
```

Anche gli argomenti di un tipo non stringa verranno convertiti implicitamente in una stringa:

```
SELECT CONCAT('This', ' is my ', 3, 'rd string') -- returns 'This is my 3rd string'
```

Anche le variabili di tipo non stringa verranno convertite in formato stringa, non è necessario copiarle manualmente o trasmetterle alla stringa:

```
DECLARE @Age INT=23;
SELECT CONCAT('Ram is ', @Age,' years old');  -- returns 'Ram is 23 years old'
```

SQL Server 2012

Le versioni precedenti non supportano la funzione `CONCAT` e devono invece utilizzare l'operatore di concatenazione di stringhe (`+`). I tipi non stringa devono essere espressi o convertiti in tipi di stringa per concatenarli in questo modo.

```
SELECT 'This is the number ' + CAST(42 AS VARCHAR(5)) --returns 'This is the number 42'
```

Inferiore

Restituisce un'espressione di carattere (`varchar` o `nvarchar`) dopo aver convertito tutti i caratteri maiuscoli in caratteri minuscoli.

parametri:

1. Espressione di carattere Qualsiasi espressione di carattere o dati binari che possa essere convertita implicitamente in `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'
```

Superiore

Restituisce un'espressione di carattere (`varchar` o `nvarchar`) dopo aver convertito tutti i caratteri minuscoli in maiuscoli.

parametri:

1. Espressione di carattere Qualsiasi espressione di carattere o dati binari che possa essere convertita implicitamente in `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

Restituisce un'espressione di carattere (`varchar` o `nvarchar`) dopo aver rimosso tutti gli spazi bianchi iniziali, cioè gli spazi bianchi da sinistra fino al primo carattere di spazio non bianco.

parametri:

1. espressione del personaggio. Qualsiasi espressione di carattere o dati binari che può essere convertita implicitamente in `varcher` , ad eccezione di `text` , `ntext` e `image` .

```
SELECT LTRIM('    This is my string') -- Returns 'This is my string'
```

RTrim

Restituisce un'espressione di carattere (`varchar` o `nvarchar`) dopo aver rimosso tutti gli spazi bianchi finali, ovvero gli spazi dall'estremità destra della stringa fino al primo carattere spazio non vuoto a sinistra.

parametri:

1. espressione del personaggio. Qualsiasi espressione di carattere o dati binari che può essere convertita implicitamente in `varcher` , ad eccezione di `text` , `ntext` e `image` .

```
SELECT RTRIM('This is my string     ') -- Returns 'This is my string'
```

Unicode

Restituisce il valore intero che rappresenta il valore Unicode del primo carattere dell'espressione di input.

parametri:

1. Espressione di caratteri Unicode. Qualsiasi espressione valida `nchar` o `nvarchar` .

```
SELECT UNICODE(N'Ɛ') -- Returns 400
```

```
DECLARE @Unicode nvarchar(11) = N'Ɛ is a char'
SELECT UNICODE(@Unicode) -- Returns 400
```

nchar

Restituisce i caratteri Unicode (`nchar(1)` o `nvarchar(2)`) corrispondenti all'argomento intero che riceve, come definito dallo standard Unicode.

parametri:

1. espressione intera. Qualsiasi espressione intera che è un numero positivo compreso tra 0 e 65535, o se le regole di confronto del database supportano il flag di carattere supplementare (CS), l'intervallo supportato è compreso tra 0 e 1114111. Se l'espressione intera non rientra in questo intervallo, `null` è restituito.

```
SELECT NCHAR(257) -- Returns 'ā'
SELECT NCHAR(400) -- Returns 'Ɛ'
```

Inverso

Restituisce un valore stringa in ordine inverso.

parametri:

1. espressione di stringa. Qualsiasi stringa o dati binari che possono essere convertiti implicitamente in `varchar` .

```
Select REVERSE('Sql Server') -- Returns 'revreS lqS'
```

PATINDEX

Restituisce la posizione iniziale della prima occorrenza di un modello specificato nell'espressione specificata.

parametri:

1. modello. Un'espressione di carattere contiene la sequenza da trovare. Limitato a una lunghezza massima di 8000 caratteri. I caratteri jolly (`%` , _) possono essere utilizzati nel modello. Se il modello non inizia con un carattere jolly, può corrispondere solo a ciò che si trova all'inizio dell'espressione. Se non termina con un carattere jolly, può corrispondere solo a ciò che si trova alla fine dell'espressione.

2. espressione. Qualsiasi tipo di dati stringa.

```
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.
```

Spazio

Restituisce una stringa (`varchar`) di spazi ripetuti.

parametri:

1. espressione intera. Qualsiasi espressione intera, fino a 8000. Se negativo, viene restituito un valore `null` . se 0, viene restituita una stringa vuota. (Per restituire una stringa più lunga di 8000 spazi, utilizzare Replica.

```
SELECT SPACE(-1) -- Returns NULL
SELECT SPACE(0)  -- Returns an empty string
SELECT SPACE(3)  -- Returns '   ' (a string containing 3 spaces)
```

Replicare

Ripete un valore di stringa un numero specificato di volte.

parametri:

1. espressione di stringa. L'espressione di stringa può essere una stringa di caratteri o dati binari.
2. espressione intera. Qualsiasi tipo intero, incluso `bigint` . Se negativo, viene restituito un valore `null` . Se 0, viene restituita una stringa vuota.

```
SELECT REPLICATE('a', -1)  -- Returns NULL

SELECT REPLICATE('a', 0)  -- Returns ''

SELECT REPLICATE('a', 5)  -- Returns 'aaaaa'

SELECT REPLICATE('Abc', 3) -- Returns 'AbcAbcAbc'
```

Nota: se l'espressione stringa non è di tipo `varchar(max)` o `nvarchar(max)` , il valore restituito non supererà 8000 caratteri. La replica si interrompe prima di aggiungere la stringa che farà sì che il valore restituito superi tale limite:

```
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
```

Sostituire

Restituisce una stringa (`varchar` o `nvarchar`) in cui tutte le occorrenze di una sottostringa specificata vengono sostituite con un'altra sottostringa.

parametri:

1. espressione di stringa. Questa è la stringa che verrebbe cercata. Può essere un carattere o un tipo di dati binari.
2. modello. Questa è la stringa secondaria che verrebbe sostituita. Può essere un carattere o un tipo di dati binari. L'argomento pattern non può essere una stringa vuota.
3. sostituzione. Questa è la stringa secondaria che sostituisce la stringa secondaria del modello. Può essere un carattere o dati binari.

```
SELECT REPLACE('This is my string', 'is', 'XX') -- Returns 'ThXX XX my string'.
```

Gli appunti:

- Se l'espressione stringa non è di tipo `varchar(max)` o `nvarchar(max)` , la funzione `replace` tronca il valore restituito a 8.000 caratteri.
- Il tipo di dati restituiti dipende dai tipi di dati di input: restituisce `nvarchar` se uno dei valori di input è `nvarchar` o altrimenti `varchar` .
- Restituisce `NULL` se uno qualsiasi dei parametri di input è `NULL`

String_Split

SQL Server 2016

Divide un'espressione di stringa usando un separatore di caratteri. Nota che `STRING_SPLIT()` è una funzione valutata a livello di tabella e pertanto deve essere utilizzata all'interno della clausola `FROM` .

parametri:

1. stringa. Qualsiasi espressione di tipo di carattere (`char` , `nchar` , `varchar` o `nvarchar`)
2. separatore. Un'espressione a carattere singolo di qualsiasi tipo (`char(1)` , `nchar(1)` , `varchar(1)` o `nvarchar(1)`).

Restituisce una singola tabella di colonne in cui ogni riga contiene un frammento della stringa. Il nome delle colonne è `value` e il tipo di dati è `nvarchar` se uno qualsiasi dei parametri è `nchar` o `nvarchar` , altrimenti `varchar` .

L'esempio seguente divide una stringa usando lo spazio come separatore:

```
SELECT value FROM STRING_SPLIT('Lorem ipsum dolor sit amet.', ' ');
```

Risultato:

```
value
-----
Lorem
ipsum
dolor
sit
amet.
```

Osservazioni:

> La funzione `STRING_SPLIT` è disponibile solo con il livello di compatibilità **130** . Se il livello di compatibilità del database è inferiore a 130, SQL Server non sarà in grado di trovare ed eseguire la funzione `STRING_SPLIT` . È possibile modificare il livello di compatibilità di un database utilizzando il seguente comando:

```
ALTER DATABASE [database_name] SET COMPATIBILITY_LEVEL = 130
```

SQL Server 2016

Le versioni precedenti di SQL Server non dispongono di una funzione di stringa divisa integrata. Esistono molte funzioni definite dall'utente che gestiscono il problema della divisione di una stringa. Puoi leggere l'articolo di Aaron Bertrand Split stringhe nel modo giusto - o il modo migliore per un confronto completo di alcuni di essi.

Str

Restituisce i dati dei caratteri (`varchar`) convertiti da dati numerici.

parametri:

1. espressione fluttuante. Un tipo di dati numerico approssimativo con un punto decimale.
2. lunghezza. **opzionale.** La lunghezza totale dell'espressione stringa che verrebbe restituita, comprese le cifre, il punto decimale e gli spazi iniziali (se necessario). Il valore predefinito è 10.
3. decimale. **opzionale.** Il numero di cifre a destra del punto decimale. Se superiore a 16, il risultato verrebbe troncato a sedici posti a destra del separatore decimale.

```
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

Restituisce una stringa Unicode circondata da delimitatori per renderla un identificatore delimitato di SQL Server valido.

parametri:

1. stringa di caratteri. Una stringa di dati Unicode, fino a 128 caratteri (`sysname`). Se una stringa

di input è più lunga di 128 caratteri, la funzione restituisce `null` .
2. citare il carattere. **Opzionale** . Un singolo carattere da utilizzare come delimitatore. Può essere una virgoletta singola (`'` o ``), una parentesi sinistra o destra (`{` , `[` , `(` , `<` o `>` , `)` , `]` , `}`) o una virgoletta doppia (`"`). Qualsiasi altro valore restituirà null Il valore predefinito è parentesi quadre.

```
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

Restituisce un codice di quattro caratteri (`varchar`) per valutare la somiglianza fonetica di due stringhe.

parametri:

1. espressione del personaggio. Un'espressione alfanumerica di dati di carattere.

La funzione soundex crea un codice di quattro caratteri basato su come l'espressione del personaggio suonerebbe quando pronunciata. il primo carattere è la versione maiuscola del primo carattere del parametro, il resto 3 caratteri sono numeri che rappresentano le lettere nell'espressione (tranne a, e, i, o, u, h, w e y che vengono ignorati) .

```
SELECT SOUNDEX ('Smith') -- Returns 'S530'

SELECT SOUNDEX ('Smythe') -- Returns 'S530'
```

Differenza

Restituisce un valore intero (`int`) che indica la differenza tra i valori soundex di due espressioni di carattere.

parametri:

1. espressione del personaggio 1.

2. espressione del personaggio 2.

Entrambi i parametri sono espressioni alfanumeriche di dati di carattere.

Il numero intero restituito è il numero di caratteri nei valori soundex dei parametri che sono gli stessi, quindi 4 significa che le espressioni sono molto simili e 0 significa che sono molto diverse.

```
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
```

Formato

SQL Server 2012

Restituisce un valore `NVARCHAR` formattato con il formato e la cultura specificati (se specificato). Viene principalmente utilizzato per convertire i tipi di data e ora in stringhe.

parametri:

1. `value` . Un'espressione di un tipo di dati supportato da formattare. i tipi validi sono elencati di seguito.
2. `format` . Un modello di formato `NVARCHAR` . Consulta la documentazione ufficiale Microsoft per stringhe di formato standard e personalizzate .
3. `culture` . **Opzionale** . argomento `nvarchar` che specifica una cultura. Il valore predefinito è la cultura della sessione corrente.

DATA

Usando le stringhe di formato standard:

```
DECLARE @d DATETIME = '2016-07-31';

SELECT
    FORMAT ( @d, 'd', 'en-US' ) AS 'US English Result' -- Returns '7/31/2016'
   ,FORMAT ( @d, 'd', 'en-gb' ) AS 'Great Britain English Result' -- Returns '31/07/2016'
   ,FORMAT ( @d, 'd', 'de-de' ) AS 'German Result' -- Returns '31.07.2016'
   ,FORMAT ( @d, 'd', 'zh-cn' ) AS 'Simplified Chinese (PRC) Result' -- Returns '2016/7/31'
   ,FORMAT ( @d, 'D', 'en-US' ) AS 'US English Result' -- Returns 'Sunday, July 31, 2016'
   ,FORMAT ( @d, 'D', 'en-gb' ) AS 'Great Britain English Result' -- Returns '31 July 2016'
   ,FORMAT ( @d, 'D', 'de-de' ) AS 'German Result' -- Returns 'Sonntag, 31. Juli 2016'
```

Utilizzando stringhe di formato personalizzate:

```
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` può essere utilizzato anche per la formattazione di `CURRENCY` , `PERCENTAGE` e `NUMBERS` .

MONETA

```
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 €'
```

Possiamo specificare il numero di cifre dopo il decimale.

```
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'
```

PERCENTUALE

```
   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 %'
```

NUMERO

```
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'
```

Elenco dei tipi di valori validi: (fonte)

```
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
```

Note importanti:

- `FORMAT` restituisce `NULL` per errori diversi da una cultura che non è valida. Ad esempio, `NULL` viene restituito se il valore specificato nel formato non è valido.
- `FORMAT` si basa sulla presenza di .NET Framework Common Language Runtime (CLR).
- `FORMAT` si basa sulle regole di formattazione CLR che impongono di eseguire il escape di due punti e punti. Pertanto, quando la stringa di formato (secondo parametro) contiene due punti o punti, i due punti o il periodo devono essere preceduti da una barra rovesciata quando un valore di input (primo parametro) è del tipo di dati temporali.

Vedi anche Formattazione di data e ora usando l' esempio di documentazione di FORMAT .

String_escape

SQL Server 2016

Evita caratteri speciali nei testi e restituisce il testo (`nvarchar(max)`) con caratteri di escape.

parametri:

1. testo. è un'espressione `nvarchar` che rappresenta la stringa che deve essere sfuggita.
2. genere. Regole di escape che verranno applicate. Attualmente l'unico valore supportato è `'json'`.

```
SELECT STRING_ESCAPE('\   /
\\    "     ', 'json') -- returns '\\\t\/\n\\\\\t\"\t'
```

Elenco di caratteri che saranno sfuggiti:

```
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
```

Leggi Funzioni di stringa online: https://riptutorial.com/it/sql-server/topic/4113/funzioni-di-stringa

Capitolo 39: Funzioni logiche

Examples

SCEGLIERE

SQL Server 2012

Restituisce l'elemento all'indice specificato da un elenco di valori. Se l' `index` supera i limiti dei `values` viene restituito `NULL` .

parametri:

1. `index` : numero intero, indice per oggetto in `values` . 1-based.
2. `values` : qualsiasi tipo, elenco separato da virgola

```
SELECT CHOOSE (1, 'apples', 'pears', 'oranges', 'bananas') AS chosen_result

chosen_result
-------------
apples
```

IIF

SQL Server 2012

Restituisce uno dei due valori, a seconda se una determinata espressione booleana restituisce true o false.

parametri:

1. `boolean_expression` valutato per determinare quale valore restituire
2. `true_value` restituito se `boolean_expression` restituisce true
3. `false_value` restituito se `boolean_expression` è falsa

```
SELECT IIF (42 > 23, 'I knew that!', 'That is not true.') AS iif_result

iif_result
------------
I knew that!
```

SQL Server 2012

`IIF` può essere sostituito da una dichiarazione `CASE` . L'esempio sopra è scritto come

```
SELECT CASE WHEN 42 > 23 THEN 'I knew that!' ELSE 'That is not true.' END AS iif_result

iif_result
------------
```

```
I knew that!
```

Leggi Funzioni logiche online: https://riptutorial.com/it/sql-server/topic/10647/funzioni-logiche

Capitolo 40: Generare un intervallo di date

Parametri

Parametro	Dettagli
@FromDate	Il limite inferiore compreso dell'intervallo di date generato.
@Ad oggi	Il limite superiore compreso dell'intervallo di date generato.

Osservazioni

La maggior parte degli esperti consiglia di creare una tabella delle date invece di generare una sequenza al volo. Vedi http://dba.stackexchange.com/questions/86435/filling-in-date-holes-in-grouped-by-date-sql-data

Examples

Generazione dell'intervallo di date con CTE ricorsivo

Usando un CTE ricorsivo, puoi generare un intervallo di date inclusivo:

```
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)
```

L'impostazione `MaxRecursion` predefinita è 100. Generando più di 100 date utilizzando questo metodo sarà necessario il segmento `Option (MaxRecursion N)` della query, dove `N` è l'impostazione `MaxRecursion` desiderata. Impostando questo a `0` si rimuoverà del tutto la limitazione `MaxRecursion` .

Generazione di un intervallo di date con una tabella di conteggio

Un altro modo per generare un intervallo di date consiste nell'utilizzare una tabella di riscontro per creare le date tra l'intervallo:

```
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
```

Leggi Generare un intervallo di date online: https://riptutorial.com/it/sql-server/topic/3232/generare-un-intervallo-di-date

Capitolo 41: Gestione del database SQL di Azure

Examples

Trova informazioni sul livello di servizio per il database SQL di Azure

Il database SQL di Azure ha diverse edizioni e livelli di prestazioni.

È possibile trovare la versione, l'edizione (di base, standard o premium) e l'obiettivo di servizio (S0, S1, P4, P11, ecc.) Del database SQL eseguito come servizio in Azure utilizzando le seguenti istruzioni:

```
select @@version
SELECT DATABASEPROPERTYEX('Wwi', 'EDITION')
SELECT DATABASEPROPERTYEX('Wwi', 'ServiceObjective')
```

Cambia il livello di servizio del database SQL di Azure

È possibile scalare o ridimensionare il database SQL di Azure utilizzando l'istruzione ALTER DATABASE:

```
ALTER DATABASE WWI
MODIFY (SERVICE_OBJECTIVE = 'P6')
-- or
ALTER DATABASE CURRENT
MODIFY (SERVICE_OBJECTIVE = 'P2')
```

Se si tenta di cambiare il livello di servizio mentre è in corso la modifica del livello di servizio del database corrente, si otterrà il seguente errore:

> Messaggio 40802, livello 16, stato 1, riga 1 Un'assegnazione dell'obiettivo di servizio sul server "....." e il database "....." sono già in corso. Attendi fino a quando lo stato di assegnazione dell'obiettivo di servizio per il database è contrassegnato come "Completato".

Rieseguire la dichiarazione ALTER DATABASE al termine del periodo di transizione.

Replica del database SQL di Azure

È possibile creare una replica secondaria del database con lo stesso nome su un altro server SQL di Azure, rendendo primario il database locale e iniziando la replica asincrona dei dati dal primario al nuovo secondario.

```
ALTER DATABASE <<mydb>>
ADD SECONDARY ON SERVER <<secondaryserver>>
```

```
WITH ( ALLOW_CONNECTIONS = ALL )
```

Il server di destinazione potrebbe trovarsi in un altro data center (utilizzabile per la replica geografica). Se esiste già un database con lo stesso nome sul server di destinazione, il comando avrà esito negativo. Il comando viene eseguito sul database master sul server che ospita il database locale che diventerà il primario. Quando ALLOW_CONNECTIONS è impostato su ALL (è impostato su NO per impostazione predefinita), la replica secondaria sarà un database di sola lettura che consentirà a tutti gli accessi con le autorizzazioni appropriate di connettersi.

La replica del database secondario potrebbe essere promossa a primary usando il seguente comando:

```
ALTER DATABASE mydb FAILOVER
```

È possibile rimuovere il database secondario sul server secondario:

```
ALTER DATABASE <<mydb>>
REMOVE SECONDARY ON SERVER <<testsecondaryserver>>
```

Creare il database SQL di Azure nel pool elastico

È possibile inserire il proprio database SQL in SQL pool elastico:

```
CREATE DATABASE wwi
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

È possibile creare una copia di un database esistente e inserirla in un pool elastico:

```
CREATE DATABASE wwi
AS COPY OF myserver.WideWorldImporters
( SERVICE_OBJECTIVE = ELASTIC_POOL ( name = mypool1 ) )
```

Leggi Gestione del database SQL di Azure online: https://riptutorial.com/it/sql-server/topic/7113/gestione-del-database-sql-di-azure

Capitolo 42: Gestione delle transazioni

Parametri

Parametro	Dettagli
transaction_name	per nominare la transazione - utile con il parametro [*con segno*] che consentirà una registrazione significativa - sensibile al maiuscolo / minuscolo (!)
con il segno ['descrizione']	può essere aggiunto a [*nome_operazione*] e memorizzerà un segno nel registro

Examples

scheletro di base della transazione con gestione degli errori

```
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
```

Leggi Gestione delle transazioni online: https://riptutorial.com/it/sql-server/topic/5859/gestione-delle-transazioni

Capitolo 43: grilletto

introduzione

Un trigger è un tipo speciale di stored procedure, che viene eseguito automaticamente dopo che si è verificato un evento. Esistono due tipi di trigger: Trigger linguaggio definizione dati e trigger linguaggio di manipolazione dati.

Di solito è legato a un tavolo e si attiva automaticamente. Non è possibile chiamare esplicitamente alcun trigger.

Examples

Tipi e classificazioni di Trigger

In SQL Server esistono due categorie di trigger: Trigger DDL e Trigger DML.

I trigger DDL vengono attivati in risposta agli eventi DDL (Data Definition Language). Questi eventi corrispondono principalmente alle istruzioni Transact-SQL che iniziano con le parole chiave `CREATE` , `ALTER` e `DROP` .

I trigger DML vengono attivati in risposta agli eventi DML (Data Manipulation Language). Questi eventi corrispondono alle istruzioni Transact-SQL che iniziano con le parole chiave `INSERT` , `UPDATE` e `DELETE` .

I trigger DML sono classificati in due tipi principali:

1. After Triggers (per trigger)

    - AFTER INSERT Trigger.
    - AFTER UPDATE Trigger.
    - AFTER DELETE Trigger.

2. Invece di trigger

    - INVIO DI INSERIMENTO Trigger.
    - INVECE DI AGGIORNAMENTO Trigger.
    - INVECE DI CANCELLARE Trigger.

Trigger DML

I trigger DML sono attivati come risposta alle istruzioni dml (`insert` , `update` o `delete`).
È possibile creare un trigger dml per indirizzare uno o più eventi dml per una singola tabella o vista. Ciò significa che un singolo trigger dml può gestire l'inserimento, l'aggiornamento e l'eliminazione di record da una tabella o vista specifica, ma può solo gestire i dati modificati su quella singola tabella o vista.

Trigger DML fornisce l'accesso alle tabelle `inserted` ed `deleted` che contengono informazioni sui dati che sono stati / saranno interessati dall'istruzione di inserimento, aggiornamento o cancellazione che ha attivato il trigger.

Tieni presente che i trigger DML sono basati su istruzioni, non basati su righe. Ciò significa che se l'istruzione ha effettuato più di una riga, le tabelle inserite o eliminate conterranno più di una riga.

Esempi:

```
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
```

Tutti gli esempi precedenti aggiungeranno record a tblAudit ogni volta che un record viene aggiunto, eliminato o aggiornato in tblSomething.

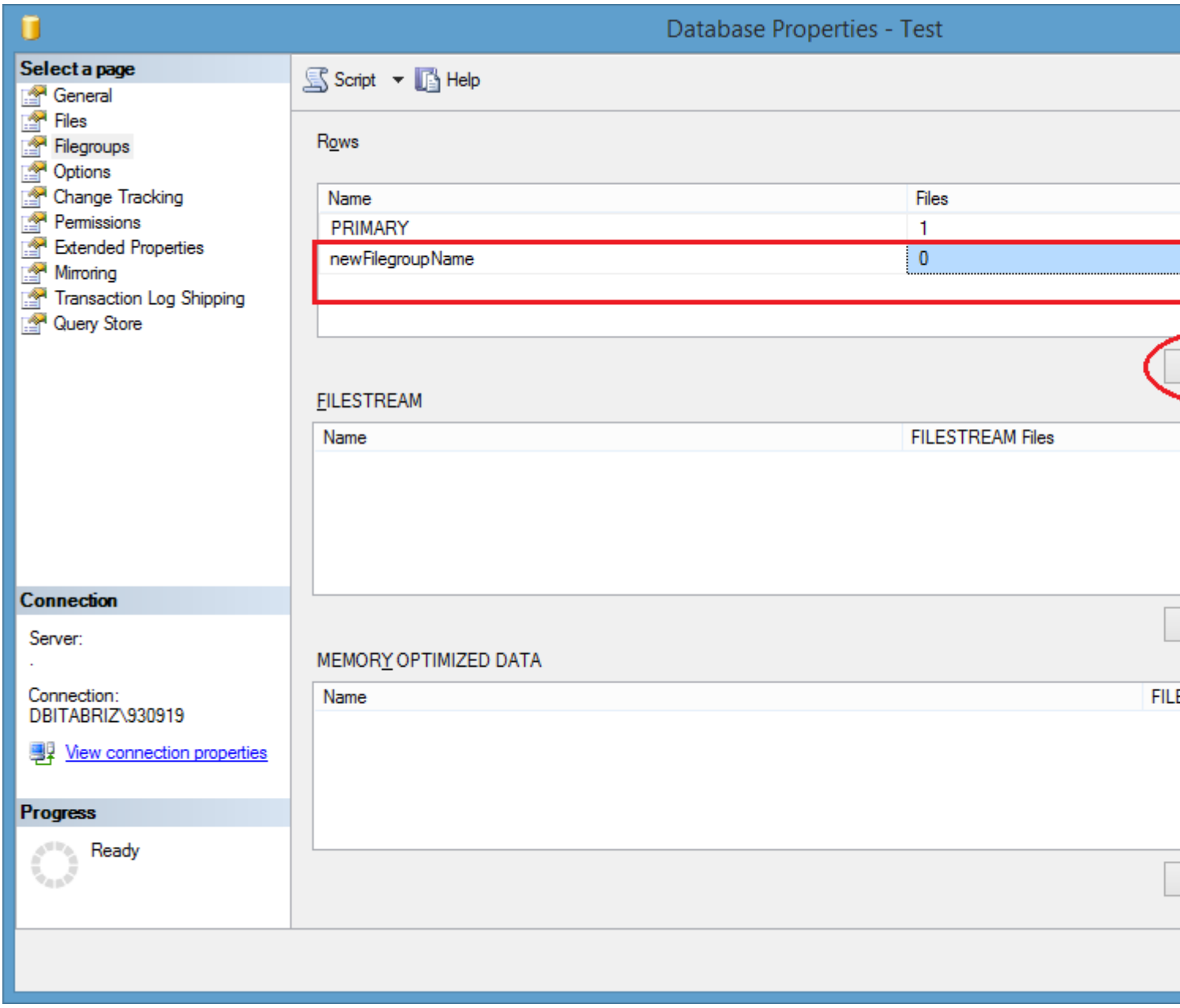
Leggi grilletto online: https://riptutorial.com/it/sql-server/topic/5032/grilletto

Capitolo 44: Gruppo di file

Examples

Crea filegroup nel database

Possiamo crearlo in due modi. Prima dalla modalità di progettazione delle proprietà del database:



E con gli script sql:

```
USE master;
GO
-- Create the database with the default data
-- filegroup and a log file. Specify the
-- growth increment and the max size for the
```

```
-- primary data file.

CREATE DATABASE TestDB ON PRIMARY
(
    NAME = 'TestDB_Primary',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_Prm.mdf',
    SIZE = 1 GB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
), FILEGROUP TestDB_FG1
(
    NAME = 'TestDB_FG1_1',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_FG1_1.ndf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
),
(
    NAME = 'TestDB_FG1_2',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB_FG1_2.ndf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
) LOG ON
(
    NAME = 'TestDB_log',
    FILENAME = 'C:\Program Files\Microsoft SQL
Server\MSSQL12.MSSQLSERVER\MSSQL\DATA\TestDB.ldf',
    SIZE = 10 MB,
    MAXSIZE = 10 GB,
    FILEGROWTH = 1 GB
);

go
ALTER DATABASE TestDB MODIFY FILEGROUP TestDB_FG1 DEFAULT;
go

-- Create a table in the user-defined filegroup.
USE TestDB;
Go

CREATE TABLE MyTable
(
    col1 INT PRIMARY KEY,
    col2 CHAR(8)
)
ON TestDB_FG1;
GO
```

Leggi Gruppo di file online: https://riptutorial.com/it/sql-server/topic/5461/gruppo-di-file

Capitolo 45: Importazione BULK

Examples

INSERIMENTO BULK con opzioni

È possibile personalizzare le regole di analisi utilizzando diverse opzioni nella clausola WITH:

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

In questo esempio, CODEPAGE specifica che un file di origine nel file UTF-8 e TERMINATORS sono in coma e una nuova riga.

INSERIMENTO BULK

Il comando BULK INSERT può essere utilizzato per importare file in SQL Server:

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

Il comando BULK INSERT eseguirà il mapping delle colonne nei file con colonne nella tabella di destinazione.

Leggere l'intero contenuto del file usando OPENROWSET (BULK)

È possibile leggere il contenuto del file utilizzando la funzione OPENROWSET (BULK) e memorizzare il contenuto in alcune tabelle:

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

L'opzione SINGLE_BLOB leggerà l'intero contenuto da un file come singola cella.

Leggi il file usando OPENROWSET (BULK) e formatta il file

Yu può definire il formato del file che verrà importato usando l'opzione FORMATFILE:

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

Il file di formato, format_file.fmt, descrive le colonne 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
```

Leggi il file json usando OPENROWSET (BULK)

È possibile utilizzare OPENROWSET per leggere il contenuto del file e passarlo ad altre funzioni che analizzeranno i risultati.

L'esempio seguente mostra hot per leggere l'intero contenuto del file JSON utilizzando OPENROWSET (BULK) e quindi fornisce BulkColumn alla funzione OPENJSON che analizzerà JSON e restituirà le colonne:

```
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
```

Leggi Importazione BULK online: https://riptutorial.com/it/sql-server/topic/7330/importazione-bulk

Capitolo 46: Indice

Examples

Crea indice cluster

Con un indice cluster le pagine foglia contengono le righe della tabella effettiva. Pertanto, può esistere solo un indice cluster.

```
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
```

Crea indice non consolidato

Gli indici non in cluster hanno una struttura separata dalle righe di dati. Un indice non in cluster contiene i valori delle chiavi di indice non cluster e ogni voce di valore chiave ha un puntatore alla riga di dati che contiene il valore della chiave. È possibile avere un massimo di 999 indici non cluster su SQL Server 2008/2012.

Collegamento per riferimento: 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
```

Mostra informazioni sugli indici

```
SP_HELPINDEX tableName
```

Indice in vista

```
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)
```

Indice di caduta

```
DROP INDEX IX_NonClustered ON Employees
```

Restituisce indici di dimensioni e frammentazione

```
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');
```

Riorganizza e ricostruisce l'indice

valore avg_fragmentation_in_percent	Dichiarazione correttiva
> 5% e <= 30%	RIORGANIZZARE
> 30%	RICOSTRUIRE

```
ALTER INDEX IX_NonClustered ON tableName REORGANIZE;


ALTER INDEX ALL ON Production.Product
 REBUILD WITH (FILLFACTOR = 80, SORT_IN_TEMPDB = ON,
          STATISTICS_NORECOMPUTE = ON);
```

Ricreare o riorganizzare tutti gli indici su una tabella

La ricostruzione degli indici viene eseguita utilizzando la seguente dichiarazione

```
ALTER INDEX All ON tableName REBUILD;
```

Questo elimina l'indice e lo ricrea, rimuovendo la frustrazione, recupera lo spazio su disco e riordina le pagine indice.

Si può anche riorganizzare un indice usando

```
ALTER INDEX All ON tableName REORGANIZE;
```

che utilizzerà risorse di sistema minime e deframmenta il livello foglia di indici cluster e non cluster su tabelle e viste riordinando fisicamente le pagine a livello di foglia in modo che corrispondano all'ordine logico, da sinistra a destra, dei nodi foglia

Ricostruisci tutto il database degli indici

```
EXEC sp_MSForEachTable 'ALTER INDEX ALL ON ? REBUILD'
```

Indagini sull'indice

Potresti usare "SP_HELPINDEX Table_Name", ma Kimberly Tripp ha una stored procedure (che può essere trovata qui), che è un esempio migliore, poiché mostra di più sugli indici, incluse le colonne e la definizione del filtro, ad esempio:
Uso:

```
USE Adventureworks
EXEC sp_SQLskills_SQL2012_helpindex 'dbo.Product'
```

In alternativa, Tibor Karaszi ha una procedura memorizzata (trovata qui). Più avanti mostrerà le informazioni sull'utilizzo dell'indice e, facoltativamente, fornirà un elenco di suggerimenti sugli indici. Uso:

```
USE Adventureworks
EXEC sp_indexinfo 'dbo.Product'
```

Leggi Indice online: https://riptutorial.com/it/sql-server/topic/4998/indice

Capitolo 47: Indicizzazione di testo completo

Examples

A. Creazione di un indice univoco, un catalogo full-text e un indice full-text

Nell'esempio seguente viene creato un indice univoco nella colonna JobCandidateID della tabella HumanResources.JobCandidate del database di esempio AdventureWorks2012. L'esempio crea quindi un catalogo full-text predefinito, ft. Infine, l'esempio crea un indice full-text sulla colonna Resume, usando il catalogo ft e l'elenco di stop del sistema.

```
USE AdventureWorks2012;
GO
CREATE UNIQUE INDEX ui_ukJobCand ON HumanResources.JobCandidate(JobCandidateID);
CREATE FULLTEXT CATALOG ft AS DEFAULT;
CREATE FULLTEXT INDEX ON HumanResources.JobCandidate(Resume)
   KEY INDEX ui_ukJobCand
   WITH STOPLIST = SYSTEM;
GO
```

https://www.simple-talk.com/sql/learn-sql-server/understanding-full-text-indexing-in-sql-server/

https://msdn.microsoft.com/en-us/library/cc879306.aspx

https://msdn.microsoft.com/en-us/library/ms142571.aspx

Creazione di un indice full-text su diverse colonne di tabelle

```
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
```

Creazione di un indice full-text con un elenco di proprietà di ricerca senza popolarlo

```
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
```

E popolarlo in seguito con

```
ALTER FULLTEXT INDEX ON Production.Document SET CHANGE_TRACKING AUTO;
GO
```

Ricerca full-text

```
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;
```

Per ulteriori informazioni dettagliate https://msdn.microsoft.com/en-us/library/ms142571.aspx

Leggi Indicizzazione di testo completo online: https://riptutorial.com/it/sql-server/topic/4557/indicizzazione-di-testo-completo

Capitolo 48: In-Memory OLTP (Hekaton)

Examples

Crea tabella ottimizzata per la memoria

```
-- 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
```

Mostra i file .dll creati e le tabelle per le tabelle ottimizzate per la memoria

```
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
```

Mostra tutte le tabelle ottimizzate per la memoria:

```
SELECT
    name,type_desc,durability_desc,Is_memory_Optimized
FROM sys.tables
    WHERE Is_memory_Optimized = 1
GO
```

Tipi di tabella ottimizzati per memoria e tabelle temporali

Ad esempio, questo è il tipo di tabella basato su tempdb tradizionale:

```
CREATE TYPE dbo.testTableType AS TABLE
(
   col1 INT NOT NULL,
   col2 CHAR(10)
);
```

Per ottimizzare la memoria di questo tipo di tabella è sufficiente aggiungere l'opzione `memory_optimized=on` e aggiungere un indice se non ne esiste nessuno nel tipo originale:

```
CREATE TYPE dbo.testTableType AS TABLE
(
   col1 INT NOT NULL,
   col2 CHAR(10)
```

```
)WITH (MEMORY_OPTIMIZED=ON);
```

La tabella temporanea globale è così:

```
CREATE TABLE ##tempGlobalTabel
(
    Col1   INT   NOT NULL ,
    Col2   NVARCHAR(4000)
);
```

Tabella temporanea globale ottimizzata per la memoria:

```
CREATE TABLE dbo.tempGlobalTabel
(
    Col1   INT   NOT NULL   INDEX ix NONCLUSTERED,
    Col2   NVARCHAR(4000)
)
    WITH
        (MEMORY_OPTIMIZED = ON,
         DURABILITY = SCHEMA_ONLY);
```

Per ottimizzare la memoria delle tabelle temporali globali (## temp):

1. Crea una nuova `SCHEMA_ONLY` ottimizzata per la memoria con lo stesso schema della tabella `##temp` globale `##temp`
    - Assicurati che la nuova tabella abbia almeno un indice
2. Cambia tutti i riferimenti a `##temp` nelle tue istruzioni Transact-SQL sul nuovo temp di tabella ottimizzato per la memoria
3. Sostituisci le `DROP TABLE ##temp` nel tuo codice con `DELETE FROM temp` , per ripulire il contenuto
4. Rimuovere le istruzioni `CREATE TABLE ##temp` dal codice, che ora sono ridondanti

più informazioni

Dichiara le variabili di tabella ottimizzate per la memoria

Per prestazioni più veloci è possibile ottimizzare la memoria della variabile di tabella. Ecco il T-SQL per una variabile di tabella tradizionale:

```
DECLARE @tvp TABLE
(
    col1   INT NOT NULL ,
    Col2   CHAR(10)
);
```

Per definire le variabili ottimizzate per la memoria, devi prima creare un tipo di tabella ottimizzato per la memoria e quindi dichiarare una variabile da esso:

```
CREATE TYPE dbo.memTypeTable
AS TABLE
(
    Col1  INT NOT NULL INDEX ix1,
```

```
    Col2  CHAR(10)
)
WITH
    (MEMORY_OPTIMIZED = ON);
```

Quindi possiamo usare il tipo di tabella in questo modo:

```
DECLARE @tvp memTypeTable
insert INTO @tvp
values (1,'1'),(2,'2'),(3,'3'),(4,'4'),(5,'5'),(6,'6')

SELECT * FROM @tvp
```

Risultato:

```
Col1    Col2
1       1
2       2
3       3
4       4
5       5
6       6
```

Crea una tabella temporale con versione della memoria ottimizzata per il sistema

```
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 )
)
```

più informazioni

Leggi In-Memory OLTP (Hekaton) online: https://riptutorial.com/it/sql-server/topic/5295/in-memory-oltp--hekaton-

Capitolo 49: Inserire

Examples

Aggiungi una riga a una tabella chiamata Fatture

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

- I nomi delle colonne sono obbligatori se la tabella che stai inserendo contiene una colonna con l'attributo IDENTITY.

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

Leggi Inserire online: https://riptutorial.com/it/sql-server/topic/5323/inserire

Capitolo 50: INSERIRE

introduzione

L'istruzione INSERT INTO viene utilizzata per inserire nuovi record in una tabella.

Examples

INSERISCI Hello World INTO table

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

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

INSERISCI su colonne specifiche

Per fare un inserimento su colonne specifiche (a differenza di tutte loro) devi specificare le colonne che vuoi aggiornare.

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

Funzionerà solo se le colonne che non hai elencato sono annullabili, identità, tipo di data e ora o colonne calcolate; o colonne che hanno un vincolo di valore predefinito. Pertanto, se qualcuno di questi è non nullable, non-identità, non-timestamp, colonne di valore non calcolate, non predefinite ... allora il tentativo di questo tipo di inserimento attiverà un messaggio di errore che ti dice che devi fornire un valore per i campi applicabili.

INSERISCI più righe di dati

Per inserire più righe di dati in SQL Server 2008 o versioni successive:

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

Per inserire più righe di dati nelle versioni precedenti di SQL Server, utilizzare "UNION ALL" in questo modo:

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

Nota, la parola chiave "INTO" è facoltativa nelle query INSERT. Un altro avviso è che il server SQL supporta solo 1000 righe in un unico INSERT, quindi è necessario dividerli in lotti.

INSERISCI una singola riga di dati

Una singola riga di dati può essere inserita in due modi:

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

O

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

Si noti che la seconda istruzione di inserimento consente solo i valori esattamente nello stesso ordine delle colonne della tabella mentre nel primo inserimento, l'ordine dei valori può essere modificato come:

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

Usa OUTPUT per ottenere il nuovo ID

Quando INSERISCI, puoi utilizzare `OUTPUT INSERTED.ColumnName` per ottenere i valori dalla riga appena inserita, ad esempio l'Id appena generato, utile se hai una colonna `IDENTITY` o qualsiasi tipo di valore predefinito o calcolato.

Quando si chiama in modo programmatico questo (ad esempio, da ADO.net) lo si considererebbe una query normale e si leggeranno i valori come se avessi fatto un `SELECT` -statement.

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

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

Se l'ID della riga aggiunta recentemente è richiesto all'interno della stessa serie di query o stored procedure.

```
-- 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)
```

INSERISCI dai risultati della query SELECT

Per inserire i dati recuperati dalla query SQL (righe singole o multiple)

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

Nota, 'studente' in SELECT è una costante di stringa che verrà inserita in ogni riga.

Se necessario, è possibile selezionare e inserire dati da / nella stessa tabella

Leggi INSERIRE online: https://riptutorial.com/it/sql-server/topic/3814/inserire

Capitolo 51: Installazione di SQL Server su Windows

Examples

introduzione

Queste sono le edizioni disponibili di SQL Server, come detto da Editions Matrix :

- Express: database gratuito entry-level. Include funzionalità RDBMS di base. Limitato a 10G di dimensione del disco. Ideale per sviluppo e test.
- Edizione standard: edizione con licenza standard. Include funzionalità di base e funzionalità di Business Intelligence.
- Enterprise Edition: edizione completa di SQL Server. Include funzionalità avanzate di sicurezza e di data warehousing.
- Developer Edition: include tutte le funzionalità di Enterprise Edition e nessuna limitazione, ed è gratuito da scaricare e utilizzare solo per scopi di sviluppo.

Dopo aver scaricato / acquisito SQL Server, l'installazione viene eseguita con SQLSetup.exe, disponibile come GUI o programma da riga di comando.

L'installazione tramite uno di questi richiede di specificare un codice prodotto ed eseguire una configurazione iniziale che include funzioni di abilitazione, servizi separati e impostazione dei parametri iniziali per ciascuno di essi. È possibile abilitare servizi e funzionalità aggiuntivi in qualsiasi momento eseguendo il programma SQLSetup.exe nella riga di comando o nella versione della GUI.

Leggi Installazione di SQL Server su Windows online: https://riptutorial.com/it/sql-server/topic/5801/installazione-di-sql-server-su-windows

Capitolo 52: Interrogare i risultati per pagina

Examples

Row_number ()

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

Da cui produrrà un set di risultati con un campo RowID che è possibile utilizzare per passare da una pagina all'altra.

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

Leggi Interrogare i risultati per pagina online: https://riptutorial.com/it/sql-server/topic/5803/interrogare-i-risultati-per-pagina

Capitolo 53: Istantanee del database

Osservazioni

Un'istantanea del database è una vista statica di sola lettura di un database SQL Server che è coerente a livello di transazione con il database di origine al momento della creazione dello snapshot.

Un'istantanea del database risiede sempre sulla stessa istanza del server del suo database di origine. Quando il database di origine viene aggiornato, lo snapshot del database viene aggiornato.

Un'istantanea differisce da un backup poiché il processo di creazione dello snapshot è istantaneo e lo snapshot occupa lo spazio solo quando vengono applicate le modifiche nel database di origine. Un backup memorizza invece una copia completa dei dati come al momento della creazione del backup.

Inoltre, un'istantanea fornisce una copia istantanea di sola lettura del database, mentre un backup deve essere ripristinato su un server per poter essere letto (e una volta ripristinato può essere scritto anche)

Le istantanee del database sono disponibili solo nelle edizioni Enterprise e Developer.

Examples

Creare uno snapshot del database

Un'istantanea del database è una vista statica di sola lettura di un database di SQL Server (il database di origine). È simile al backup, ma è disponibile come qualsiasi altro database in modo che il client possa interrogare il database degli snapshot.

```
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
```

Puoi anche creare un'istantanea del database con più file:

```
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;
```

Ripristina un'istantanea del database

Se i dati in un database di origine vengono danneggiati o vengono scritti dati errati nel database, in alcuni casi, il ripristino del database su uno snapshot del database che precede il danno potrebbe essere un'alternativa appropriata al ripristino del database da un backup.

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

> **Attenzione:** questo *cancellerà tutte le modifiche* apportate al database di origine da quando è stata scattata l'istantanea!

CANCELLA Istantanea

È possibile eliminare le istantanee esistenti del database utilizzando l'istruzione DELETE DATABASE:

```
DROP DATABASE Mydatabase_morning
```

In questa dichiarazione si dovrebbe fare riferimento al nome dell'istantanea del database.

Leggi Istantanee del database online: https://riptutorial.com/it/sql-server/topic/677/istantanee-del-database

Capitolo 54: JSON in SQL Server

Sintassi

- **JSON_VALUE** (espressione, percorso): estrae un valore scalare da una stringa JSON.
- **JSON_QUERY** (espressione [, percorso]): estrae un oggetto o una matrice da una stringa JSON.
- **OPENJSON** (jsonExpression [, path]) - funzione valore di tabella che analizza il testo JSON e restituisce oggetti e proprietà in JSON come righe e colonne.
- **ISJSON** (espressione): verifica se una stringa contiene JSON valido.
- **JSON_MODIFY** (expression, path, newValue) - Aggiorna il valore di una proprietà in una stringa JSON e restituisce la stringa JSON aggiornata.

Parametri

parametri	Dettagli
espressione	Tipicamente il nome di una variabile o di una colonna che contiene testo JSON.
sentiero	Un'espressione del percorso JSON che specifica la proprietà da aggiornare. path ha la seguente sintassi: [append] [lax | strict] $. <percorso json>
jsonExpression	Espressione di caratteri Unicode contenente il testo JSON.

Osservazioni

La funzione OPENJSON è disponibile solo con il livello di compatibilità 130. Se il livello di compatibilità del database è inferiore a 130, SQL Server non sarà in grado di trovare ed eseguire la funzione OPENJSON. Attualmente tutti i database SQL di Azure sono impostati su 120 per impostazione predefinita. È possibile modificare il livello di compatibilità di un database utilizzando il seguente comando:

```
ALTER DATABASE <Database-Name-Here> SET COMPATIBILITY_LEVEL = 130
```

Examples

Formato risultati query come JSON con FOR JSON

Dati della tabella di input (tabella Persone)

Id	Nome	Età
1	John	23
2	Jane	31

domanda

```
SELECT Id, Name, Age
FROM People
FOR JSON PATH
```

Risultato

```
[
    {"Id":1,"Name":"John","Age":23},
    {"Id":2,"Name":"Jane","Age":31}
]
```

Analizzare il testo JSON

Le funzioni **JSON_VALUE** e **JSON_QUERY** analizzano il testo JSON e restituiscono valori scalari o oggetti / matrici sul percorso nel testo JSON.

```
DECLARE @json NVARCHAR(100) = '{"id": 1, "user":{"name":"John"}, "skills":["C#","SQL"]}'

SELECT
    JSON_VALUE(@json, '$.id') AS Id,
    JSON_VALUE(@json, '$.user.name') AS Name,
    JSON_QUERY(@json, '$.user') AS UserObject,
    JSON_QUERY(@json, '$.skills') AS Skills,
    JSON_VALUE(@json, '$.skills[0]') AS Skill0
```

Risultato

Id	Nome	UserObject	Abilità	Skill0
1	John	{ "Name": "John"}	["C #", "SQL"]	C #

Unisciti alle entità JSON padre e figlio utilizzando CROSS APPLY OPENJSON

Unisci gli oggetti parent con le loro entità figlio, ad esempio vogliamo una tabella relazionale di ogni persona e dei suoi hobby

```
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"}
        ]
    }
 ]'
```

domanda

```
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
```

In alternativa, questa query può essere scritta utilizzando la clausola WITH.

```
SELECT
    Id, person.PersonName, Hobby
FROM OPENJSON (@json)
WITH(
    Id int '$.id',
    PersonName nvarchar(100) '$.user.name',
    Hobbies nvarchar(max) '$.hobbies' AS JSON
) as person
CROSS APPLY OPENJSON(Hobbies)
WITH(
    Hobby nvarchar(100) '$.name'
)
```

Risultato

Id	PersonName	Passatempo
1	John	Lettura
1	John	Fare surf
2	Jane	Programmazione
2	Jane	In esecuzione

Indicizza sulle proprietà JSON utilizzando colonne calcolate

Quando si memorizzano i documenti JSON in SQL Server, è necessario essere in grado di filtrare e ordinare in modo efficiente i risultati delle query sulle proprietà dei documenti JSON.

```
CREATE TABLE JsonTable
(
    id int identity primary key,
    jsonInfo nvarchar(max),
    CONSTRAINT [Content should be formatted as JSON]
    CHECK (ISJSON(jsonInfo)>0)
)
```

```
INSERT INTO JsonTable
VALUES(N'{"Name":"John","Age":23}'),
(N'{"Name":"Jane","Age":31}'),
(N'{"Name":"Bob","Age":37}'),
(N'{"Name":"Adam","Age":65}')
GO
```

Data la tabella sopra Se vogliamo trovare la riga con il nome = 'Adam', dovremmo eseguire la seguente query.

```
SELECT *
FROM JsonTable Where
JSON_VALUE(jsonInfo, '$.Name') = 'Adam'
```

Tuttavia questo richiederà al server SQL di eseguire una tabella completa che su un grande tavolo non è efficace.

Per accelerare, vorremmo aggiungere un indice, ma non possiamo direttamente consultare le proprietà nel documento JSON. La soluzione consiste nell'aggiungere una colonna calcolata sul percorso JSON `$.Name` , quindi aggiungere un indice nella colonna calcolata.

```
ALTER TABLE JsonTable
ADD vName as JSON_VALUE(jsonInfo, '$.Name')

CREATE INDEX idx_name
ON JsonTable(vName)
```

Ora quando eseguiamo la stessa query, invece di una scansione completa della tabella, il server SQL utilizza un indice per cercare nell'indice non cluster e trovare le righe che soddisfano le condizioni specificate.

Nota: affinché il server SQL utilizzi l'indice, è necessario creare la colonna calcolata con la stessa espressione che si intende utilizzare nelle query, in questo esempio `JSON_VALUE(jsonInfo, '$.Name')` , tuttavia è anche possibile utilizzare il nome della colonna calcolata `vName`

Formattare una riga tabella come un singolo oggetto JSON utilizzando FOR JSON

L' opzione **WITHOUT_ARRAY_WRAPPER** nella clausola *FOR JSON* rimuoverà le parentesi dell'array dall'output JSON. Ciò è utile se si restituisce una riga singola nella query.

> Nota: questa opzione produrrà output JSON non valido se viene restituita più di una riga.

Dati della tabella di input (tabella Persone)

Id	Nome	Età
1	John	23
2	Jane	31

domanda

```
SELECT Id, Name, Age
FROM People
WHERE Id = 1
FOR JSON PATH, WITHOUT_ARRAY_WRAPPER
```

Risultato

```
{"Id":1,"Name":"John","Age":23}
```

Analizza il testo JSON usando la funzione OPENJSON

La funzione **OPENJSON** analizza il testo JSON e restituisce più uscite. I valori che devono essere restituiti vengono specificati utilizzando i percorsi definiti nella clausola WITH. Se per alcune colonne non è specificato un percorso, il nome della colonna viene utilizzato come percorso. Questa funzione trasmette i valori restituiti ai tipi SQL definiti nella clausola WITH. L'opzione AS JSON deve essere specificata nella definizione della colonna se alcuni oggetti / array devono essere restituiti.

```
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]')
```

Risultato

Id	Nome	UserObject	Abilità	Skill0
1	John	{ "Name": "John"}	["C #", "SQL"]	C #

Leggi JSON in SQL Server online: https://riptutorial.com/it/sql-server/topic/2568/json-in-sql-server

Capitolo 55: La funzione STUFF

Parametri

Parametro	Dettagli
character_expression	la stringa esistente nei tuoi dati
posizione di partenza	la posizione in `character_expression` per eliminare la `length` e quindi inserire la `replacement_string`
lunghezza	il numero di caratteri da eliminare da `character_expression`
replacement_string	la sequenza di caratteri da inserire in `character_expression`

Examples

Sostituzione base dei caratteri con STUFF ()

La funzione `STUFF()` inserisce una stringa in un'altra stringa eliminando prima un numero specificato di caratteri. Nell'esempio seguente, cancella "Svr" e lo sostituisce con "Server". Ciò accade specificando la `start_position` e la `length` della sostituzione.

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

L'esecuzione di questo esempio comporterà la restituzione della `SQL Server Documentation` anziché della `SQL Svr Documentation`.

Utilizzo di FOR XML per concatenare valori da più righe

Un uso comune per la funzione `FOR XML` consiste nel concatenare i valori di più righe.

Ecco un esempio utilizzando la tabella Clienti :

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

Nell'esempio sopra, `FOR XML PATH('')` viene utilizzato per concatenare gli indirizzi di posta elettronica, utilizzando `;` come il carattere delimitatore. Inoltre, lo scopo di `STUFF` è quello di rimuovere il leader `;` dalla stringa concatenata. `STUFF` esegue anche il casting implicito della stringa concatenata da XML a varchar.

Nota: il risultato dell'esempio precedente sarà codificato in XML, il che significa che sostituirà `<` caratteri con `<` ecc. Se non lo desideri, modifica `FOR XML PATH(''))` in `FOR XML PATH, TYPE).value('.[1]','varchar(MAX)')` , ad esempio:

```
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, '')
```

Questo può essere usato per ottenere un risultato simile a `GROUP_CONCAT` in MySQL o `string_agg` in PostgreSQL 9.0+, sebbene usiamo subquery invece di aggregati GROUP BY. (In alternativa, puoi installare un aggregato definito dall'utente come questo se stai cercando funzionalità più vicine a quelle di `GROUP_CONCAT`).

Ottieni i nomi delle colonne separati da virgola (non una lista)

```
/*
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
```

roba per la virgola separata nel server sql

`FOR XML PATH` e `STUFF` per concatenare più righe in una singola riga:

```
  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;
```

Esempio di base della funzione STUFF ().

STUFF (Original_Expression, Start, Length, Replacement_expression)

La funzione STUFF () inserisce Replace_expression, nella posizione iniziale specificata, insieme alla rimozione dei caratteri specificati usando il parametro Length.

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

L'esecuzione di questo esempio comporterà la restituzione della tabella indicata

Nome di battesimo	Cognome	E-mail	StuffedEmail
Jomes	Cacciatore	James@hotmail.com	J*****s@hotmail.com
Shyam	Rathod	Shyam@hotmail.com	S*****m@hotmail.com
ariete	shinde	Ram@hotmail.com	R ***** hotmail.com

Leggi La funzione STUFF online: https://riptutorial.com/it/sql-server/topic/703/la-funzione-stuff

Capitolo 56: Limite risultato impostato

introduzione

Man mano che le tabelle del database crescono, è spesso utile limitare i risultati delle query a un numero oa una percentuale fissi. Questo può essere ottenuto utilizzando la parola chiave `TOP` SQL Server o la clausola `OFFSET FETCH` .

Parametri

Parametro	Dettagli
`TOP`	Parola chiave limitante. Utilizzare con un numero.
`PERCENT`	Parola chiave percentuale. Viene dopo `TOP` e numero limitante.

Osservazioni

Se viene utilizzata la clausola `ORDER BY` , la limitazione si applica al set di risultati ordinato.

Examples

Limitando con TOP

Questo esempio limita il risultato `SELECT` a 100 righe.

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

È anche possibile utilizzare una variabile per specificare il numero di righe:

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

Limitazione con PERCENT

Questo esempio limita il risultato `SELECT` a 15 percento del numero totale di righe.

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

Limitare con FETCH

SQL Server 2012

`FETCH` è generalmente più utile per l'impaginazione, ma può essere utilizzato in alternativa a `TOP` :

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

Leggi Limite risultato impostato online: https://riptutorial.com/it/sql-server/topic/1555/limite-risultato-impostato

Capitolo 57: Livelli di isolamento delle transazioni

Sintassi

- IMPOSTA LIVELLO DI ISOLAMENTO DELLE TRANSAZIONI {LEGGI NON CORRISPONDENTE | LEGGI IMPEGNATI | LEGGI RIPETIBILI | SNAPSHOT | SERIALIZZABILE} [;]

Osservazioni

Riferimento MSDN: SET LIVELLO ISOLAMENTO TRANSAZIONE

Examples

Leggi non ammesso

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED
```

Questo è il livello di isolamento più permissivo, in quanto non causa alcun blocco. Specifica che le istruzioni possono leggere tutte le righe, comprese le righe che sono state scritte nelle transazioni ma non ancora confermate (vale a dire, sono ancora in transazione). Questo livello di isolamento può essere soggetto a "letture sporche".

Leggi Impegnato

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED
```

Questo livello di isolamento è il 2 ° più permissivo. Previene le letture sporche. Il comportamento di `READ COMMITTED` dipende dall'impostazione di `READ_COMMITTED_SNAPSHOT` :

- Se impostato su OFF (impostazione predefinita), la transazione utilizza blocchi condivisi per impedire ad altre transazioni di modificare le righe utilizzate dalla transazione corrente, nonché bloccare la transazione corrente dalla lettura di righe modificate da altre transazioni.
- Se impostato su ON, l' `READCOMMITTEDLOCK` tabella `READCOMMITTEDLOCK` può essere utilizzato per richiedere il blocco condiviso anziché il controllo delle versioni delle righe per le transazioni in esecuzione nella modalità `READ COMMITTED` .

Nota: `READ COMMITTED` è il comportamento predefinito di SQL Server.

Cosa sono le "letture sporche"?

Le letture sporche (o letture non vincolate) sono letture di righe che vengono modificate da una transazione aperta.

Questo comportamento può essere replicato utilizzando 2 query separate: una per aprire una transazione e scrivere alcuni dati su una tabella senza commit, l'altra per selezionare i dati da scrivere (ma non ancora impegnati) con questo livello di isolamento.

Query 1 - Prepara una transazione ma non la finisci:

```
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
```

Query 2 - Leggi le righe inclusa la transazione aperta:

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM dbo.demo;
```

Ritorna:

```
col1        col2
----------- ---------------------------------------
99          Normal transaction
42          Dirty read
```

PS: non dimenticare di pulire questi dati demo:

```
COMMIT TRANSACTION;
DROP TABLE dbo.demo;
GO
```

Leggi ripetibile

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ
```

Questo livello di isolamento della transazione è leggermente meno permissivo di `READ COMMITTED` ,

in quanto i blocchi condivisi sono collocati su tutti i dati letti da ciascuna istruzione nella transazione e vengono conservati **fino al completamento della transazione** , anziché essere rilasciati dopo ogni istruzione.

Nota: utilizzare questa opzione solo quando necessario, poiché è più probabile che causi un peggioramento delle prestazioni del database e deadlock di `READ COMMITTED` .

istantanea

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT
```

Specifica che i dati letti da qualsiasi istruzione in una transazione saranno la versione coerente dal punto di vista della transazione dei dati esistenti all'inizio della transazione, cioè, leggerà solo i dati che sono stati impegnati prima dell'avvio della transazione.

`SNAPSHOT` transazioni `SNAPSHOT` non richiedono o causano alcun blocco sui dati letti, in quanto vengono letti solo la versione (o istantanea) dei dati esistenti al momento dell'inizio della transazione.

Una transazione in esecuzione nel livello di isolamento `SNAPSHOT` legge solo le proprie modifiche dei dati mentre è in esecuzione. Ad esempio, una transazione potrebbe aggiornare alcune righe e quindi leggere le righe aggiornate, ma tale modifica sarà visibile solo alla transazione corrente fino a quando non viene confermata.

Nota: l'opzione del database `ALLOW_SNAPSHOT_ISOLATION` deve essere impostata su ON prima che sia possibile utilizzare il livello di isolamento `SNAPSHOT` .

Serializable

SQL Server 2008 R2

```
SET TRANSACTION ISOLATION LEVEL SERIALIZEABLE
```

Questo livello di isolamento è il più restrittivo. Richiede **intervallo blocca** l'intervallo di valori chiave che vengono letti da ciascuna istruzione nella transazione. Ciò significa anche che le istruzioni `INSERT` di altre transazioni verranno bloccate se le righe da inserire si trovano nell'intervallo bloccato dalla transazione corrente.

Questa opzione ha lo stesso effetto dell'impostazione di `HOLDLOCK` su tutte le tabelle in tutte le istruzioni `SELECT` in una transazione.

Nota: questo isolamento della transazione ha la concorrenza più bassa e dovrebbe essere utilizzato solo quando necessario.

Leggi Livelli di isolamento delle transazioni online: https://riptutorial.com/it/sql-

server/topic/5114/livelli-di-isolamento-delle-transazioni

Capitolo 58: Livelli di isolamento e bloccaggio

Osservazioni

Ho trovato questo link: è utile come riferimento: "Livelli di isolamento"

Examples

Esempi di impostazione del livello di isolamento

Esempio di impostazione del livello di isolamento:

```
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` - significa che una query nella transazione corrente non può accedere ai dati modificati da un'altra transazione che non è stata ancora eseguita - nessuna lettura sporca! MA, letture non ripetibili e letture fantasma sono possibili, perché i dati possono ancora essere modificati da altre transazioni.

2. `REPEATABLE READ` - significa che una query nella transazione corrente non può accedere ai dati modificati da un'altra transazione che non è stata ancora eseguita - nessuna lettura sporca! Nessuna altra transazione può modificare i dati letti dalla transazione corrente fino a quando non viene completata, eliminando le letture NON RIPETIBILI. MA, se un'altra transazione inserisce NEW ROW e la query viene eseguita più di una volta, le righe fantasma possono apparire a partire dalla seconda lettura (se corrisponde all'istruzione where della query).

3. `SNAPSHOT` - solo in grado di restituire i dati esistenti all'inizio della query. Garantisce la coerenza dei dati. Impedisce letture sporche, letture non ripetibili e letture fantasma. Per utilizzarlo è richiesta la configurazione del DB:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

4. `READ COMMITTED` - isolamento predefinito del server SQL. Impedisce la lettura dei dati modificati da un'altra transazione fino al commit. Utilizza i controlli di blocco condiviso e il controllo delle versioni delle righe sulle tabelle che impedisce le letture sporche. Dipende dalla configurazione DB READ_COMMITTED_SNAPSHOT - se abilitato - viene utilizzato il controllo delle versioni delle righe. abilitare: utilizzare questo:

```
ALTER DATABASE DBTestName SET ALLOW_SNAPSHOT_ISOLATION ON;GO;
SET TRANSACTION ISOLATION LEVEL READ COMMITTED; --return to the default one
```

5. `SERIALIZABLE` - utilizza i blocchi fisici acquisiti e trattenuti fino alla fine della transazione, che impedisce letture sporche, letture fantasma, letture non ripetibili. MA, impatta sulle prestazioni del DataBase, perché le transazioni simultanee sono serializzate e vengono eseguite una alla volta.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE ;
```

Leggi Livelli di isolamento e bloccaggio online: https://riptutorial.com/it/sql-server/topic/5331/livelli-di-isolamento-e-bloccaggio

Capitolo 59: Mascheramento dinamico dei dati

Examples

Maschera l'indirizzo email usando il mascheramento dinamico dei dati

Se hai una colonna email puoi mascherarla con la maschera email ():

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

Quando l'utente prova a selezionare le email dalla tabella Company, otterrà qualcosa come i seguenti valori:

mXXX@XXXX.com

zXXX@XXXX.com

rXXX@XXXX.com

Aggiungi maschera parziale sulla colonna

Puoi aggiungere una maschera parziale sulla colonna che mostrerà pochi caratteri dall'inizio e alla fine della stringa e mostrerà la maschera al posto dei caratteri nel mezzo:

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

Nei parametri della funzione parziale è possibile specificare quanti valori dall'inizio verranno mostrati, quanti valori verranno visualizzati alla fine e quale sarà il pattern mostrato nel mezzo.

Quando l'utente prova a selezionare le email dalla tabella Company, otterrà qualcosa come i seguenti valori:

(381) XXXXXXX39

(360) XXXXXXX01

(415) XXXXXXX05

Mostrare il valore casuale dell'intervallo usando la maschera casuale ()

Maschera casuale mostrerà un numero di rundom nell'intervallo specificato anziché il valore effettivo:

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

Si noti che in alcuni casi il valore visualizzato potrebbe corrispondere al valore effettivo nella colonna (se il numero selezionato casualmente corrisponde al valore nella cella).

Aggiunta di una maschera predefinita sulla colonna

Se si aggiunge la maschera predefinita sulla colonna, al posto del valore effettivo nell'istruzione SELECT verrà visualizzata la maschera:

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

Controllare chi può vedere i dati non mascherati

Puoi concedere agli utenti privilegiati il diritto di vedere i valori non mascherati usando la seguente dichiarazione:

```
GRANT UNMASK TO MyUser
```

Se alcuni utenti hanno già il permesso di smascherare, puoi revocare questa autorizzazione:

```
REVOKE UNMASK TO MyUser
```

Leggi Mascheramento dinamico dei dati online: https://riptutorial.com/it/sql-server/topic/7052/mascheramento-dinamico-dei-dati

Capitolo 60: Memorizzazione di JSON nelle tabelle SQL

Examples

JSON memorizzato come colonna di testo

JSON è il formato testuale, quindi è archiviato nelle colonne standard di NVARCHAR. La raccolta NoSQL è equivalente alla tabella dei valori delle chiavi a due colonne:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max)
)
```

Usa `nvarchar(max)` poiché non sei sicuro di quale sarebbe la dimensione dei tuoi documenti JSON. `nvarchar(4000)` e `varchar(8000)` hanno prestazioni migliori ma con limiti di dimensione a 8 KB.

Assicurati che JSON sia formattato correttamente usando ISJSON

Poiché JSON è una colonna di testo memorizzata, è possibile assicurarsi che sia formattata correttamente. Puoi aggiungere il vincolo CHECK sulla colonna JSON che controlla il testo JSON correttamente formattato:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max)
       CONSTRAINT [Data should be formatted as JSON]
       CHECK (ISJSON(Data) > 0)
)
```

Se hai già una tabella, puoi aggiungere il vincolo di controllo usando l'istruzione ALTER TABLE:

```
ALTER TABLE ProductCollection
    ADD CONSTRAINT [Data should be formatted as JSON]
        CHECK (ISJSON(Data) > 0)
```

Esporre valori dal testo JSON come colonne calcolate

Puoi esporre i valori dalla colonna JSON come colonne calcolate:

```
CREATE TABLE ProductCollection (
  Id int identity primary key,
  Data nvarchar(max),
  Price AS JSON_VALUE(Data, '$.Price'),
  Color JSON_VALUE(Data, '$.Color') PERSISTED
)
```

Se aggiungi la colonna calcolata PERSISTED, il valore del testo JSON verrà materializzato in questa colonna. In questo modo le tue query possono leggere più velocemente il valore dal testo JSON perché non è necessaria alcuna analisi. Ogni volta che viene modificato il JSON in questa riga, il valore verrà ricalcolato.

Aggiunta dell'indice sul percorso JSON

Le query che filtrano o ordinano i dati per un certo valore nella colonna JSON di solito utilizzano la scansione completa della tabella.

```
SELECT * FROM ProductCollection
WHERE JSON_VALUE(Data, '$.Color') = 'Black'
```

Per ottimizzare questo tipo di query, è possibile aggiungere una colonna calcolata non persistente che espone l'espressione JSON utilizzata nel filtro o nell'ordinamento (in questo esempio JSON_VALUE (Dati, '$ .Color')) e creare l'indice su questa colonna:

```
ALTER TABLE ProductCollection
ADD vColor as JSON_VALUE(Data, '$.Color')

CREATE INDEX idx_JsonColor
ON ProductCollection(vColor)
```

Le query utilizzeranno l'indice invece della scansione semplice della tabella.

JSON memorizzato nelle tabelle in memoria

Se è possibile utilizzare tabelle ottimizzate per la memoria, è possibile memorizzare JSON come testo:

```
CREATE TABLE ProductCollection (
  Id int identity primary key nonclustered,
  Data nvarchar(max)
) WITH (MEMORY_OPTIMIZED=ON)
```

Vantaggi di JSON in memoria:

- I dati JSON sono sempre in memoria, quindi non c'è accesso al disco
- Non ci sono blocchi e latch mentre si lavora con JSON

Leggi Memorizzazione di JSON nelle tabelle SQL online: https://riptutorial.com/it/sql-server/topic/5029/memorizzazione-di-json-nelle-tabelle-sql

Capitolo 61: MERGE

introduzione

A partire da SQL Server 2008, è possibile eseguire operazioni di inserimento, aggiornamento o eliminazione in un'unica istruzione utilizzando l'istruzione MERGE.

L'istruzione MERGE consente di unirsi a un'origine dati con una tabella o una vista di destinazione e quindi eseguire più azioni rispetto alla destinazione in base ai risultati di tale unione.

Sintassi

- Come da MSDN - https://msdn.microsoft.com/en-us/library/bb510625.aspx [WITH <espressione_comprensione_condivisa> [, ... n]] MERGE [TOP (espressione) [PERCENT]] [INTO] < target_table> [WITH (<merge_hint>)] [[AS] table_alias] UTILIZZO <table_source> ON <merge_search_condition> [QUANDO ABBINATO [AND <clause_search_condition>] THEN <merge_matched>] [... n] [QUANDO NON ABBINATO [BY TARGET] [AND <clause_search_condition>] THEN <merge_not_matched>] [QUANDO NON ABBINATI DA SOURCE [AND <clause_search_condition>] THEN <merge_matched>] [... n] [<output_clause>] [OPTION (<query_hint> [,. ..n])]; <target_table> :: = {[database_name. nome_schema. | nome_schema.] target_table} <merge_hint> :: = {{[<table_hint_limited> [, ... n]] [[,] INDICE (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> | derived_table [AS] table_alias [(column_alias [, ... n])] | <join_table> | <pivoted_table> | <unpivoted_table>} <merge_search_condition> :: = <search_condition> <merge_matched> :: = {UPDATE SET <set_clause> | DELETE} <set_clause> :: = SET {column_name = {espressione | DEFAULT | NULL} | {udt_column_name. {{property_name = expression | field_name = espressione} | method_name (argomento [, ... n])}} | column_name {.WRITE (expression, @Offset, @Length)} | @variabile = espressione | @variabile = colonna = espressione | column_name {+ = | - = | * = | / = | % = | & = | ^ = | | =} espressione | @variable {+ = | - = | * = | / = | % = | & = | ^ = | | =} espressione | @variable = column {+ = | - = | * = | / = | % = | & = | ^ = | | =} espressione} [, ... n] <merge_not_matched> :: = {INSERT [(column_list)] {VALUES (values_list) | VALORI PREDEFINITI}} <clause_search_condition> :: = <search_condition> :: = {[NOT] | (<search_condition>)} [{AND | OR} [NOT] {| (<search_condition>)}] [, ... n] :: = {espressione {= | <> | ! = |

  | > = | ! > | <| <= | ! <} espressione | string_expression [NOT] LIKE string_expression [ESCAPE 'escape_character'] | espressione [NOT] TRA espressione ed espressione | espressione IS [NOT] NULL | CONTAINS ({column | *}, '<contains_search_condition>') | FREETEXT ({column | *}, 'freetext_string') | espressione [NOT] IN (subquery | expression [, ... n]) | espressione {= | <> | ! = | | > = | ! > | <| <= | ! <} {TUTTI | ALCUNI | QUALSIASI} (sottoquery) | EXISTS (sottoquery)} <output_clause> :: = {[OUTPUT <dml_select_list> INTO

{@table_variable | output_table} [(column_list)]] [OUTPUT <dml_select_list>]} <dml_select_list> :: = {<nome_colonna> | scalar_expression} [[AS] column_alias_identifier] [, ... n] <column_name> :: = {DELETED | INSERITO | from_table_name}. {* | column_name} | $ azione

Osservazioni

Esegue operazioni di inserimento, aggiornamento o cancellazione su una tabella di destinazione in base ai risultati di un join con una tabella di origine. Ad esempio, è possibile sincronizzare due tabelle inserendo, aggiornando o eliminando le righe in una tabella in base alle differenze rilevate nell'altra tabella.

Examples

MERGE per inserire / aggiornare / eliminare

```
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
```

Descrizione:

- `MERGE INTO targetTable` - tabella da modificare
- `USING sourceTable` - fonte di dati (può essere una tabella o una vista o una funzione con valori di tabella)
- `ON ...` - condizioni di join tra `targetTable` e `sourceTable` .
- `WHEN MATCHED` : azioni da intraprendere quando viene trovata una corrispondenza
- - `AND (targetTable.PKID > 100)` - condizioni aggiuntive che devono essere soddisfatte affinché l'azione possa essere intrapresa
- `THEN DELETE` : elimina il record corrispondente dal `targetTable`
- `THEN UPDATE` - aggiorna le colonne del record corrispondente specificato da `SET ...`.
- `WHEN NOT MATCHED` : le azioni da intraprendere quando la corrispondenza non viene trovata in **`targetTable`**
- `WHEN NOT MATCHED BY SOURCE` : le azioni da intraprendere quando la corrispondenza non viene

trovata in **`sourceTable`**

Commenti:

Se non è necessaria un'azione specifica, allora ometti la condizione, ad esempio rimuovendo `WHEN NOT MATCHED THEN INSERT` , impedirai l'inserimento di record

La dichiarazione di unione richiede un punto e virgola.

restrizioni:

- `WHEN MATCHED` non consente l'azione `INSERT`
- `UPDATE` azione `UPDATE` può aggiornare una riga solo una volta. Ciò implica che la condizione di join deve produrre corrispondenze univoche.

Unisci utilizzando origine CTE

```
WITH SourceTableCTE AS
(
    SELECT * FROM SourceTable
)
MERGE
 TargetTable AS target
USING SourceTableCTE AS source
ON (target.PKID = source.PKID)
WHEN MATCHED THEN
    UPDATE SET target.ColumnA = source.ColumnA
WHEN NOT MATCHED THEN
    INSERT (ColumnA) VALUES (Source.ColumnA);
```

MERGE usando la tabella delle fonti derivate

```
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);
```

Unisci esempio: sincronizza la tabella di origine e di destinazione

Per illustrare la dichiarazione MERGE, considerare le seguenti due tabelle:

1. **dbo.Product** : questa tabella contiene informazioni sul prodotto che la società sta attualmente vendendo

2. **dbo.ProductNew** : questa tabella contiene informazioni sul prodotto che la società venderà in futuro.

Il seguente T-SQL creerà e popolerà queste due tabelle

```
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)
```

Ora, supponiamo di voler sincronizzare la tabella di destinazione dbo.Product con la tabella dbo.ProductNew. Ecco il criterio per questo compito:

1. I prodotti presenti sia nella tabella di origine dbo.ProductNew che nella tabella di destinazione dbo.Product vengono aggiornati nella tabella di destinazione dbo.Product con nuovi prodotti.

2. Qualsiasi prodotto nella tabella di origine dbo.ProductNew che non esiste nella tabella di destinazione dob.Product viene inserito nella tabella di destinazione dbo.Product.

3. Qualsiasi prodotto nella tabella di destinazione dbo.Product che non esiste nella tabella di origine dbo.ProductNew deve essere eliminato dalla tabella di destinazione dbo.Product. Ecco la dichiarazione MERGE per eseguire questa attività.

```
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.*;
```

Nota: il punto e virgola deve essere presente alla fine dell'istruzione MERGE.

	$action	ProductID	ProductName	PRICE	ProductID	ProductName	PRICE
1	UPDATE	1	Asus Notebook	300.00	1	IPod	300.00
2	UPDATE	2	Hp Notebook	400.00	2	IPhone	400.00
3	UPDATE	3	Dell Notebook	100.00	3	ChromeCast	100.00

Unisci utilizzando EXCEPT

Utilizzare EXCEPT per impedire aggiornamenti ai record non modificati

```
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;
```

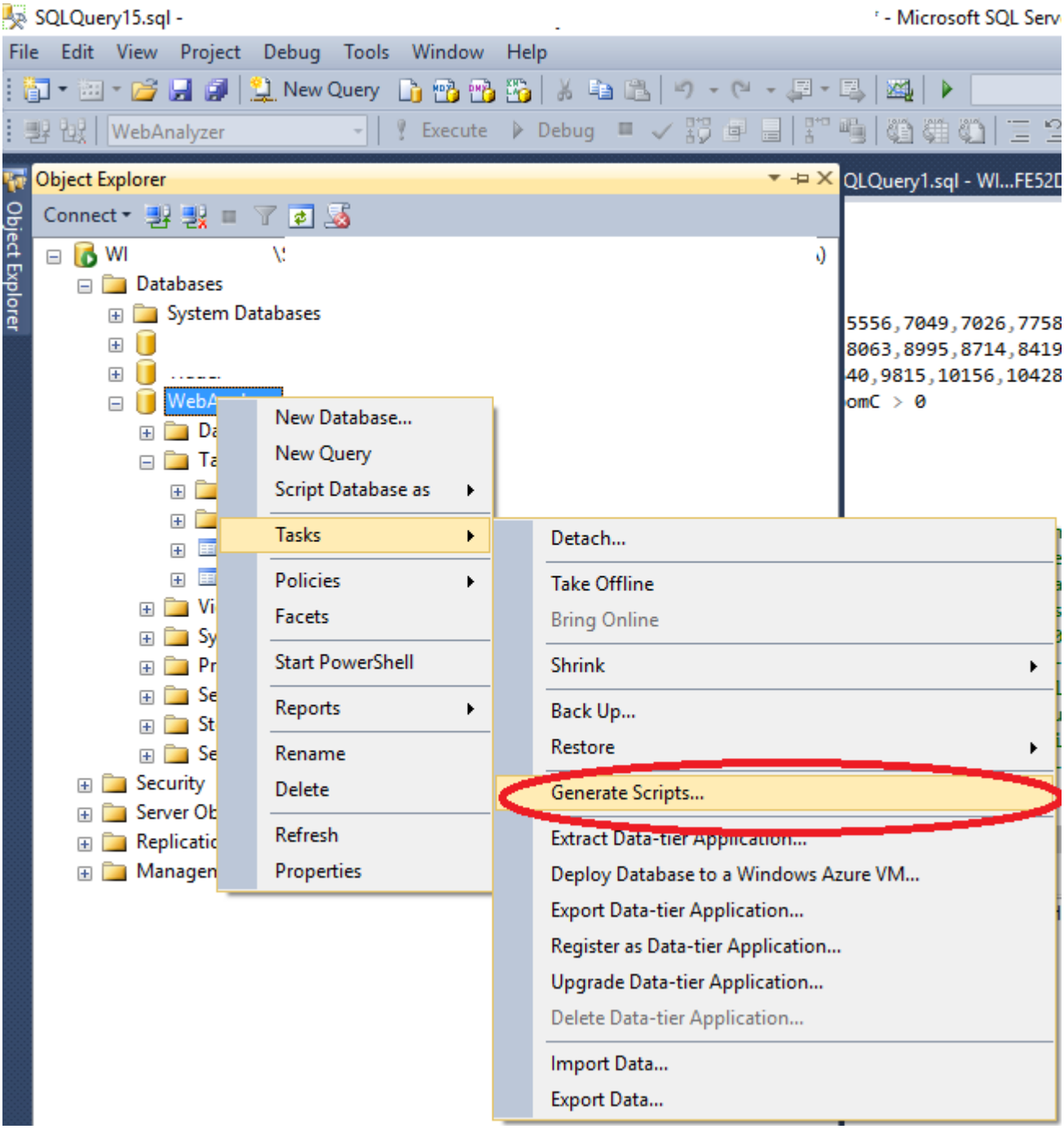
Leggi MERGE online: https://riptutorial.com/it/sql-server/topic/4550/merge

Capitolo 62: Migrazione

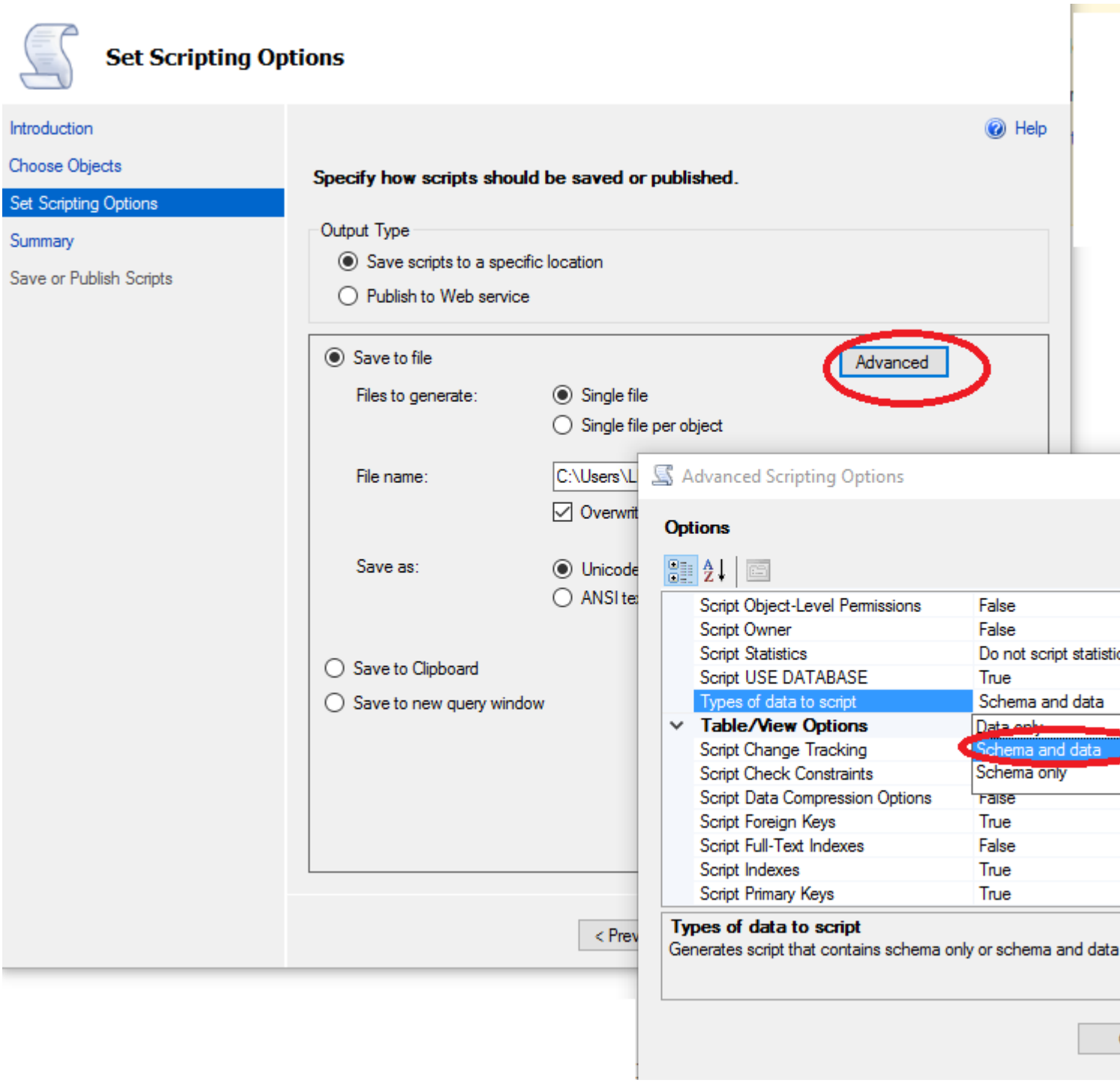
Examples

Come generare script di migrazione

1. Fare clic con il tasto `destro del mouse` sul database che si desidera migrare, quindi -> `Tasks` -> `Generate Scripts...`



2. Wizard si aprirà fare clic su `Next` quindi scegliere gli oggetti che si desidera migrare e fare nuovamente clic su `Next` , quindi fare clic su `Advanced` scorrere un po 'verso il basso e in `Types of data to script` scegliere `Schema and data` (a meno che non si desiderino solo strutture)



3. Fare clic su accoppiare più volte `Next` e `Finish` e si dovrebbe avere il database scritto nel file `.sql` .

4. eseguire il file `.sql` sul nuovo server e si dovrebbe fare.

Leggi Migrazione online: https://riptutorial.com/it/sql-server/topic/4451/migrazione

Capitolo 63: Modifica il testo JSON

Examples

Modifica il valore nel testo JSON sul percorso specificato

La funzione JSON_MODIFY utilizza il testo JSON come parametro di input e modifica un valore sul percorso specificato utilizzando il terzo argomento:

```
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}
```

Di conseguenza, avremo un nuovo testo JSON con "Prezzo": 39,99 e l'altro valore non verrà modificato. Se l'oggetto sul percorso specificato non esiste, JSON_MODIFY inserirà la chiave: coppia di valori.

Per eliminare la chiave: coppia di valori, inserisci NULL come nuovo valore:

```
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 per impostazione predefinita cancellerà la chiave se non ha valore, quindi puoi usarla per eliminare una chiave.

Aggiungi un valore scalare in un array JSON

JSON_MODIFY ha la modalità 'aggiungi' che aggiunge valore alla matrice.

```
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"]}
```

Se la matrice sul percorso specificato non esiste, JSON_MODIFY (append) creerà una nuova matrice con un singolo elemento:

```
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"]}
```

Inserisci un nuovo oggetto JSON nel testo JSON

La funzione JSON_MODIFY consente di inserire oggetti JSON nel testo JSON:

```
declare @json nvarchar(4000) = N'{"Id":1,"Name":"Toy Car"}'
set @json = JSON_MODIFY(@json, '$.Price',
```

```
                        JSON_QUERY('{"Min":34.99,"Recommended":45.49}'))
print @json
-- Output: {"Id":1,"Name":"Toy Car","Price":{"Min":34.99,"Recommended":45.49}}
```

Poiché il terzo parametro è testo, è necessario avvolgerlo con la funzione JSON_QUERY per "trasmettere" il testo a JSON. Senza questo "cast", JSON_MODIFY tratterà il terzo parametro come testo normale e sfuggirà ai caratteri prima di inserirlo come valore stringa. Senza JSON_QUERY i risultati saranno:

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

JSON_MODIFY inserirà questo oggetto se non esiste o lo eliminerà se il valore del terzo parametro è NULL.

Inserire un nuovo array JSON generato con la query FOR JSON

È possibile generare oggetti JSON utilizzando la query SELECT standard con la clausola FOR JSON e inserirla nel testo JSON come terzo parametro:

```
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 saprà che selezionare la query con la clausola FOR JSON genera un array JSON valido e lo inserirà semplicemente nel testo JSON.

> È possibile utilizzare tutte le opzioni FOR JSON nella query SELECT, ad **eccezione di WITHOUT_ARRAY_WRAPPER** , che genererà un singolo oggetto anziché un array JSON. Vedi altro esempio in questo argomento per vedere come inserire un singolo oggetto JSON.

Inserisci un singolo oggetto JSON generato con la clausola FOR JSON

È possibile generare oggetti JSON utilizzando la query SELECT standard con la clausola FOR JSON e l'opzione WITHOUT_ARRAY_WRAPPER e inserirla nel testo JSON come terzo parametro:

```
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 con WITHOUT_ARRAY_WRAPPER l'opzione può generare testo JSON non valido se la query SELECT restituisce più di un risultato (in questo caso è necessario utilizzare TOP 1 o filtro per chiave primaria). Pertanto, JSON_MODIFY assumerà che il risultato restituito sia solo un testo semplice e lo sfugga come qualsiasi altro testo se non lo si avvolge con la funzione JSON_QUERY.

> È necessario **eseguire il wrapping della** query **FOR JSON, WITHOUT_ARRAY_WRAPPER** con la funzione **JSON_QUERY** per poter eseguire il risultato su JSON.

Leggi Modifica il testo JSON online: https://riptutorial.com/it/sql-server/topic/6883/modifica-il-testo-json

Capitolo 64: Moduli compilati in modo nativo (Hekaton)

Examples

Stored procedure compilate in modo nativo

In una procedura con compilazione nativa, il codice T-SQL viene compilato in dll ed eseguito come codice C nativo. Per creare una stored procedure compilata nativa è necessario:

- Utilizzare la sintassi standard CREATE PROCEDURE
- Imposta l'opzione NATIVE_COMPILATION nella definizione della stored procedure
- Utilizzare l'opzione SCHEMABINDING nella definizione della stored procedure
- Definisci l'opzione ESEGUI COME PROPRIETARIO nella definizione della procedura memorizzata

Invece del blocco BEGIN END standard, è necessario utilizzare il blocco BEGIN ATOMIC:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Esempio:

```
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
```

Funzione scalare nativamente compilata

Il codice nella funzione compilata in modo nativo verrà trasformato in codice C e compilato come dll. Per creare una funzione scalare nativa compilata è necessario:

- Usa la sintassi CREATE FUNCTION standard

- Imposta l'opzione NATIVE_COMPILATION nella definizione della funzione
- Utilizzare l'opzione SCHEMABINDING nella definizione della funzione

Invece del blocco BEGIN END standard, è necessario utilizzare il blocco BEGIN ATOMIC:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Esempio:

```
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)
```

Funzione valore di tabella inline nativa

La funzione valore di tabella compilato nativo restituisce la tabella come risultato. Il codice nella funzione compilata in modo nativo verrà trasformato in codice C e compilato come dll. Nella versione 2016 sono supportate solo le funzioni con valori di tabella incorporati. Per creare una funzione di valore di tabella nativa è necessario:

- Usa la sintassi CREATE FUNCTION standard
- Imposta l'opzione NATIVE_COMPILATION nella definizione della funzione
- Utilizzare l'opzione SCHEMABINDING nella definizione della funzione

Invece del blocco BEGIN END standard, è necessario utilizzare il blocco BEGIN ATOMIC:

```
BEGIN ATOMIC
   WITH (TRANSACTION ISOLATION LEVEL=SNAPSHOT, LANGUAGE='us_english')
   -- T-Sql code goes here
END
```

Esempio:

```
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
);
```

Leggi Moduli compilati in modo nativo (Hekaton) online: https://riptutorial.com/it/sql-server/topic/6089/moduli-compilati-in-modo-nativo--hekaton-

Capitolo 65: Negozio di query

Examples

Abilita l'archivio query sul database

L'archivio query può essere abilitato sul database utilizzando il seguente comando:

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

SQL Server / Database SQL di Azure raccoglierà informazioni sulle query eseguite e fornirà informazioni nelle viste sys.query_store:

- sys.query_store_query
- sys.query_store_query_text
- sys.query_store_plan
- sys.query_store_runtime_stats
- sys.query_store_runtime_stats_interval
- sys.database_query_store_options
- sys.query_context_settings

Ottieni statistiche di esecuzione per query / piani SQL

La seguente query restituirà informazioni su qeries, i loro piani e le statistiche medie riguardanti la loro durata, il tempo della CPU, le letture fisiche e logiche.

```
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
```

Rimuovi i dati dall'archivio query

Se si desidera rimuovere alcune query o piani di query dall'archivio query, è possibile utilizzare i seguenti comandi:

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

I parametri per queste stored procedure sono l'ID della query / piano recuperato dalle visualizzazioni di sistema.

Puoi anche rimuovere le statistiche di esecuzione per un piano particolare senza rimuovere il piano dallo store:

```
EXEC sp_query_store_reset_exec_stats 3;
```

Parametro fornito a questo ID piano di procedura.

Forzare il piano per la query

SQL Query Optimizer sceglierà il piano possibile baes che può trovare per alcune query. Se è possibile trovare alcuni piani che funzionano in modo ottimale per alcune query, è possibile forzare QO a utilizzare sempre tale piano utilizzando la seguente stored procedure:

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

Da questo punto, QO utilizzerà sempre il piano fornito per la query.

Se si desidera rimuovere questa associazione, è possibile utilizzare la seguente stored procedure:

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

Da questo punto, QO tenterà nuovamente di trovare il piano migliore.

Leggi Negozio di query online: https://riptutorial.com/it/sql-server/topic/7349/negozio-di-query

Capitolo 66: Nomi alias in Sql Server

introduzione

Ecco alcuni modi diversi per fornire nomi alias alle colonne in Sql Server

Examples

Utilizzando AS

Questo è il metodo ANSI SQL funziona in tutti gli RDBMS. Approccio ampiamente usato.

```
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
```

Utilizzando =

Questo è il mio approccio preferito. Niente in relazione alle prestazioni è solo una scelta personale. Rende il codice pulito. Puoi vedere facilmente i nomi delle colonne risultanti invece di scorrere il codice se hai una grande espressione.

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

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

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

Dare alias dopo il nome della tabella derivata

Questo è un approccio strano che la maggior parte della gente non sa nemmeno che esiste.

```
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)
```

- **dimostrazione**

Senza usare AS

Questa sintassi sarà simile all'utilizzo della parola chiave `AS` . Solo noi non dobbiamo usare la parola chiave `AS`

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

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

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

Leggi Nomi alias in Sql Server online: https://riptutorial.com/it/sql-server/topic/10784/nomi-alias-in-sql-server

Capitolo 67: NULL

introduzione

In SQL Server, `NULL` rappresenta i dati mancanti o sconosciuti. Ciò significa che `NULL` non è realmente un valore; è meglio descritto come un segnaposto per un valore. Questo è anche il motivo per cui non è possibile confrontare `NULL` con alcun valore e nemmeno con un altro `NULL` .

Osservazioni

SQL Server fornisce altri metodi per gestire i valori null, ad esempio `IS NULL` , `IS NOT NULL` , `ISNULL()` , `COALESCE()` e altri.

Examples

Confronto NULL

`NULL` è un caso speciale quando si tratta di confronti.

Assumi i seguenti dati.

```
id someVal
 ----
 0 NULL
 1 1
 2 2
```

Con una query:

```
 SELECT id
 FROM table
 WHERE someVal = 1
```

restituire l'ID `1`

```
 SELECT id
 FROM table
 WHERE someVal <> 1
```

restituire l'ID `2`

```
 SELECT id
 FROM table
 WHERE someVal IS NULL
```

restituire l'ID `0`

```
SELECT id
FROM table
WHERE someVal IS NOT NULL
```

restituire entrambi gli ID `1` e `2` .

Se si desidera che i valori NULL vengano "contati" come valori nel confronto a `=` , `<>` , è necessario prima convertirli in un tipo di dati numerabile:

```
SELECT id
FROM table
WHERE ISNULL(someVal, -1) <> 1
```

O

```
SELECT id
FROM table
WHERE someVal IS NULL OR someVal <> 1
```

restituisce `0` e `2`

O è possibile modificare l'impostazione ANSI Null .

ANSI NULLI

Da MSDN

> In una versione futura di SQL Server, ANSI_NULLS sarà sempre attivo e tutte le applicazioni che impostano esplicitamente l'opzione su OFF genereranno un errore. Evitare l'uso di questa funzione in un nuovo lavoro di sviluppo e pianificare di modificare le applicazioni che attualmente utilizzano questa funzione.

`ANSI NULLS` impostato su off consente un confronto `=` / `<>` di valori nulli.

Dati i seguenti dati:

```
id someVal
 ----
 0 NULL
 1 1
 2 2
```

E con ANSI NULLS su questa query:

```
SELECT id
FROM table
WHERE someVal = NULL
```

non produrrebbe risultati Tuttavia la stessa query, con ANSI NULLS disattivato:

```
set ansi_nulls off

SELECT id
FROM table
WHERE someVal = NULL
```

Restituisce id `0` .

È ZERO()

La funzione `IsNull()` accetta due parametri e restituisce il secondo parametro se il primo è `null` .

parametri:

1. controllare l'espressione. Qualsiasi espressione di qualsiasi tipo di dati.
2. valore di sostituzione. Questo è il valore che verrebbe restituito se l'espressione di controllo è nullo. Il valore di sostituzione deve essere di un tipo di dati che può essere convertito implicitamente nel tipo di dati dell'espressione di controllo.

La funzione `IsNull()` restituisce lo stesso tipo di dati dell'espressione di controllo.

```
DECLARE @MyInt int -- All variables are null until they are set with values.

SELECT ISNULL(@MyInt, 3) -- Returns 3.
```

Vedi anche `COALESCE` , sopra

È null / Non è nullo

Poiché null non è un valore, non è possibile utilizzare operatori di confronto con valori null.
Per verificare se una colonna o una variabile contiene null, è necessario utilizzare `is null` :

```
DECLARE @Date date = '2016-08-03'
```

La seguente istruzione selezionerà il valore `6` , poiché tutti i confronti con valori nulli si considerano falsi o sconosciuti:

```
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
```

Impostando il contenuto della variabile @Date su `null` e riprova, la seguente dichiarazione restituirà `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 ()` Valuta gli argomenti in ordine e restituisce il valore corrente della prima espressione che inizialmente non valuta `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.
```

Sebbene ISNULL () funzioni in modo simile a COALESCE (), la funzione ISNULL () accetta solo due parametri, uno da verificare e uno da utilizzare se il primo parametro è NULL. Vedi anche `ISNULL` , sotto

NULL con NOT IN SubQuery

Durante la gestione non in sub-query con null nella sub-query, è necessario eliminare NULLS per ottenere i risultati attesi

```
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
```

Durante la gestione non in sub-query con null, prestare attenzione al risultato previsto

Leggi NULL online: https://riptutorial.com/it/sql-server/topic/5044/null

Capitolo 68: OPENJSON

Examples

Ottieni chiave: coppie di valori dal testo JSON

La funzione OPENJSON analizza il testo JSON e restituisce tutte le chiavi: coppie di valori al primo livello di JSON:

```
declare @json NVARCHAR(4000) = N'{"Name":"Joe","age":27,"skills":["C#","SQL"]}';
SELECT * FROM OPENJSON(@json);
```

chiave	valore	genere
Nome	Joe	1
età	27	2
abilità	["C #", "SQL"]	4

Il tipo di colonna descrive il tipo di valore, ad esempio null (0), string (1), number (2), boolean (3), array (4) e object (5).

Trasforma l'array JSON in un insieme di righe

La funzione OPENJSON analizza la raccolta di oggetti JSON e restituisce valori dal testo JSON come set di righe.

```
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
  )
```

Nella clausola WITH viene specificato lo schema di restituzione della funzione OPENJSON. Le chiavi negli oggetti JSON sono recuperate per nome di colonna. Se alcune chiavi in JSON non sono specificate nella clausola WITH (ad es. Price in questo esempio) verranno ignorate. I valori vengono automaticamente convertiti in tipi specificati.

Numero	Data	Cliente	Quantità
SO43659	2011-05-31T00: 00: 00	MSFT	1
SO43661	2011-06-01T00: 00: 00	Nokia	3

Trasforma i campi JSON nidificati in un insieme di righe

La funzione OPENJSON analizza la raccolta di oggetti JSON e restituisce valori dal testo JSON come set di righe. Se i valori nell'oggetto di input sono nidificati, è possibile specificare un parametro di mapping aggiuntivo in ogni colonna nella clausola WITH:

```
declare @json nvarchar(4000) = N'[
  {"data":{"num":"SO43659","date":"2011-05-
31T00:00:00"},"info":{"customer":"MSFT","Price":59.99,"qty":1}},
  {"data":{"number":"SO43661","date":"2011-06-
01T00:00:00"},"info":{"customer":"Nokia","Price":24.99,"qty":3}}
]'

SELECT    *
FROM OPENJSON (@json)
    WITH (
          Number   varchar(200) '$.data.num',
          Date     datetime '$.data.date',
          Customer varchar(200) '$.info.customer',
          Quantity int '$.info.qty',
  )
```

Nella clausola WITH viene specificato lo schema di restituzione della funzione OPENJSON. Dopo il tipo viene specificato il percorso dei nodi JSON in cui deve essere trovato il valore restituito. Le chiavi negli oggetti JSON vengono recuperate da questi percorsi. I valori vengono automaticamente convertiti in tipi specificati.

Numero	Data	Cliente	Quantità
SO43659	2011-05-31T00: 00: 00	MSFT	1
SO43661	2011-06-01T00: 00: 00	Nokia	3

Estrazione di oggetti secondari JSON interni

OPENJSON può estrarre frammenti di oggetti JSON all'interno del testo JSON. Nella definizione della colonna che fa riferimento all'oggetto secondario JSON, impostare il tipo nvarchar (max) e l'opzione AS JSON:

```
declare @json nvarchar(4000) = N'[
  {"Number":"SO43659","Date":"2011-05-
31T00:00:00","info":{"customer":"MSFT","Price":59.99,"qty":1}},
  {"Number":"SO43661","Date":"2011-06-
01T00:00:00","info":{"customer":"Nokia","Price":24.99,"qty":3}}
]'
```

```
SELECT    *
FROM OPENJSON (@json)
    WITH (
          Number   varchar(200),
          Date     datetime,
          Info nvarchar(max) '$.info' AS JSON
  )
```

La colonna Info verrà mappata sull'oggetto "Info". I risultati saranno:

Numero	Data	Informazioni
SO43659	2011-05-31T00: 00: 00	{ "Cliente": "MSFT", "Prezzo": 59,99, "qty": 1}
SO43661	2011-06-01T00: 00: 00	{ "Cliente": "Nokia", "Prezzo": 24.99, "qty": 3}

Lavorare con i sotto-array JSON nidificati

JSON può avere una struttura complessa con array interni. In questo esempio, abbiamo una matrice di ordini con sub array nidificato di 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}]}

]'
```

Possiamo analizzare le proprietà del livello root usando OPENJSON che restituirà il frammento AS JSON dell'array Items. Quindi possiamo applicare nuovamente OPENJSON sull'array Items e aprire la tabella JSON interna. La tabella del primo livello e la tabella interna saranno "uniti" come nel JOIN tra le tabelle standard:

```
SELECT    *
FROM
    OPENJSON (@json)
    WITH (  Number varchar(200), Date datetime,
            Items nvarchar(max) AS JSON )
        CROSS APPLY
            OPENJSON (Items)
            WITH ( Price float, Quantity int)
```

risultati:

Numero	Data	Elementi	Prezzo	Quantità
SO43659	2011-05-31 00: 00: 00.000	[{ "Prezzo": 11.99, "Quantità": 1}, { "Prezzo": 12.99, "Quantità": 5}]	11.99	1

Numero	Data	Elementi	Prezzo	Quantità
SO43659	2011-05-31 00: 00: 00.000	[{ "Prezzo": 11.99, "Quantità": 1}, { "Prezzo": 12.99, "Quantità": 5}]	12.99	5
SO43661	2011-06-01 00: 00: 00.000	[{ "Prezzo": 21.99, "Quantità": 3}, { "Prezzo": 22.99, "Quantità": 2}, { "Prezzo": 23.99, "Quantità": 2}]	21.99	3
SO43661	2011-06-01 00: 00: 00.000	[{ "Prezzo": 21.99, "Quantità": 3}, { "Prezzo": 22.99, "Quantità": 2}, { "Prezzo": 23.99, "Quantità": 2}]	22.99	2
SO43661	2011-06-01 00: 00: 00.000	[{ "Prezzo": 21.99, "Quantità": 3}, { "Prezzo": 22.99, "Quantità": 2}, { "Prezzo": 23.99, "Quantità": 2}]	23.99	2

Leggi OPENJSON online: https://riptutorial.com/it/sql-server/topic/5030/openjson

Capitolo 69: Operazioni DDL di base in MS SQL Server

Examples

Iniziare

Questa sezione descrive alcune **DDL** base (= **"D** ata **D** efinizione **L** anguage") comanda di creare un database, una tabella all'interno di un database, una vista e infine una procedura memorizzata.

Crea Database

Il seguente comando SQL crea un nuovo database `Northwind` sul server corrente, utilizzando il percorso `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
```

Nota: un database T-SQL è composto da due file, il file di database `*.mdf` e il suo log delle transazioni `*.ldf` . Entrambi devono essere specificati quando viene creato un nuovo database.

Crea tabella

Il seguente comando SQL crea una nuova tabella `Categories` nel database corrente, utilizzando lo schema `dbo` (puoi cambiare contesto di database con `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
```

Crea vista

Il seguente comando SQL crea una nuova vista `Summary_of_Sales_by_Year` nel database corrente, utilizzando lo schema `dbo` (puoi cambiare contesto di database con `Use <DatabaseName>`):

```
CREATE VIEW dbo.Summary_of_Sales_by_Year AS
  SELECT ord.ShippedDate, ord.OrderID, ordSub.Subtotal
  FROM Orders ord
  INNER JOIN [Order Subtotals] ordSub ON ord.OrderID = ordSub.OrderID
```

Questo si unirà tabelle `Orders` e `[Order Subtotals]` per visualizzare le colonne `ShippedDate` , `OrderID` e `Subtotal` . Poiché la tabella `[Order Subtotals]` ha uno spazio vuoto nel suo nome nel database Northwind, deve essere racchiusa tra parentesi quadre.

Crea procedura

Il seguente comando SQL crea una nuova stored procedure `CustOrdersDetail` nel database corrente, utilizzando lo schema `dbo` (è possibile cambiare il contesto del database con `Use <DatabaseName>`):

```
CREATE PROCEDURE dbo.MyCustOrdersDetail @OrderID int, @MinQuantity int=0
AS BEGIN
  SELECT ProductName,
    UnitPrice=ROUND(Od.UnitPrice, 2),
    Quantity,
    Discount=CONVERT(int, Discount * 100),
    ExtendedPrice=ROUND(CONVERT(money, Quantity * (1 - Discount) * Od.UnitPrice), 2)
  FROM Products P, [Order Details] Od
  WHERE Od.ProductID = P.ProductID and Od.OrderID = @OrderID
  and Od.Quantity>=@MinQuantity
END
```

Questa stored procedure, dopo che è stata creata, può essere richiamata come segue:

```
exec dbo.MyCustOrdersDetail 10248
```

che restituirà tutti i dettagli dell'ordine con @ OrderId = 10248 (e quantità> = 0 come predefinito).

Oppure puoi specificare il parametro opzionale

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

che restituirà solo ordini con una quantità minima di 10 (o più).

Leggi Operazioni DDL di base in MS SQL Server online: https://riptutorial.com/it/sql-server/topic/5463/operazioni-ddl-di-base-in-ms-sql-server

Capitolo 70: Opzioni avanzate

Examples

Abilita e mostra le opzioni avanzate

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

Abilita la compressione di backup predefinita

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

Imposta la percentuale del fattore di riempimento predefinito

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

Il server deve essere riavviato prima che la modifica possa avere effetto.

Imposta l'intervallo di recupero del sistema

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

Abilita il permesso di cmd

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

Imposta la dimensione massima della memoria del server

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

Imposta il numero di compiti del checkpoint

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

Leggi Opzioni avanzate online: https://riptutorial.com/it/sql-server/topic/5185/opzioni-avanzate

Capitolo 71: Ordinamento / ordine di file

Examples

Nozioni di base

Per prima cosa, impostiamo la tabella di esempio.

```
-- 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
```

Ricordare che quando si recuperano i dati, se non si specifica una clausola di ordinamento delle righe (ORDER BY), il server SQL non garantisce l'ordinamento (ordine delle colonne) **in qualsiasi momento** . Davvero, in qualsiasi momento. E non c'è motivo di discuterne, è stato mostrato letteralmente migliaia di volte e su Internet.

No ORDER BY == nessuna selezione. Fine della storia.

```
-- 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
```

Ci sono due direzioni in cui i dati possono essere ordinati da:

- ascendente (spostandosi verso l'alto), usando ASC
- discendente (spostandosi in basso), usando 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
```

Quando ordini la colonna testuale ((n) char o (n) varchar), fai attenzione che l'ordine rispetti le regole di confronto. Per ulteriori informazioni sulle regole di confronto, consultare l'argomento.

Ordinare e ordinare i dati può consumare risorse. Questo è dove gli indici creati correttamente vengono a portata di mano. Per ulteriori informazioni sugli indici, consultare l'argomento.

C'è la possibilità di pseudo-randomizzare l'ordine delle righe nel tuo gruppo di risultati. Basta forzare l'ordine ad apparire non deterministico.

```
SELECT * FROM SortOrder ORDER BY CHECKSUM(NEWID())
GO
```

L'ordine può essere ricordato in una stored procedure, e questo è il modo in cui dovresti farlo se è l'ultimo passo per manipolare il set di righe prima di mostrarlo all'utente finale.

```
CREATE PROCEDURE GetSortOrder
AS
    SELECT *
    FROM SortOrder
    ORDER BY ID DESC
GO

EXEC GetSortOrder
GO
```

Esiste anche un supporto limitato (e hacky) per l'ordine nelle viste di SQL Server, ma si consiglia di NON utilizzarlo.

```
/* 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
```

Per ordinare puoi utilizzare nomi di colonne, alias o numeri di colonna nel tuo ORDER BY.

```
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
```

Vi sconsiglio di usare i numeri nel vostro codice, a meno che non vogliate dimenticarcene il momento dopo averlo eseguito.

Ordina per caso

Se vuoi ordinare i tuoi dati in ordine numerico o alfabetico, puoi semplicemente utilizzare l' `order by [column]` . Se si desidera ordinare utilizzando una gerarchia personalizzata, utilizzare un'istruzione case.

```
Group
-----
Total
Young
MiddleAge
Old
Male
Female
```

Utilizzando un `order by` base `order by` :

```
Select * from MyTable
Order by Group
```

restituisce un ordinamento alfabetico, che non è sempre auspicabile:

```
Group
-----
Female
Male
MiddleAge
Old
Total
Young
```

Aggiunta di un'istruzione "caso", che assegna valori numerici crescenti nell'ordine in cui si

desidera ordinare i dati:

```
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
```

restituisce i dati nell'ordine specificato:

```
Group
-----
Total
Male
Female
Young
MiddleAge
Old
```

Leggi Ordinamento / ordine di file online: https://riptutorial.com/it/sql-server/topic/5332/ordinamento---ordine-di-file

Capitolo 72: ORDINATO DA

Osservazioni

Lo scopo della clausola ORDER BY è di ordinare i dati restituiti da una query.

È importante notare che l' **ordine delle righe restituite da una query non è definito a meno che non ci sia una clausola ORDER BY.**

Vedere la documentazione MSDN per i dettagli completi della clausola ORDER BY: https://msdn.microsoft.com/en-us/library/ms188385.aspx

Examples

Semplice clausola ORDER BY

Utilizzando la tabella Impiegati , di seguito è riportato un esempio per restituire le colonne Id, FName e LName nell'ordine (ascendente) LName:

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

Ritorna:

Id	FName	LName
2	John	Johnson
1	Giacomo	fabbro
4	Johnathon	fabbro
3	Michael	Williams

Per ordinare in ordine discendente aggiungere la parola chiave DESC dopo il parametro field, ad es. La stessa query nell'ordine decrescente LName è:

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

ORDINA DA più campi

È possibile specificare più campi per la clausola `ORDER BY` , in ordine ascendente o DESCending.

Ad esempio, utilizzando la http://stackoverflow.com/documentation/sql/280/example-databases/1207/item-sales-table#t=20160721131406643421 1 tabella, possiamo restituire una

query che ordina da SaleDate in ordine ascendente, e Quantità in ordine decrescente.

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

Si noti che la parola chiave `ASC` è facoltativa e i risultati vengono ordinati in ordine crescente di un dato campo per impostazione predefinita.

Ordina con logica complessa

Se vogliamo ordinare i dati in modo diverso per gruppo, possiamo aggiungere una sintassi `CASE` a `ORDER BY` . In questo esempio, vogliamo ordinare ai dipendenti del Dipartimento 1 il cognome e i dipendenti del Dipartimento 2 per stipendio.

Id	FName	LName	Numero di telefono	ManagerID	DepartmentID	Stipendio	Data di assunzione
1	Giacomo	fabbro	1234567890	NULLO	1	1000	01-01-2002
2	John	Johnson	2468101214	1	1	400	23-03-2005
3	Michael	Williams	1357911131	1	2	600	12-05-2009
4	Johnathon	fabbro	1212121212	2	1	500	24-07-2016
5	Sam	sassone	1372141312	2	2	400	25-03-2015

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

Ordinazione personalizzata

Se si desidera ordinare in base a una colonna utilizzando qualcosa di diverso dall'ordinamento alfabetico / numerico, è possibile utilizzare il `case` per specificare l'ordine desiderato.

`order by Group` resi di `order by Group` :

Gruppo	Contare
Non ritirato	6
Pensionato	4
Totale	10

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

Gruppo	Contare
Totale	10
Pensionato	4
Non ritirato	6

Leggi ORDINATO DA online: https://riptutorial.com/it/sql-server/topic/4149/ordinato-da

Capitolo 73: paginatura

introduzione

Offset e paging di riga in varie versioni di SQL Server

Sintassi

- SELECT * FROM TableName ORDINA DA id OFFSET 10 ROWS FETCH NEXT 10 ROWS SOLO;

Examples

Impaginazione utilizzando ROW_NUMBER con un'espressione tabella comune

SQL Server 2008

La funzione `ROW_NUMBER` può assegnare un numero incrementale a ciascuna riga in un set di risultati. Combinato con un'espressione di tabella comune che utilizza un operatore `BETWEEN` , è possibile creare "pagine" di set di risultati. Ad esempio: pagina uno contenente i risultati 1-10, pagina due contenente i risultati 11-20, pagina 3 contenente i risultati 21-30 e così via.

```
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
```

Nota: non è possibile utilizzare `ROW_NUMBER` in una clausola `WHERE` come:

```
SELECT object_id,
    name,
    type,
    create_date
FROM sys.objects
WHERE ROW_NUMBER() OVER (ORDER BY name) BETWEEN 41 AND 50
```

Sebbene ciò sia più conveniente, SQL Server restituirà il seguente errore in questo caso:

> Messaggio 4108, livello 15, stato 1, riga 6

Le funzioni di finestra possono essere visualizzate solo nelle clausole SELECT o ORDER BY.

Paginazione con OFFSET FETCH

SQL Server 2012

La clausola `OFFSET FETCH` implementa la paginazione in modo più conciso. Con esso, è possibile saltare le righe N1 (specificate in `OFFSET`) e restituire le successive righe N2 (specificate in `FETCH`):

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 40 ROWS FETCH NEXT 10 ROWS ONLY
```

La clausola `ORDER BY` è necessaria per fornire risultati deterministici.

Paginaton con query interna

Nelle versioni precedenti di SQL Server, gli sviluppatori dovevano utilizzare il doppio ordinamento combinato con la parola chiave `TOP` per restituire righe in una pagina:

```
 SELECT TOP 10 *
 FROM
 (
    SELECT
    TOP 50 object_id,
        name,
        type,
        create_date
    FROM sys.objects
    ORDER BY name ASC
) AS data
ORDER BY name DESC
```

La query interna restituirà le prime 50 righe ordinate per `name` . Quindi la query esterna invertirà l'ordine di queste 50 righe e selezionerà le prime 10 righe (queste ultime saranno le ultime 10 righe nel gruppo prima dell'inversione).

Cercapersone in varie versioni di 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 utilizzando ORDER BY OFFSET e FETCH NEXT

Per ottenere le 10 righe successive, esegui questa query:

```
SELECT * FROM TableName ORDER BY id OFFSET 10 ROWS FETCH NEXT 10 ROWS ONLY;
```

Punti chiave da considerare quando lo si utilizza:

- `ORDER BY` è obbligatorio per utilizzare la clausola `OFFSET` e `FETCH` .
- `OFFSET` clausola `OFFSET` è obbligatoria con `FETCH` . Non puoi mai usare, `ORDER BY` ... `FETCH` .
- `TOP` non può essere combinato con `OFFSET` e `FETCH` nella stessa espressione di query.

Leggi paginatura online: https://riptutorial.com/it/sql-server/topic/6874/paginatura

Capitolo 74: Parametri valutati a tabella

Osservazioni

I parametri con valori di tabella (TVP in breve) sono parametri passati a una stored procedure o funzione che contiene dati strutturati in tabelle. L'utilizzo di parametri con valori di tabella richiede la creazione di un tipo di tabella definito dall'utente per il parametro utilizzato.

I parametri con valori memorizzati sono parametri di sola lettura.

Examples

Utilizzo di un parametro con valori di tabella per inserire più righe in una tabella

Innanzitutto, definisci un tipo di tabella definito usato da utilizzare:

```
CREATE TYPE names as TABLE
(
    FirstName varchar(10),
    LastName varchar(10)
)
GO
```

Creare la stored procedure:

```
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
```

Esecuzione della stored procedure:

```
DECLARE @names dbo.Names
INSERT INTO @Names VALUES
('Zohar', 'Peled'),
('First', 'Last')

EXEC dbo.prInsertNames @Names
```

Leggi Parametri valutati a tabella online: https://riptutorial.com/it/sql-server/topic/5285/parametri-valutati-a-tabella

Capitolo 75: ParseName

Sintassi

- PARSENAME ('nome_oggetto', oggetto_oggetto)

Parametri

'OBJECT_NAME'	object_piece
È il nome dell'oggetto per il quale recuperare la parte dell'oggetto specificato. nome_oggetto è sysname. Questo parametro è un nome oggetto qualificato facoltativamente. Se tutte le parti del nome dell'oggetto sono qualificate, questo nome può avere quattro parti: il nome del server, il nome del database, il nome del proprietario e il nome dell'oggetto.	L'oggetto parte da restituire. object_piece è di tipo int e può avere questi valori: 1 = Nome oggetto 2 = Nome schema 3 = Nome database 4 = Nome server

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
```

Ritorna:

server	DB
HeadofficeSQL1	Vento del nord

Proprietario	Oggetto
dbo	autori

Leggi ParseName online: https://riptutorial.com/it/sql-server/topic/5775/parsename

Capitolo 76: partizionamento

Examples

Recupera i valori al contorno della partizione

```
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
```

Commutazione delle partizioni

Secondo questa [pagina TechNet Microsoft] [1],

> **Il partizionamento dei dati** consente di gestire e accedere ai sottoinsiemi dei tuoi dati in modo rapido ed efficiente mantenendo l'integrità dell'intera raccolta di dati.

Quando si chiama la seguente query, i dati non vengono spostati fisicamente; solo i metadati relativi alla posizione dei dati cambiano.

```
ALTER TABLE [SourceTable] SWITCH TO [TargetTable]
```

Le tabelle devono avere le stesse colonne con gli stessi tipi di dati e le stesse impostazioni NULL, devono essere nello stesso gruppo di file e la nuova tabella di destinazione deve essere vuota. Vedi il link alla pagina qui sopra per maggiori informazioni sul cambio di partizione.

[1]: https://technet.microsoft.com/en-us/library/ms191160(v=sql.105).aspx La proprietà `IDENTITY` colonna potrebbe essere diversa.

Richiama i valori della tabella delle partizioni, della colonna, dello schema, della funzione, del totale e del minimo e massimo utilizzando una singola query

```
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)
```

Basta annullare un commento `where` clausola e sostituire `table_name` con `actual table name` per visualizzare i dettagli dell'oggetto desiderato.

Leggi partizionamento online: https://riptutorial.com/it/sql-server/topic/3212/partizionamento

Capitolo 77: PER IL PERCORSO XML

Osservazioni

Esistono anche diverse altre modalità `FOR XML` :

- `FOR XML RAW` : crea un elemento `<row>` per riga.
- `FOR XML AUTO` - Tenta di autorizzare euristicamente una gerarchia.
- `FOR XML EXPLICIT` : fornisce un maggiore controllo sulla forma dell'XML, ma è più ingombrante di `FOR XML PATH` .

Examples

Ciao World XML

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

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

Specifica degli spazi dei nomi

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>
```

Specifica della struttura usando le espressioni XPath

```
SELECT
    'XPath example' AS 'head/title',
    'This example demonstrates ' AS 'body/p',
    'https://www.w3.org/TR/xpath/' AS 'body/p/a/@href',
    'XPath expressions' AS 'body/p/a'
FOR XML PATH('html')
```

```
<html>
```

```
    <head>
        <title>XPath example</title>
    </head>
    <body>
        <p>This example demonstrates <a href="https://www.w3.org/TR/xpath/">XPath
expressions</a></p>
    </body>
</html>
```

In `FOR XML PATH` , le colonne senza un nome diventano nodi di testo. `NULL` o `''` diventano quindi nodi di testo vuoti. Nota: puoi convertire una colonna con nome in una senza nome usando `AS *`

```
DECLARE @tempTable TABLE (Ref INT, Des NVARCHAR(100), Qty INT)
INSERT INTO @tempTable VALUES (100001, 'Normal', 1), (100002, 'Foobar', 1), (100003, 'Hello
World', 2)

SELECT ROW_NUMBER() OVER (ORDER BY Ref) AS '@NUM',
     'REF' AS 'FLD/@NAME', REF AS 'FLD', '',
     'DES' AS 'FLD/@NAME', DES AS 'FLD', '',
     'QTY' AS 'FLD/@NAME', QTY AS 'FLD'
FROM @tempTable
FOR XML PATH('LIN'), ROOT('row')
```

```
<row>
  <LIN NUM="1">
    <FLD NAME="REF">100001</FLD>
    <FLD NAME="DES">Normal</FLD>
    <FLD NAME="QTY">1</FLD>
  </LIN>
  <LIN NUM="2">
    <FLD NAME="REF">100002</FLD>
    <FLD NAME="DES">Foobar</FLD>
    <FLD NAME="QTY">1</FLD>
  </LIN>
  <LIN NUM="3">
    <FLD NAME="REF">100003</FLD>
    <FLD NAME="DES">Hello World</FLD>
    <FLD NAME="QTY">2</FLD>
  </LIN>
</row>
```

L'utilizzo di nodi di testo (vuoti) aiuta a separare il nodo di output precedente da quello successivo, in modo che SQL Server sappia iniziare un nuovo elemento per la colonna successiva. Altrimenti, viene confuso quando l'attributo esiste già su quello che pensa sia l'elemento "corrente".

Ad esempio, senza le stringhe vuote tra l'elemento e l'attributo `SELECT` , SQL Server restituisce un errore:

> La colonna "FLD / @ NAME" centrata sugli attributi non deve venire dopo un fratello non attributo-centrico nella gerarchia XML in PERCORSO FOR XML.

Si noti inoltre che questo esempio ha anche avvolto l'XML in un elemento radice chiamato `row` , specificato da `ROOT('row')`

Utilizzo di FOR XML PATH per concatenare i valori

Il `FOR XML PATH` può essere utilizzato per concatenare i valori in stringa. L'esempio seguente concatena i valori in una stringa `CSV` :

```
DECLARE @DataSource TABLE
(
    [rowID] TINYINT
   ,[FirstName] NVARCHAR(32)
);

INSERT INTO @DataSource ([rowID], [FirstName])
VALUES (1, 'Alex')
      ,(2, 'Peter')
      ,(3, 'Alexsandyr')
      ,(4, 'George');

SELECT STUFF
(
    (
        SELECT ',' + [FirstName]
        FROM @DataSource
        ORDER BY [rowID] DESC
        FOR XML PATH(''), TYPE
    ).value('.', 'NVARCHAR(MAX)')
    ,1
    ,1
    ,''
);
```

Poche note importanti:

- la clausola `ORDER BY` può essere utilizzata per ordinare i valori in un modo preferito
- se si utilizza un valore più lungo come separatore della concatenazione, anche il parametro della funzione `STUFF` deve essere modificato;

```
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
    ,''
);
```

- come l'opzione `TYPE` e la funzione `.value` vengono utilizzate, la concatenazione funziona con la stringa `NVARCHAR(MAX)`

Leggi PER IL PERCORSO XML online: https://riptutorial.com/it/sql-server/topic/727/per-il-percorso-xml

Capitolo 78: PER JSON

Examples

PER IL PERCORSO JSON

Formatta i risultati della query SELECT come testo JSON. FOR JSON PATH clausola viene aggiunta dopo la query:

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

I nomi delle colonne verranno utilizzati come chiavi in JSON e i valori delle celle verranno generati come valori JSON. Il risultato della query sarebbe un array di oggetti JSON:

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

I valori NULL nella colonna principal_id verranno ignorati (non verranno generati).

PER PERCORSO JSON con alias di colonne

FOR JSON PATH consente di controllare il formato del JSON di output utilizzando gli alias di colonna:

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

L'alias della colonna verrà utilizzato come nome della chiave. Gli alias di colonna separati da punti (data.name e data.type) verranno generati come oggetti nidificati. Se due colonne hanno lo stesso prefisso nella notazione del punto, saranno raggruppate insieme in un singolo oggetto (dati in questo esempio):

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

FOR JSON clausola senza matrice wrapper (singolo oggetto in uscita)

L'opzione WITHOUT_ARRAY_WRAPPER consente di generare un singolo oggetto anziché l'array. Usa questa opzione se sai che restituirai una singola riga / oggetto:

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

Un singolo oggetto verrà restituito in questo caso:

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

INCLUDE_NULL_VALUES

La clausola FOR JSON ignora i valori NULL nelle celle. Se vuoi generare "chiave": coppie nulle per le celle che contengono valori NULL, aggiungi l'opzione INCLUDE_NULL_VALUES nella query:

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

Verranno generati valori NULL nella colonna principal_id:

```
[
  {"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}
]
```

Racchiudere i risultati con l'oggetto ROOT

Gli avvolgitori hanno restituito l'array JSON nell'oggetto root aggiuntivo con la chiave specificata:

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

Il risultato della query sarebbe una matrice di oggetti JSON all'interno dell'oggetto wrapper:

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

PER JSON AUTO

Nidifica automaticamente i valori dalla seconda tabella come sub-array nidificato di oggetti JSON:

```
SELECT top 5 o.object_id, o.name, c.column_id, c.name
FROM sys.objects o
```

```
    JOIN sys.columns c ON o.object_id = c.object_id
FOR JSON AUTO
```

Il risultato della query sarebbe array di oggetti JSON:

```
[
  {
   "object_id":3,
   "name":"sysrscols",
   "c":[
        {"column_id":12,"name":"bitpos"},
        {"column_id":6,"name":"cid"}
       ]
  },
  {
    "object_id":5,
    "name":"sysrowsets",
    "c":[
         {"column_id":13,"name":"colguid"},
         {"column_id":3,"name":"hbcolid"},
         {"column_id":8,"name":"maxinrowlen"}
     ]
  }
]
```

Creazione di una struttura JSON nidificata personalizzata

Se è necessaria una struttura JSON complessa che non può essere creata utilizzando FOR JSON PATH o FOR JSON AUTO, è possibile personalizzare l'output JSON inserendo sotto-query FOR JSON come espressioni di colonna:

```
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
```

Ogni sottoquery produrrà risultati JSON che verranno inclusi nel contenuto principale di JSON.

Leggi PER JSON online: https://riptutorial.com/it/sql-server/topic/4661/per-json

Capitolo 79: PHANTOM legge

introduzione

Nei sistemi di database, l'isolamento determina in che modo l'integrità della transazione è visibile ad altri utenti e sistemi, quindi definisce come / quando le modifiche apportate da una operazione diventano visibili ad altre. La lettura fantasma può verificarsi quando si ricevono dati non ancora commessi nel database.

Osservazioni

È possibile leggere i vari `ISOLATION LEVEL` su MSDN

Examples

Livello di isolamento LEGGI NON CORRETTO

Creare una tabella di esempio su un database di esempio

```
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]
```

Ora apri un primo editor di query (sul database) inserisci il codice sottostante ed esegui (**non toccare il --rollback**) in questo caso inserisci una riga su DB ma **non** commetti modifiche.

```
begin tran

INSERT INTO Table_1 values('Title 1')

SELECT * FROM [Test].[dbo].[Table_1]

--rollback
```

Ora apri un Second Query Editor (sul database), inserisci il codice qui sotto ed esegui

```
begin tran

set transaction isolation level READ UNCOMMITTED

SELECT * FROM [Test].[dbo].[Table_1]
```

Si può notare che sul secondo editor è possibile visualizzare la riga appena creata (ma non impegnata) dalla prima transazione. Al primo editor eseguire il rollback (selezionare la parola rollback ed eseguire).

```
-- Rollback the first transaction
rollback
```

Esegui la query sul secondo editor e vedi che il record scompare (lettura fantasma), questo si verifica perché dici, alla seconda transazione per ottenere tutte le righe, anche i non inviati.

Ciò si verifica quando si modifica il livello di isolamento con

```
set transaction isolation level READ UNCOMMITTED
```

Leggi PHANTOM legge online: https://riptutorial.com/it/sql-server/topic/8235/phantom-legge

Capitolo 80: PIVOT / UNPIVOT

Sintassi

- SELECT `<non-pivoted column>` ,
  [prima colonna imperniata] AS `<column name>` ,
  [seconda colonna ruotata] AS `<column name>` ,
  ...
  [ultima colonna imperniata] AS `<column name>`
  A PARTIRE DAL
  (`<SELECT query that produces the data>`)
  AS `<alias for the source query>`
  PERNO
  (
  `<aggregation function>` (`<column being aggregated>`)
  PER
  [`<column that contains the values that will become column headers>`]
  IN ([prima colonna imperniata], [seconda colonna imperniata],
  ... [ultima colonna imperniata])
) AS `<alias for the pivot table>` `<optional ORDER BY clause>` ;

Osservazioni

Usando gli operatori PIVOT e UNPIVOT trasformi una tabella spostando le righe (valori di colonna) di una tabella in colonne e viceversa. Come parte di questa trasformazione, le funzioni di aggregazione possono essere applicate ai valori della tabella.

Examples

Pivot semplice - Colonne statiche

Utilizzando la tabella delle vendite degli articoli dal database di esempio , calcoliamo e mostriamo la quantità totale venduta di ciascun prodotto.

Questo può essere fatto facilmente con un gruppo di, ma assumiamo che dobbiamo "ruotare" la nostra tabella dei risultati in modo che per ogni ID prodotto abbiamo una colonna.

```
SELECT [100], [145]
  FROM (SELECT ItemId , Quantity
          FROM #ItemSalesTable
       ) AS pivotIntermediate
 PIVOT (   SUM(Quantity)
           FOR ItemId IN ([100], [145])
       ) AS pivotTable
```

Poiché le nostre "nuove" colonne sono numeri (nella tabella di origine), abbiamo bisogno di parentesi quadre `[]`

Questo ci darà un risultato simile

100	145
45	18

Semplice PIVOT & UNPIVOT (T-SQL)

Di seguito è riportato un semplice esempio che mostra il prezzo medio dell'articolo di ciascun articolo per giorno della settimana.

In primo luogo, supponiamo di avere un tavolo che tiene registri giornalieri dei prezzi di tutti gli articoli.

```
CREATE TABLE tbl_stock(item NVARCHAR(10), weekday NVARCHAR(10), price INT);

INSERT INTO tbl_stock VALUES
('Item1', 'Mon', 110), ('Item2', 'Mon', 230), ('Item3', 'Mon', 150),
('Item1', 'Tue', 115), ('Item2', 'Tue', 231), ('Item3', 'Tue', 162),
('Item1', 'Wed', 110), ('Item2', 'Wed', 240), ('Item3', 'Wed', 162),
('Item1', 'Thu', 109), ('Item2', 'Thu', 228), ('Item3', 'Thu', 145),
('Item1', 'Fri', 120), ('Item2', 'Fri', 210), ('Item3', 'Fri', 125),
('Item1', 'Mon', 122), ('Item2', 'Mon', 225), ('Item3', 'Mon', 140),
('Item1', 'Tue', 110), ('Item2', 'Tue', 235), ('Item3', 'Tue', 154),
('Item1', 'Wed', 125), ('Item2', 'Wed', 220), ('Item3', 'Wed', 142);
```

La tabella dovrebbe apparire come di seguito:

```
+========+=========+=======+
|   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 |
+--------+---------+-------+
```

Per eseguire l'aggregazione che consiste nel trovare il prezzo medio per articolo per ogni giorno

della settimana, useremo l'operatore relazionale `PIVOT` per ruotare la `weekday` della `weekday` di espressione con valori di tabella in valori di riga aggregati come di seguito:

```
SELECT * FROM tbl_stock
PIVOT (
    AVG(price) FOR weekday IN ([Mon], [Tue], [Wed], [Thu], [Fri])
) pvt;
```

Risultato:

```
+--------+------+------+------+------+------+
|  item  |  Mon |  Tue |  Wed |  Thu |  Fri |
+--------+------+------+------+------+------+
|  Item1 |  116 |  112 |  117 |  109 |  120 |
|  Item2 |  227 |  233 |  230 |  228 |  210 |
|  Item3 |  145 |  158 |  152 |  145 |  125 |
+--------+------+------+------+------+------+
```

Infine, per eseguire l'operazione inversa di `PIVOT` , possiamo utilizzare l'operatore relazionale `UNPIVOT` per ruotare le colonne in righe come di seguito:

```
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;
```

Risultato:

```
+=======+========+=========+
|  item |  price | weekday |
+=======+========+=========+
| Item1 |    116 |     Mon |
+-------+--------+---------+
| Item1 |    112 |     Tue |
+-------+--------+---------+
| Item1 |    117 |     Wed |
+-------+--------+---------+
| Item1 |    109 |     Thu |
+-------+--------+---------+
| Item1 |    120 |     Fri |
+-------+--------+---------+
| Item2 |    227 |     Mon |
+-------+--------+---------+
| Item2 |    233 |     Tue |
+-------+--------+---------+
| Item2 |    230 |     Wed |
+-------+--------+---------+
| Item2 |    228 |     Thu |
+-------+--------+---------+
| Item2 |    210 |     Fri |
+-------+--------+---------+
| Item3 |    145 |     Mon |
+-------+--------+---------+
```

```
| Item3 |    158 |     Tue |
+-------+--------+---------+
| Item3 |    152 |     Wed |
+-------+--------+---------+
| Item3 |    145 |     Thu |
+-------+--------+---------+
| Item3 |    125 |     Fri |
+-------+--------+---------+
```

PIVOT dinamico

Un problema con la query `PIVOT` è che devi specificare tutti i valori all'interno della selezione `IN` se vuoi vederli come colonne. Un modo rapido per aggirare questo problema è quello di creare una selezione IN dinamica che renda dinamico il tuo `PIVOT` .

Per la dimostrazione useremo una tabella `Books` nel database di un `Bookstore` `Books` . Partiamo dal presupposto che la tabella è abbastanza de-normalizzata e ha colonne seguenti

```
Table: Books
-----------------------------
BookId (Primary Key Column)
Name
Language
NumberOfPages
EditionNumber
YearOfPrint
YearBoughtIntoStore
ISBN
AuthorName
Price
NumberOfUnitsSold
```

Lo script di creazione per la tabella sarà simile a:

```
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
```

Ora se abbiamo bisogno di interrogare il database e calcolare il numero di libri in inglese, russo, tedesco, hindi, lingue latine comprate nel bookstore ogni anno e presentare il nostro output in un piccolo formato di report, possiamo usare la query PIVOT come questa

```
SELECT * FROM
  (
   SELECT YearBoughtIntoStore AS [Year Bought],[Language], NumberOfBooks
   FROM BookList
  ) sourceData
 PIVOT
  (
  SUM(NumberOfBooks)
  FOR [Language] IN (English, Russian, German, Hindi, Latin)
  ) pivotrReport
```

Caso speciale è quando non abbiamo un elenco completo delle lingue, quindi utilizzeremo SQL dinamico come di seguito

```
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)
```

Leggi PIVOT / UNPIVOT online: https://riptutorial.com/it/sql-server/topic/591/pivot---unpivot

Capitolo 81: Pivot SQL dinamico

introduzione

Questo argomento illustra come eseguire un pivot dinamico in SQL Server.

Examples

Pivot dinamico dinamico di base

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

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


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

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

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


exec sp_executeSql @query
```

Leggi Pivot SQL dinamico online: https://riptutorial.com/it/sql-server/topic/10751/pivot-sql-dinamico

Capitolo 82: Privilegi o permessi

Examples

Regole semplici

Concessione dell'autorizzazione per creare tabelle

```
USE AdventureWorks;
GRANT CREATE TABLE TO MelanieK;
GO
```

Concessione dell'autorizzazione SHOWPLAN a un ruolo dell'applicazione

```
USE AdventureWorks2012;
GRANT SHOWPLAN TO AuditMonitor;
GO
```

Concessione CREATE VIEW con GRANT OPTION

```
USE AdventureWorks2012;
GRANT CREATE VIEW TO CarmineEs WITH GRANT OPTION;
GO
```

Concessione di tutti i diritti a un utente su un database specifico

```
use YourDatabase
go
exec sp_addrolemember 'db_owner', 'UserName'
go
```

Leggi Privilegi o permessi online: https://riptutorial.com/it/sql-server/topic/5333/privilegi-o-permessi

Capitolo 83: Procedura di archiviazione

introduzione

In SQL Server, una procedura è un programma memorizzato in cui è possibile passare i parametri. Non restituisce un valore come fa una funzione. Tuttavia, può restituire uno stato di successo / errore alla procedura che lo ha chiamato.

Sintassi

- CREA {PROCEDURA | PROC} [nome_schema.] Nome_procedura
- [@parameter [tipo_schema_nome.] tipo di dati
- [VARYING] [= valore predefinito] [OUT | OUTPUT | SOLA LETTURA]
- , @parameter [tipo_schema_nome.] tipo di dati
- [VARYING] [= valore predefinito] [OUT | OUTPUT | SOLA LETTURA]]
- [CON {ENCRYPTION | RECOMPILE | ESEGUI COME clausola}]
- [PER RISPOSTA]
- COME
- INIZIO
- [Declaration_section]
- executable_section
- FINE;

Examples

Creazione ed esecuzione di una stored procedure di base

Utilizzando la tabella `Authors` nel database della libreria

```
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
```

È possibile eseguire una procedura con alcune sintassi diverse. Innanzitutto, è possibile utilizzare `EXECUTE` o `EXEC`

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

Inoltre, è possibile omettere il comando EXEC. Inoltre, non è necessario specificare quale parametro si sta passando, come si passa in tutti i parametri.

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

Quando si desidera specificare i parametri di input in un ordine diverso da come sono dichiarati nella procedura, è possibile specificare il nome del parametro e assegnare i valori. Per esempio

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

    SELECT
        Param1 = @Param1,
        Param2 = @Param2

END
```

l'ordine normale per eseguire questa procedura è specificare il valore per @ Param1 prima e poi @ Param2 secondo. Quindi assomiglierà a qualcosa del genere

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

Ma è anche possibile che tu possa usare quanto segue

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

in questo, si specifica il valore per @ param2 first e @ Param1 second. Il che significa che non devi mantenere lo stesso ordine che è stato dichiarato nella procedura, ma puoi avere qualsiasi ordine come desideri. ma dovrai specificare a quale parametro stai impostando il valore

Accedi alla stored procedure da qualsiasi database

Inoltre, è possibile creare una procedura con un prefisso `sp_` questi procuedres, come tutte le stored procedure di sistema, possono essere eseguite senza specificare il database a causa del comportamento predefinito di SQL Server. Quando si esegue una stored procedure che inizia con "sp_", SQL Server cerca prima la procedura nel database master. Se la procedura non viene trovata in master, viene visualizzata nel database attivo. Se si dispone di una stored procedure a cui si desidera accedere da tutti i database, crearla in master e utilizzare un nome che includa il prefisso "sp_".

```
Use Master

CREATE PROCEDURE sp_GetName
(
    @input_id INT = NULL,      --Input parameter,  id of the person, NULL default
    @name VARCHAR(128) = NULL  --Input parameter, name of the person, NULL default
```

```
)
AS
BEGIN
    SELECT Name + ' is from ' + Country
    FROM Authors
    WHERE Id = @input_id OR Name = @name
END
GO
```

PROCEDURA MEMORIZZATA con parametri OUT

Le stored procedure possono restituire valori utilizzando la parola chiave `OUTPUT` nel relativo elenco di parametri.

Creazione di una stored procedure con un singolo parametro out

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

Esecuzione della stored procedure

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

Creazione di una stored procedure con più parametri out

```
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
```

Esecuzione della stored procedure

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

Procedura memorizzata con If ... Else e Insert Into operation

Crea tabella di esempio `Employee` :

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

Crea una stored procedure che controlla se i valori passati nella stored procedure non sono nulli o non vuoti e eseguono operazioni di inserimento nella tabella Employee.

```
CREATE PROCEDURE spSetEmployeeDetails
(
    @ID int,
    @Name VARCHAR(25),
    @Gender VARCHAR(6),
    @DeptId INT
)
AS
BEGIN
    IF (
        (@ID IS NOT NULL AND LEN(@ID) !=0)
        AND (@Name IS NOT NULL AND LEN(@Name) !=0)
        AND (@Gender IS NOT NULL AND LEN(@Gender) !=0)
        AND (@DeptId IS NOT NULL AND LEN(@DeptId) !=0)
    )
    BEGIN
        INSERT INTO Employee
        (
            Id,
            EmpName,
            EmpGender,
            EmpDeptId
        )
        VALUES
        (
```

```
            @ID,
            @Name,
            @Gender,
            @DeptId
        )
    END
ELSE
    PRINT 'Incorrect Parameters'
END
GO
```

Esegui la stored procedure

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

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

SQL dinamico in stored procedure

Dynamic SQL ci consente di generare ed eseguire istruzioni SQL in fase di esecuzione. SQL dinamico è necessario quando le nostre istruzioni SQL contengono identificatori che possono cambiare in tempi di compilazione diversi.

Semplice esempio di SQL dinamico:

```
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
```

Nella query sql sopra, possiamo vedere che possiamo usare sopra la query definendo i valori in `@table_name, @col_name, and @col_value` in fase di esecuzione. La query viene generata in fase di esecuzione ed eseguita. Questa è la tecnica in cui possiamo creare interi script come stringa in una variabile ed eseguirla. Possiamo creare query più complesse utilizzando il concetto dinamico di SQL e concatenazione. Questo concetto è molto potente quando si desidera creare uno script che può essere utilizzato in diverse condizioni.

Esecuzione della stored procedure

```
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
```

Tabella che ho usato

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

Produzione

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

Semplice loop

Prima consente di recuperare alcuni dati in una tabella temporanea denominata `#systables` e annunci un numero di riga incrementale in modo che possiamo interrogare un record alla volta

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

Quindi dichiariamo alcune variabili per controllare il ciclo e memorizzare il nome della tabella in questo esempio

```
declare
    @rn int = 1,
    @maxRn int = (
                    select
                        max(rn)
```

```
                    from
                        #systables as s
                    )
declare    @tablename sys name
```

Ora possiamo effettuare il loop usando un po 'di tempo. `@rn select` , ma questa potrebbe anche essere un'istruzione separata per ex `set @rn = @rn + 1` che dipenderà dalle vostre esigenze. Usiamo anche il valore di `@rn` prima che venga incrementato per selezionare un singolo record da `#systables` . Infine stampiamo il nome della tabella.

```
while @rn <= @maxRn
    begin

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

        print @tablename
    end
```

Semplice loop

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

    RETURN;
END
GO
```

Leggi Procedura di archiviazione online: https://riptutorial.com/it/sql-server/topic/3213/procedura-di-archiviazione

Capitolo 84: PROVA A PRENDERE

Osservazioni

TRY / CATCH è un costrutto linguistico specifico per T-SQL di MS SQL Server.

Permette la gestione degli errori all'interno di T-SQL, simile a quella vista nel codice .NET.

Examples

Transazione in TRY / CATCH

Ciò ripristinerà entrambi gli inserimenti a causa di un datetime non valido:

```
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
```

Questo imposterà entrambi gli inserti:

```
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
```

Aumentare gli errori nel blocco try-catch

La funzione RAISERROR genera un errore nel blocco TRY CATCH:

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

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

RAISERROR con secondo parametro maggiore di 10 (11 in questo esempio) interrompe l'esecuzione in TRY BLOCK e genera un errore che verrà gestito nel blocco CATCH. È possibile accedere al messaggio di errore utilizzando la funzione ERROR_MESSAGE (). L'output di questo campione è:

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

Aumentare i messaggi di informazione nel try catch block

RAISERROR con severità (secondo parametro) minore o uguale a 10 non genererà eccezioni.

```
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
```

Dopo l'istruzione RAISERROR, verrà eseguita la terza istruzione e il blocco CATCH non verrà richiamato. Il risultato dell'esecuzione è:

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

Rilanciare l'eccezione generata da RAISERROR

È possibile ripetere l'errore che si cattura nel blocco CATCH utilizzando l'istruzione TRHOW:

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

Si noti che in questo caso si genera un errore con argomenti formattati (quarto e quinto parametro). Questo potrebbe essere utile se vuoi aggiungere più informazioni nel messaggio. Il risultato dell'esecuzione è:

```
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'
```

Eccezione di lancio nei blocchi TRY / CATCH

Puoi lanciare un'eccezione nel try catch block:

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

Eccezione con essere gestita nel blocco CATCH e quindi re-generata con il comando LANCIO senza parametri.

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

THROW è simile a RAISERROR con le seguenti differenze:

- La raccomandazione è che le nuove applicazioni debbano utilizzare THROW anziché RASIERROR.
- THROW può usare qualsiasi numero come primo argomento (numero di errore), RAISERROR può usare solo id nella vista sys.messages
- THROW ha gravità 16 (non può essere modificato)
- THROW non può formattare argomenti come RAISERROR. Usa la funzione FORMATMESSAGE come argomento di RAISERROR se ti serve questa funzione.

Leggi PROVA A PRENDERE online: https://riptutorial.com/it/sql-server/topic/5189/prova-a-prendere

Capitolo 85: Query con dati JSON

Examples

Utilizzo dei valori da JSON nella query

La funzione JSON_VALUE consente di acquisire dati dal testo JSON sul percorso specificato come secondo argomento e utilizzare questo valore in qualsiasi parte della query di selezione:

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

Utilizzo dei valori JSON nei report

Una volta estratti i valori JSON dal testo JSON, è possibile utilizzarli in qualsiasi parte della query. È possibile creare un tipo di rapporto sui dati JSON con aggregazioni di raggruppamento, ecc.

```
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
```

Filtraggio del testo JSON non valido dai risultati della query

Se alcuni testi JSON potrebbero non essere formattati correttamente, è possibile rimuovere tali voci dalla query utilizzando la funzione ISJSON.

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

Aggiorna valore nella colonna JSON

La funzione JSON_MODIFY può essere utilizzata per aggiornare il valore su qualche percorso. È possibile utilizzare questa funzione per modificare il valore originale della cella JSON nell'istruzione UPDATE:

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

La funzione JSON_MODIFY aggiornerà o creerà il tasto prezzo (se non esiste). Se il nuovo valore è NULL, la chiave verrà rimossa. La funzione JSON_MODIFY tratterà il nuovo valore come stringa (sfugge ai caratteri speciali, lo avvolge con virgolette doppie per creare una stringa JSON

corretta). Se il tuo nuovo valore è frammento JSON, dovresti eseguirlo con la funzione JSON_QUERY:

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

La funzione JSON_QUERY senza secondo parametro si comporta come un "cast su JSON". Poiché il risultato di JSON_QUERY è un frammento JSON valido (oggetto o array), JSON_MODIFY non sfuggirà a questo valore quando modifica l'input JSON.

Aggiungi un nuovo valore nell'array JSON

La funzione JSON_MODIFY può essere utilizzata per aggiungere un nuovo valore ad un array all'interno di JSON:

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

Il nuovo valore verrà aggiunto alla fine dell'array o verrà creato un nuovo array con valore ["vendite"]. La funzione JSON_MODIFY tratterà il nuovo valore come stringa (sfugge ai caratteri speciali, lo avvolge con virgolette doppie per creare una stringa JSON corretta). Se il tuo nuovo valore è frammento JSON, dovresti eseguirlo con la funzione JSON_QUERY:

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

La funzione JSON_QUERY senza secondo parametro si comporta come un "cast su JSON". Poiché il risultato di JSON_QUERY è un frammento JSON valido (oggetto o array), JSON_MODIFY non sfuggirà a questo valore quando modifica l'input JSON.

Tavolo JOIN con collezione JSON interna

Se si dispone di una "tabella figlio" formattata come raccolta JSON e memorizzata in-row come colonna JSON, è possibile decomprimere questa raccolta, trasformarla in tabella e aggiungerla alla riga padre. Invece dell'operatore JOIN standard, dovresti usare CROSS APPLY. In questo esempio, le parti del prodotto sono formattate come raccolta di oggetti JSON e memorizzate nella colonna Dati:

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

Il risultato della query è equivalente al join tra le tabelle Prodotto e Parte.

Ricerca di righe che contengono valore nell'array JSON

In questo esempio, l'array di tag può contenere varie parole chiave come ["promo", "vendite"], quindi è possibile aprire questo array e filtrare i valori:

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

OPENJSON aprirà la raccolta interna dei tag e la restituirà come tabella. Quindi possiamo filtrare i risultati per un certo valore nella tabella.

Leggi Query con dati JSON online: https://riptutorial.com/it/sql-server/topic/5028/query-con-dati-json

Capitolo 86: RAGGRUPPA PER

Examples

Raggruppamento semplice

Tabella degli ordini

Identificativo del cliente	Codice prodotto	Quantità	Prezzo
1	2	5	100
1	3	2	200
1	4	1	500
2	1	4	50
3	5	6	700

Quando si raggruppa per una colonna specifica, vengono restituiti solo i valori univoci di questa colonna.

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

Valore di ritorno:

identificativo del cliente
1
2
3

Le funzioni di aggregazione come `count()` applicano a ciascun gruppo e non alla tabella completa:

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

Valore di ritorno:

identificativo del cliente	numberOfProducts	prezzo totale
1	3	800
2	1	50
3	1	700

GROUP BY multiple columns

Si potrebbe desiderare GROUP BY più di una colonna

```
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
```

raggrupperà per età e nome e produrrà:

Età	Nome	contare
18	luke	3
21	luke	2
18	opaco	1
21	opaco	2

Raggruppa per più tabelle, più colonne

Raggruppa per viene spesso utilizzato con la dichiarazione di join. Supponiamo di avere due tavoli. Il primo è il tavolo degli studenti:

Id	Nome e cognome	Età
1	Matt Jones	20
2	Frank Blue	21

Id	Nome e cognome	Età
3	Anthony Angel	18

La seconda tabella è la tabella di discussione che ogni studente può prendere:

icLsoggetto	Soggetto
1	Matematica
2	PE
3	Fisica

E poiché uno studente può frequentare molte materie e una materia può essere frequentata da molti studenti (quindi N: N relazione), abbiamo bisogno di avere una terza tabella "delimitata". Chiamiamo la tabella Students_subjects:

icLsoggetto	student_id
1	1
2	2
2	1
3	2
1	3
1	1

Ora diciamo che vogliamo sapere il numero di soggetti che ogni studente sta frequentando. Qui l'istruzione `GROUP BY` autonoma non è sufficiente in quanto le informazioni non sono disponibili tramite una singola tabella. Pertanto, è necessario utilizzare `GROUP BY` con l'istruzione `JOIN` :

```
Select Students.FullName, COUNT(Subject Id) as SubjectNumber FROM Students_Subjects
LEFT JOIN Students
ON Students_Subjects.Student_id = Students.Id
GROUP BY Students.FullName
```

Il risultato della query fornita è il seguente:

Nome e cognome	SubjectNumber
Matt Jones	3
Frank Blue	2

Nome e cognome	SubjectNumber
Anthony Angel	1

Per un esempio ancora più complesso di utilizzo di GROUP BY, diciamo che lo studente potrebbe essere in grado di assegnare lo stesso oggetto al suo nome più di una volta (come mostrato nella tabella Students_Subjects). In questo scenario potremmo essere in grado di contare il numero di volte in cui ogni argomento è stato assegnato a uno studente mediante il raggruppamento di più di una colonna:

```
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
```

Questa query fornisce il seguente risultato:

Nome e cognome	Soggetto	SubjectNumber
Matt Jones	Matematica	2
Matt Jones	PE	1
Frank Blue	PE	1
Frank Blue	Fisica	1
Anthony Angel	Matematica	1

VISTA

Poiché la clausola `WHERE` viene valutata prima di `GROUP BY` , non è possibile utilizzare `WHERE` per ridurre i risultati del raggruppamento (in genere una funzione di aggregazione, ad esempio `COUNT(*)`). Per soddisfare questa esigenza, è possibile utilizzare la clausola `HAVING` .

Ad esempio, utilizzando i seguenti dati:

```
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' )
```

Se vogliamo ottenere il numero di ordini che ogni persona ha piazzato, useremmo

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

e prendi

Nome	Ordini
opaco	2
John	5
Luca	4

Tuttavia, se vogliamo limitarlo a individui che hanno effettuato più di due ordini, possiamo aggiungere una clausola `HAVING` .

```
SELECT Name, COUNT(*) AS 'Orders'
FROM @orders
GROUP BY Name
HAVING COUNT(*) > 2
```

cederà

Nome	Ordini
John	5
Luca	4

Si noti che, proprio come `GROUP BY` , le colonne inserite in `HAVING` devono corrispondere esattamente alle loro controparti `SELECT` . Se nell'esempio precedente avessimo invece detto

```
SELECT Name, COUNT(DISTINCT OrderID)
```

la nostra clausola `HAVING` avrebbe dovuto dire

```
HAVING COUNT(DISTINCT OrderID) > 2
```

GROUP BY con ROLLUP e CUBE

L'operatore ROLLUP è utile per generare report che contengono totali parziali e totali.

- CUBE genera un set di risultati che mostra gli aggregati per tutte le combinazioni di valori nelle colonne selezionate.
- ROLLUP genera un set di risultati che mostra gli aggregati per una gerarchia di valori nelle colonne selezionate.

Articolo	Colore	Quantità
tavolo	Blu	124
tavolo	Rosso	223
Sedia	Blu	101
Sedia	Rosso	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 righe (s) interessate)

Se la parola chiave ROLLUP nella query viene modificata in CUBE, il set di risultati CUBE è lo stesso, tranne che queste due righe aggiuntive vengono restituite alla fine:

```
ALL                  Blue                 225.00
ALL                  Red                  433.00
```

https://technet.microsoft.com/en-us/library/ms189305(v=sql.90).aspx

Leggi RAGGRUPPA PER online: https://riptutorial.com/it/sql-server/topic/3231/raggruppa-per

Capitolo 87: Recupera informazioni sul database

Osservazioni

Come con altri sistemi di database relazionali, SQL Server espone i metadati relativi ai database.

Questo viene fornito dallo standard `INFORMATION_SCHEMA` dello standard ISO o dalle viste del catalogo `sys` SQL Server.

Examples

Contare il numero di tabelle in un database

Questa query restituirà il numero di tabelle nel database specificato.

```
USE YourDatabaseName
SELECT COUNT(*) from INFORMATION_SCHEMA.TABLES
WHERE TABLE_TYPE = 'BASE TABLE'
```

Di seguito è riportato un altro modo per eseguire tutte le tabelle utente con SQL Server 2008+. Il riferimento è qui .

```
SELECT COUNT(*) FROM sys.tables
```

Recupera un elenco di tutte le stored procedure

Le seguenti query restituiranno un elenco di tutte le stored procedure nel database, con informazioni di base su ciascuna stored procedure:

SQL Server 2005

```
SELECT *
FROM INFORMATION_SCHEMA.ROUTINES
WHERE ROUTINE_TYPE = 'PROCEDURE'
```

Le `ROUTINE_NAME` , `ROUTINE_SCHEMA` e `ROUTINE_DEFINITION` sono generalmente le più utili.

SQL Server 2005

```
SELECT *
FROM sys.objects
WHERE type = 'P'
```

SQL Server 2005

```
SELECT *
FROM sys.procedures
```

Si noti che questa versione ha un vantaggio sulla selezione da sys.objects poiché include le colonne aggiuntive `is_auto_executed` , `is_execution_replicated` , `is_repl_serializable` e `skips_repl_constraints` .

SQL Server 2005

```
SELECT *
FROM sysobjects
WHERE type = 'P'
```

Si noti che l'output contiene molte colonne che non saranno mai correlate a una stored procedure.

La prossima serie di query restituirà tutte le stored procedure nel database che includono la stringa "SearchTerm":

SQL Server 2005

```
SELECT o.name
FROM syscomments c
INNER JOIN sysobjects o
    ON c.id=o.id
WHERE o.xtype = 'P'
    AND c.TEXT LIKE '%SearchTerm%'
```

SQL Server 2005

```
SELECT p.name
FROM sys.sql_modules AS m
INNER JOIN sys.procedures AS p
    ON m.object_id = p.object_id
WHERE definition LIKE '%SearchTerm%'
```

Ottieni l'elenco di tutti i database su un server

Metodo 1: la query sottostante sarà applicabile per la versione di SQL Server 2000+ (contiene 12 colonne)

```
SELECT * FROM dbo.sysdatabases
```

Metodo 2: Sotto interrogare estrarre informazioni sui database con maggiori informazioni (es: Stato, Isolamento, modello di recupero ecc.)

Nota: questa è una vista del catalogo e sarà disponibile versioni di SQL Server 2005+

```
SELECT * FROM sys.databases
```

Metodo 3: per visualizzare solo i nomi dei database è possibile utilizzare sp_MSForEachDB non documentato

```
EXEC sp_MSForEachDB 'SELECT ''?'' AS DatabaseName'
```

Metodo 4: Sotto SP vi aiuterà a fornire le dimensioni del database insieme al nome del database, proprietario, stato ecc. Sul server

```
EXEC sp_helpdb
```

Metodo 5 Allo stesso modo, la procedura memorizzata sotto fornirà il nome del database, la dimensione del database e le Note

```
EXEC sp_databases
```

File di database

Visualizza tutti i file di dati per tutti i database con dimensioni e informazioni di crescita

```
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
```

Recupera le opzioni del database

La seguente query restituisce le opzioni e i metadati del database:

```
select * from sys.databases WHERE name = 'MyDatabaseName';
```

Mostra la dimensione di tutte le tabelle nel database corrente

```
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
```

Determina il percorso di autorizzazione di un accesso di Windows

Questo mostrerà il tipo di utente e il percorso di autorizzazione (da quale gruppo di finestre l'utente ottiene le sue autorizzazioni).

```
xp_logininfo 'DOMAIN\user'
```

Recupera tabelle contenenti colonne conosciute

Questa query restituirà tutte le `COLUMNS` e le `TABLES` associate per un determinato nome di colonna. È progettato per mostrare quali tabelle (sconosciute) contengono una colonna specificata (nota)

```
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%'
```

Verifica se vengono utilizzate funzionalità specifiche dell'organizzazione

A volte è utile verificare che il tuo lavoro sulla Developer Edition non abbia introdotto una dipendenza da nessuna funzionalità limitata all'edizione Enterprise.

Puoi farlo usando la vista di sistema `sys.dm_db_persisted_sku_features` , in questo modo:

```
SELECT * FROM sys.dm_db_persisted_sku_features
```

Contro il database stesso.

Questo elencherà le funzioni in uso, se presenti.

Cerca e restituisci tutte le tabelle e le colonne contenenti un valore di colonna specificato

Questo script, da qui e qui , restituirà tutte le tabelle e le colonne in cui esiste un valore specificato. Questo è potente per scoprire dove un determinato valore si trova in un database. Può essere tassativo, pertanto è consigliabile eseguirlo prima in un ambiente di backup / test.

```
DECLARE @SearchStr nvarchar(100)
SET @SearchStr = '## YOUR STRING HERE ##'
```

```
    -- Copyright © 2002 Narayana Vyas Kondreddi. All rights reserved.
    -- Purpose: To search all columns of all tables for a given search string
    -- Written by: Narayana Vyas Kondreddi
    -- Site: http://vyaskn.tripod.com
    -- Updated and tested by Tim Gaunt
    -- http://www.thesitedoctor.co.uk
    --
http://blogs.thesitedoctor.co.uk/tim/2010/02/19/Search+Every+Table+And+Field+In+A+SQL+Server+Database+U

    -- Tested on: SQL Server 7.0, SQL Server 2000, SQL Server 2005 and SQL Server 2010
    -- Date modified: 03rd March 2011 19:00 GMT
    CREATE TABLE #Results (ColumnName nvarchar(370), ColumnValue nvarchar(3630))

    SET NOCOUNT ON

    DECLARE @TableName nvarchar(256), @ColumnName nvarchar(128), @SearchStr2 nvarchar(110)
    SET  @TableName = ''
    SET @SearchStr2 = QUOTENAME('%' + @SearchStr + '%','''')

    WHILE @TableName IS NOT NULL

    BEGIN
        SET @ColumnName = ''
        SET @TableName =
        (
            SELECT MIN(QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME))
            FROM     INFORMATION_SCHEMA.TABLES
            WHERE         TABLE_TYPE = 'BASE TABLE'
                AND    QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME) > @TableName
                AND    OBJECTPROPERTY(
                        OBJECT_ID(
                            QUOTENAME(TABLE_SCHEMA) + '.' + QUOTENAME(TABLE_NAME)
                             ), 'IsMSShipped'
                               ) = 0
        )

        WHILE (@TableName IS NOT NULL) AND (@ColumnName IS NOT NULL)

        BEGIN
            SET @ColumnName =
            (
                SELECT MIN(QUOTENAME(COLUMN_NAME))
                FROM     INFORMATION_SCHEMA.COLUMNS
                WHERE         TABLE_SCHEMA    = PARSENAME(@TableName, 2)
                    AND    TABLE_NAME    = PARSENAME(@TableName, 1)
                    AND    DATA_TYPE IN ('char', 'varchar', 'nchar', 'nvarchar', 'int',
'decimal')
                    AND    QUOTENAME(COLUMN_NAME) > @ColumnName
            )

            IF @ColumnName IS NOT NULL

            BEGIN
                INSERT INTO #Results
                EXEC
                (
                    'SELECT ''' + @TableName + '.' + @ColumnName + ''', LEFT(' + @ColumnName +
', 3630) FROM ' + @TableName + ' (NOLOCK) ' +
                    ' WHERE ' + @ColumnName + ' LIKE ' + @SearchStr2
```

```
                )
            END
        END
    END

    SELECT ColumnName, ColumnValue FROM #Results

DROP TABLE #Results
- See more at: http://thesitedoctor.co.uk/blog/search-every-table-and-field-in-a-sql-server-database-updated#sthash.bBEqfJVZ.dpuf
```

Ottieni tutti gli schemi, tabelle, colonne e indici

```
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;
```

Restituisce un elenco di lavori di SQL Agent, con informazioni sulla pianificazione

```
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]
```

Recupera le informazioni sulle operazioni di backup e ripristino

Per ottenere l'elenco di tutte le operazioni di backup eseguite sull'istanza del database corrente:

```
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
```

Per ottenere l'elenco di tutte le operazioni di ripristino eseguite sull'istanza del database corrente:

```
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
```

Trova ogni menzione di un campo nel database

```
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
```

Troverà menzioni di `myField` in SProcs, Views, ecc.

Leggi Recupera informazioni sul database online: https://riptutorial.com/it/sql-server/topic/697/recupera-informazioni-sul-database

Capitolo 88: Recupera informazioni sulla tua istanza

Examples

Recupera server locali e remoti

Per recuperare un elenco di tutti i server registrati nell'istanza:

```
EXEC sp_helpserver;
```

Ottieni informazioni sulle sessioni correnti e sulle esecuzioni di query

```
sp_who2
```

Questa procedura può essere utilizzata per trovare informazioni sulle sessioni correnti del server SQL. Poiché si tratta di una procedura, è spesso utile memorizzare i risultati in una tabella temporanea o in una variabile di tabella in modo da poter ordinare, filtrare e trasformare i risultati secondo necessità.

Il seguente può essere usato per una versione interrogabile di `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
```

Esempi:

```
-- 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
```

Recupera edizione e versione di istanza

```
SELECT    SERVERPROPERTY('ProductVersion') AS ProductVersion,
          SERVERPROPERTY('ProductLevel') AS ProductLevel,
          SERVERPROPERTY('Edition') AS Edition,
          SERVERPROPERTY('EngineEdition') AS EngineEdition;
```

Recupera il tempo di istanza in giorni

```
SELECT  DATEDIFF(DAY, login_time, getdate()) UpDays
FROM    master..sysprocesses
WHERE   spid = 1
```

Informazioni sulla versione di SQL Server

Per scoprire l'edizione di SQL Server, il livello del prodotto e il numero di versione, nonché il nome del computer host e il tipo di server:

```
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;
```

Informazioni generali su database, tabelle, procedure memorizzate e su come cercarle.

Query per cercare gli ultimi sp eseguiti 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
```

Query per cercare tra le stored procedure

```
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
```

Query per trovare la colonna da tutte le tabelle del database

```
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;
```

Interrogare su per verificare i dettagli del ripristino

```
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
```

Interrogare per trovare il registro

```
select top 100 * from databaselog
Order by Posttime desc
```

Interrogare per controllare i dettagli di Sps

```
SELECT name, create_date, modify_date
FROM sys.objects
WHERE type = 'P'
Order by modify_date desc
```

Leggi Recupera informazioni sulla tua istanza online: https://riptutorial.com/it/sql-server/topic/2029/recupera-informazioni-sulla-tua-istanza

Capitolo 89: Resource Governor

Osservazioni

Resource Governor in SQL Server è una funzionalità che consente di gestire l'utilizzo delle risorse da parte di diverse applicazioni e utenti. Calcia in tempo reale impostando i limiti della CPU e della memoria. Aiuterà a evitare che un processo pesante mangi tutte le risorse di sistema, mentre per esempio sono in attesa attività più piccole.

Disponibile solo in Enterprise Edition

Examples

Leggendo le statistiche

```
select *
from sys.dm_resource_governor_workload_groups

select *
from sys.dm_resource_governor_resource_pools
```

Creare un pool per le query ad hoc

Prima crea un pool di risorse oltre a quello predefinito

```
CREATE RESOURCE POOL [PoolAdhoc] WITH(min_cpu_percent=0,
        max_cpu_percent=50,
        min_memory_percent=0,
        max_memory_percent=50)
GO
```

Creare il gruppo worload per il pool

```
CREATE WORKLOAD GROUP [AdhocMedium] WITH(importance=Medium) USING [PoolAdhoc]
```

Creare la funzione che contiene la logica per il governatore risorse e allegarla

```
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
```

Leggi Resource Governor online: https://riptutorial.com/it/sql-server/topic/4146/resource-governor

Capitolo 90: schemi

Examples

Creazione di uno schema

```
CREATE SCHEMA dvr AUTHORIZATION Owner
    CREATE TABLE sat_Sales (source int, cost int, partid int)
    GRANT SELECT ON SCHEMA :: dvr TO  User1
    DENY SELECT ON SCHEMA :: dvr to User 2
GO
```

Alter Schema

```
ALTER SCHEMA dvr
    TRANSFER dbo.tbl_Staging;
GO
```

Questo trasferirebbe la tabella tbl_Staging dallo schema dbo allo schema dvr

Schemi che cadono

```
DROP SCHEMA dvr
```

Scopo

Lo schema fa riferimento a una tabella di database specifica e al modo in cui sono correlate tra loro. Fornisce un modello organizzativo di come viene costruito il database. Ulteriori vantaggi nell'implementazione degli schemi di database è che gli schemi possono essere utilizzati come metodo per limitare / garantire l'accesso a tabelle specifiche all'interno di un database.

Leggi schemi online: https://riptutorial.com/it/sql-server/topic/5806/schemi

Capitolo 91: SCOPE_IDENTITY ()

Sintassi

- SELEZIONA SCOPE_IDENTITY ();
- SELEZIONA SCOPE_IDENTITY () AS [SCOPE_IDENTITY];
- SCOPE_IDENTITY ()

Examples

Introduzione con un semplice esempio

SCOPE_IDENTITY () restituisce l'ultimo valore di identità inserito in una colonna Identity nello stesso ambito. Un ambito è un modulo: una stored procedure, trigger, funzione o batch. Pertanto, due istruzioni sono nello stesso ambito se si trovano nella stessa stored procedure, funzione o batch.

INSERT INTO ([column1], [column2]) VALUES (8,9);
PARTIRE
SELEZIONA SCOPE_IDENTITY () AS [SCOPE_IDENTITY];
PARTIRE

Leggi SCOPE_IDENTITY () online: https://riptutorial.com/it/sql-server/topic/5326/scope-identity---

Capitolo 92: SE ALTRO

Examples

Istruzione IF singola

Come la maggior parte degli altri linguaggi di programmazione, T-SQL supporta anche le istruzioni IF..ELSE.

Ad esempio nell'esempio seguente `1 = 1` è l'espressione, che `BEGIN..END` True e il controllo immette nel blocco `BEGIN..END` e l'istruzione Print stampa la stringa `'One is equal to One'`

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

Più dichiarazioni IF

Possiamo utilizzare più istruzioni IF per controllare più espressioni totalmente indipendenti l'una dall'altra.

Nell'esempio seguente, viene valutata l'espressione di ogni istruzione `IF` e, se è vero, il codice all'interno del blocco `BEGIN...END` viene eseguito. In questo particolare esempio, la prima e la terza espressione sono vere e solo le istruzioni di stampa verranno eseguite.

```
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
```

Singola dichiarazione IF..ELSE

In una singola istruzione `IF..ELSE` , se l'espressione `IF..ELSE` True `BEGIN..END IF` il controllo entra nel primo blocco `BEGIN..END` e solo il codice all'interno di quel blocco viene eseguito, il blocco Else viene semplicemente ignorato.

D'altra parte se l'espressione `ELSE BEGIN..END False` il blocco `ELSE BEGIN..END` viene eseguito e il controllo non inserisce mai il primo blocco `BEGIN..END` .

Nell'esempio seguente l'espressione valuterà su false e verrà eseguito il blocco Else stampando la stringa `'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
```

Più IF ... ELSE con dichiarazioni ELSE finali

Se abbiamo più istruzioni `IF...ELSE IF` ma vogliamo anche eseguire qualche parte di codice se nessuna delle espressioni viene valutata su True, allora possiamo semplicemente aggiungere un blocco `ELSE` finale che viene eseguito solo se nessuna delle `IF` oppure le espressioni `ELSE IF` vengono valutate su true.

Nell'esempio seguente nessuna espressione `IF` o `ELSE IF` è True, quindi viene eseguito solo il blocco `ELSE` e viene stampato `'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
```

Più istruzioni IF ... ELSE

Più spesso che non abbiamo bisogno di controllare più espressioni e intraprendere azioni specifiche basate su quelle espressioni. Questa situazione viene gestita utilizzando più istruzioni `IF...ELSE IF` .

In questo esempio tutte le espressioni sono valutate dall'alto verso il basso. Non appena un'espressione restituisce true, viene eseguito il codice all'interno di quel blocco. Se nessuna espressione viene valutata come vera, non viene eseguito nulla.

```
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
```

Leggi SE ALTRO online: https://riptutorial.com/it/sql-server/topic/5186/se-altro

Capitolo 93: SELECT statement

introduzione

In SQL, le istruzioni `SELECT` restituiscono insiemi di risultati da raccolte di dati come tabelle o viste. `SELECT` istruzioni `SELECT` possono essere utilizzate con varie altre clausole come `WHERE` , `GROUP BY` o `ORDER BY` per perfezionare ulteriormente i risultati desiderati.

Examples

SELEZIONA di base dalla tabella

Seleziona tutte le colonne da una tabella (tabella di sistema in questo caso):

```
SELECT *
FROM sys.objects
```

Oppure seleziona solo alcune colonne specifiche:

```
SELECT object_id, name, type, create_date
FROM sys.objects
```

Filtra le righe usando la clausola WHERE

La clausola WHERE filtra solo le righe che soddisfano alcune condizioni:

```
SELECT *
FROM sys.objects
WHERE type = 'IT'
```

Ordina i risultati utilizzando ORDER BY

La clausola ORDER BY ordina le righe nel set di risultati restituito da alcune colonne o espressioni:

```
SELECT *
FROM sys.objects
ORDER BY create_date
```

Raggruppa i risultati utilizzando GROUP BY

La clausola GROUP BY raggruppa le righe in base a qualche valore:

```
SELECT type, count(*) as c
FROM sys.objects
```

```
GROUP BY type
```

È possibile applicare alcune funzioni su ciascun gruppo (funzione di aggregazione) per calcolare la somma o il conteggio dei record nel gruppo.

genere	c
SQ	3
S	72
IT	16
PK	1
U	5

Filtra i gruppi usando la clausola HAVING

La clausola HAVING rimuove i gruppi che non soddisfano la condizione:

```
SELECT type, count(*) as c
FROM sys.objects
GROUP BY type
HAVING count(*) < 10
```

genere	c
SQ	3
PK	1
U	5

Restituzione solo delle prime N righe

La clausola TOP restituisce solo le prime N righe nel risultato:

```
SELECT TOP 10 *
FROM sys.objects
```

Impaginazione utilizzando OFFSET FETCH

La clausola OFFSET FETCH è la versione più avanzata di TOP. Ti permette di saltare le righe N1 e prendere le prossime righe N2:

```
SELECT *
FROM sys.objects
```

```
ORDER BY object_id
OFFSET 50 ROWS FETCH NEXT 10 ROWS ONLY
```

Puoi utilizzare OFFSET senza recupero per saltare solo le prime 50 righe:

```
SELECT *
FROM sys.objects
ORDER BY object_id
OFFSET 50 ROWS
```

SELEZIONA senza FROM (nessuna fonte di dati)

L'istruzione SELECT può essere eseguita senza la clausola FROM:

```
declare @var int = 17;

SELECT @var as c1, @var + 2 as c2, 'third' as c3
```

In questo caso, viene restituita una riga con valori / risultati di espressioni.

Leggi SELECT statement online: https://riptutorial.com/it/sql-server/topic/4662/select-statement

Capitolo 94: sequenze

Examples

Crea sequenza

```
CREATE SEQUENCE [dbo].[CustomersSeq]
AS INT
START WITH 10001
INCREMENT BY 1
MINVALUE -1;
```

Usa sequenza in tabella

```
CREATE TABLE [dbo].[Customers]
(
    CustomerID INT DEFAULT (NEXT VALUE FOR [dbo].[CustomersSeq]) NOT NULL,
    CustomerName VARCHAR(100),
);
```

Inserisci nella tabella con sequenza

```
INSERT INTO [dbo].[Customers]
       ([CustomerName])
 VALUES
       ('Jerry'),
       ('Gorge')

SELECT * FROM [dbo].[Customers]
```

risultati

Identificativo del cliente	Nome del cliente
10001	Jerry
10002	Gola

Elimina da e inserisci nuovo

```
DELETE FROM [dbo].[Customers]
WHERE CustomerName = 'Gorge';

INSERT INTO [dbo].[Customers]
       ([CustomerName])
 VALUES ('George')

SELECT * FROM [dbo].[Customers]
```

risultati

Identificativo del cliente	Nome del cliente
10001	Jerry
10003	Giorgio

Leggi sequenze online: https://riptutorial.com/it/sql-server/topic/5324/sequenze

Capitolo 95: Sicurezza a livello di riga

Examples

Predicato del filtro RLS

Sql Server 2016+ e il database Azure Sql consentono di filtrare automaticamente le righe restituite nell'istruzione select utilizzando un determinato predicato. Questa funzione è chiamata **sicurezza a livello di riga** .

Innanzitutto, è necessaria una funzione valutata a livello di tabella che contenga un predicato che descriva quale sia la condizione che consentirà agli utenti di leggere i dati da una tabella:

```
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 questo esempio, il predicato afferma che solo gli utenti che hanno un valore in SESSION_CONTEXT che corrisponde all'argomento di input possono accedere alla società. Puoi inserire qualsiasi altra condizione, ad es. Che controlli il ruolo del database o l'id_database dell'utente corrente, ecc.

> La maggior parte del codice sopra riportato è un modello che verrà copiato e incollato. L'unica cosa che cambierà qui è il nome e gli argomenti di predicato e condizione nella clausola WHERE. Ora si creano criteri di sicurezza che applicheranno questo predicato su alcune tabelle.

Ora puoi creare una politica di sicurezza che applicherà il predicato su alcune tabelle:

```
CREATE SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany(CompanyID) ON dbo.Company
    WITH (State=ON)
```

Questa politica di sicurezza assegna un predicato alla tabella aziendale. Ogni volta che qualcuno tenta di leggere i dati dalla tabella Company, i criteri di sicurezza applicheranno il predicato su ogni riga, passeranno la colonna CompanyID come parametro del predicato e il predicato valuterà se questa riga verrà restituita nel risultato della query SELECT.

Modifica della politica di sicurezza RLS

La politica di sicurezza è un gruppo di predicati associati alle tabelle che possono essere gestiti insieme. È possibile aggiungere o rimuovere predicati o attivare / disattivare l'intera politica.

È possibile aggiungere più predicati sulle tabelle nella politica di sicurezza esistente.

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    ADD FILTER PREDICATE dbo.pUserCanAccessCompany(CompanyID) ON dbo.Company
```

È possibile eliminare alcuni predicati dalla politica di sicurezza:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy
    DROP FILTER PREDICATE ON dbo.Company
```

È possibile disabilitare la politica di sicurezza

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = OFF );
```

È possibile abilitare la politica di sicurezza che è stata disabilitata:

```
ALTER SECURITY POLICY dbo.CompanyAccessPolicy WITH ( STATE = ON );
```

Impedire l'aggiornamento utilizzando il predicato del blocco RLS

La sicurezza a livello di riga consente di definire alcuni predicati che controlleranno chi potrebbe aggiornare le righe nella tabella. Per prima cosa è necessario definire una funzione valore di tabella che rappresenti un predicato che controllerà la politica di accesso.

CREA FUNZIONE

dbo.pUserCanAccessProduct (@CompanyID int)

```
RETURNS TABLE
WITH SCHEMABINDING
```

AS RETURN (SELEZIONA 1 come canAccess WHERE

CAST (SESSION_CONTEXT (N'CompanyID ') come int) = @CompanyID

) In questo esempio, il predicato afferma che solo gli utenti che hanno un valore in SESSION_CONTEXT che corrisponde all'argomento di input possono accedere alla società. Puoi inserire qualsiasi altra condizione, ad es. Che controlli il ruolo del database o l'id_database dell'utente corrente, ecc.

> La maggior parte del codice sopra riportato è un modello che verrà copiato e incollato. L'unica cosa che cambierà qui è il nome e gli argomenti di predicato e condizione nella clausola WHERE. Ora si creano criteri di sicurezza che applicheranno questo

predicato su alcune tabelle.

Ora possiamo creare una politica di sicurezza con il predicato che bloccherà gli aggiornamenti sulla tabella dei prodotti se la colonna CompanyID nella tabella non soddisfa il predicato.

CREARE LA POLITICA DI SICUREZZA dbo.ProductAccessPolicy AGGIUNGI BLOCCO PREDICA dbo.pUserCanAccessProduct (CompanyID) ON dbo.Product

Questo predicato verrà applicato su tutte le operazioni. Se vuoi applicare il predicato su qualche operazione puoi scrivere qualcosa come:

CREA POLITICA DI SICUREZZA dbo.ProductAccessPolicy AGGIUNGI BLOCCO PREDICA dbo.pUserCanAccessProduct (CompanyID) ON dbo.Product DOPO INSERIMENTO

Le possibili opzioni che è possibile aggiungere dopo la definizione del predicato del blocco sono:

[{DOPO {INSERIRE | AGGIORNARE } }
| {PRIMA {AGGIORNAMENTO | ELIMINA } }]

Leggi Sicurezza a livello di riga online: https://riptutorial.com/it/sql-server/topic/7045/sicurezza-a-livello-di-riga

Capitolo 96: Sposta e copia i dati intorno alle tabelle

Examples

Copia i dati da una tabella all'altra

Questo codice seleziona i dati da una tabella e li visualizza nello strumento di query (in genere SSMS)

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

Questo codice inserisce tali dati in una tabella:

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

Copia i dati in una tabella, creando al volo quella tabella

Questo codice seleziona i dati da una tabella:

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

Questo codice crea una nuova tabella denominata `MyNewTable` e inserisce tali dati in essa

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

Spostare i dati in una tabella (assumendo il metodo delle chiavi univoche)

Per *spostare i* dati, devi prima inserirli nella destinazione, quindi eliminare quello che hai inserito dalla tabella di origine. Questa non è una normale operazione SQL ma potrebbe essere illuminante

Cosa hai inserito? Normalmente nei database è necessario disporre di una o più colonne che è possibile utilizzare per identificare in modo univoco le righe, in modo da poterle assumere e utilizzarle.

Questa affermazione seleziona alcune righe

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

Per prima cosa inseriamo questi nella nostra tabella di destinazione:

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

Ora *assumendo record in entrambe le tabelle sono unici* su `Key1` , `Key2` , possiamo usarlo per trovare e cancellare i dati dalla tabella di origine

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

Ciò funziona correttamente solo se `Key1` , `Key2` sono unici in entrambe le tabelle

Infine, non vogliamo che il lavoro sia finito a metà. Se lo completiamo in una transazione, tutti i dati verranno spostati o non accadrà nulla. Ciò garantisce che non inseriamo i dati in quel momento e che non siamo in grado di eliminare i dati dalla fonte.

```
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;
```

Leggi Sposta e copia i dati intorno alle tabelle online: https://riptutorial.com/it/sql-server/topic/1467/sposta-e-copia-i-dati-intorno-alle-tabelle

Capitolo 97: SQL dinamico

Examples

Esegui istruzione SQL fornita come stringa

In alcuni casi, è necessario eseguire la query SQL inserita nella stringa. EXEC, EXECUTE o procedura di sistema sp_executesql può eseguire qualsiasi query SQL fornita come stringa:

```
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')
```

Questa procedura restituirà lo stesso set di risultati della query SQL fornita come testo dell'istruzione. sp_executesql può eseguire query SQL fornite come string letteral, variable / parameter o even expression:

```
declare @table nvarchar(40) = N'product items'
EXEC(N'SELECT * FROM ' + @table)
declare @sql nvarchar(40) = N'SELECT * FROM ' + QUOTENAME(@table);
EXEC sp_executesql @sql
```

Hai bisogno della funzione QUOTENAME per sfuggire ai caratteri speciali nella variabile @table. Senza questa funzione si otterrebbe l'errore di sintassi se la variabile @table contiene qualcosa come spazi, parentesi o qualsiasi altro carattere speciale.

SQL dinamico eseguito come utente diverso

Puoi eseguire query SQL come utente diverso usando AS USER = 'nome utente del database'

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

La query SQL verrà eseguita con l'utente del database dbo. Verranno verificati tutti i controlli di autorizzazione applicabili all'utente dbo sulla query SQL.

SQL Injection con SQL dinamico

Le query dinamiche sono

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

Se il valore della variabile utente è ***myusername '' OR 1 = 1 -*** verrà eseguita la seguente query:

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

Il commento alla fine del valore della variabile @nomeutente commenterà la parte finale della query e verrà valutata la condizione 1 = 1. L'applicazione che verifica che ci sia almeno un utente restituito da questa query restituirà un conteggio maggiore di 0 e l'accesso avrà esito positivo.

Utilizzando questo approccio, l'attaccante può accedere all'applicazione anche se non conosce nome utente e password validi.

SQL dinamico con parametri

Per evitare problemi di iniezione e di fuga, le query SQL dinamiche devono essere eseguite con parametri, ad esempio:

```
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
```

Il secondo parametro è un elenco di parametri utilizzati nella query con i loro tipi, dopo questo elenco vengono fornite le variabili che verranno utilizzate come valori dei parametri.

sp_executesql eviterà caratteri speciali ed eseguirà query sql.

Leggi SQL dinamico online: https://riptutorial.com/it/sql-server/topic/6871/sql-dinamico

Capitolo 98: SQL Server Evolution attraverso diverse versioni (2000 - 2016)

introduzione

Sto usando SQL Server dal 2004. Ho iniziato con il 2000 e ora userò SQL Server 2016. Ho creato tabelle, viste, funzioni, trigger, stored procedure e scritto molte query SQL ma non ho usato molte nuove funzionalità dalle successive versioni. L'ho cercato su google, ma sfortunatamente non ho trovato tutte le funzionalità in un unico posto. Così ho raccolto e convalidato queste informazioni da diverse fonti e messo qui. Sto solo aggiungendo le informazioni di alto livello per tutte le versioni a partire dal 2000 al 20

Examples

SQL Server versione 2000 - 2016

Le seguenti funzionalità aggiunte in SQL Server 2000 rispetto alla versione precedente:

1. Sono stati aggiunti nuovi tipi di dati (BIGINT, SQL_VARIANT, TABLE)
2. Invece di e per Trigger sono stati introdotti come avanzamento al DDL.
3. Integrità referenziale a cascata.
4. Supporto XML
5. Funzioni definite dall'utente e viste delle partizioni.
6. Visualizzazioni indicizzate (Consentire l'indice sulle viste con colonne calcolate).

Le seguenti funzionalità aggiunte nella versione 2005 rispetto alla versione precedente:

1. Miglioramento nella clausola TOP con l'opzione "WITH TIES".
2. Comandi di manipolazione dei dati (DML) e clausola OUTPUT per ottenere valori INSERTED e DELETED
3. Gli operatori PIVOT e UNPIVOT.
4. Gestione delle eccezioni con blocco TRY / CATCH
5. Funzioni di classificazione
6. Common Table Expressions (CTE)
7. Common Language Runtime (Integrazione di linguaggi .NET per creare oggetti come stored procedure, trigger, funzioni, ecc.)
8. Service Broker (Gestione del messaggio tra mittente e destinatario in modo approssimativo)
9. Crittografia dei dati (funzionalità native per supportare la crittografia dei dati archiviati in database definiti dall'utente)
10. Posta SMTP
11. Endpoint HTTP (Creazione di endpoint mediante semplice istruzione T-SQL che espone un oggetto a cui accedere tramite Internet)
12. Più set di risultati attivi (MARS). Ciò consente a una connessione di database persistente da un singolo client di avere più di una richiesta attiva per connessione.

13. SQL Server Integration Services (verrà utilizzato come strumento ETL primario (estrazione, trasformazione e caricamento)
14. Miglioramenti in Analysis Services e Reporting Services.
15. Partizionamento di tabelle e indici. Consente il partizionamento di tabelle e indici in base ai limiti delle partizioni, come specificato da una FUNZIONE PARTITION con le singole partizioni mappate ai gruppi di file tramite uno SCHEMA DI PARTITION.

Le seguenti funzionalità aggiunte nella versione 2008 dalla sua versione precedente:

1. Miglioramento nei tipi di dati DATE e TIME esistenti
2. Nuove funzioni come - SYSUTCDATETIME () e SYSDATETIMEOFFSET ()
3. Colonne di ricambio: per risparmiare una quantità significativa di spazio su disco.
4. Grandi tipi definiti dall'utente (fino a 2 GB di dimensione)
5. Introdotta una nuova funzionalità per passare un tipo di dati tabella in stored procedure e funzioni
6. Nuovo comando MERGE per le operazioni INSERT, UPDATE e DELETE
7. Nuovo tipo di dati HierarchyID
8. Tipi di dati spaziali: per rappresentare la posizione fisica e la forma di qualsiasi oggetto geometrico.
9. Domande e rapporti più rapidi con GROUPING SETS: un'estensione della clausola GROUP BY.
10. Miglioramento dell'opzione di archiviazione FILESTREAM

Le seguenti funzionalità aggiunte nella versione 2008 R2 dalla versione precedente:

1. PowerPivot - Per l'elaborazione di set di dati di grandi dimensioni.
2. Generatore di report 3.0
3. Pronto per il cloud
4. StreamInsight
5. Master Data Services
6. Integrazione con SharePoint
7. DACPAC (Pacchetti di componenti di applicazioni a livello dati)
8. Miglioramento in altre funzionalità di SQL Server 2008

Le seguenti funzionalità aggiunte nella versione 2012 dalla sua versione precedente:

1. Indici archivio colonne: riduce l'I / O e l'utilizzo della memoria su query di grandi dimensioni.
2. Impaginazione: l'impaginazione può essere eseguita usando i comandi "OFFSET" e "FETCH".
3. Database contenuto: grande funzionalità per migrazioni periodiche di dati.
4. Gruppi di disponibilità AlwaysOn
5. Supporto di Windows Server Core
6. Ruoli server definiti dall'utente
7. Supporto per i Big Data
8. PowerView
9. Miglioramenti di SQL Azure
10. Modello tabulare (SSAS)
11. Servizi di qualità dei dati DQS

12. Tabella file: un miglioramento della funzionalità FILESTREAM introdotta nel 2008.
13. Miglioramento nella gestione degli errori inclusa l'istruzione THROW
14. Miglioramento a SQL Server Management Studio Debug a. SQL Server 2012 introduce più opzioni per controllare i punti di interruzione. b. Miglioramenti alle finestre in modalità di debug
    c. Miglioramento in IntelliSense - come l'inserimento di frammenti di codice.

Le seguenti funzionalità aggiunte nella versione 2014 dalla sua versione precedente:

1. Motore OLTP in memoria: migliora le prestazioni fino a 20 volte.
2. Miglioramenti AlwaysOn
3. Estensione del pool di buffer
4. Caratteristiche del cloud ibrido
5. Miglioramento degli indici del negozio di colonne (come gli indici degli archivi di colonne aggiornabili)
6. Miglioramenti nella gestione delle query (come SELECT INTO parallelo)
7. Power BI per l'integrazione di Office 365
8. Durabilità ritardata
9. Miglioramenti per i backup del database

Le seguenti funzionalità aggiunte nella versione 2016 dalla sua versione precedente:

1. Always Encrypted - Always Encrypted è progettato per proteggere i dati a riposo o in movimento.
2. Analytics operativo in tempo reale
3. PolyBase in SQL Server
4. Supporto JSON nativo
5. Negozio di query
6. Miglioramenti a AlwaysOn
7. OLTP potenziato in memoria
8. Più file di database TempDB
9. Stretch Database
10. Sicurezza a livello di riga
11. Miglioramenti in memoria

    Miglioramenti T-SQL o nuove aggiunte in SQL Server 2016

1. TRUNCATE TABLE con PARTITION

2. GOCCIA SE ESISTE

3. Funzioni STRING_SPLIT e STRING_ESCAPE

4. ALTER TABLE ora può modificare molte colonne mentre la tabella rimane online, usando WITH (ONLINE = ON | OFF).

5. MAXDOP per DBCC CHECKDB, DBCC CHECKTABLE e DBCC CHECKFILEGROUP

6. ALTER DATABASE SET AUTOGROW_SINGLE_FILE

7. ALTER DATABASE SET AUTOGROW_ALL_FILES
8. Funzioni COMPRESS e DECOMPRESS
9. FORMATMESSAGE Statement
10. Il 2016 introduce altre 8 proprietà con SERVERPROPERTY

un. InstanceDefaultDataPath

b. InstanceDefaultLogPath

c. ProductBuild

d. ProductBuildType

e. ProductMajorVersion

f. ProductMinorVersion

g. ProductUpdateLevel

h. ProductUpdateReference

Leggi SQL Server Evolution attraverso diverse versioni (2000 - 2016) online: https://riptutorial.com/it/sql-server/topic/10129/sql-server-evolution-attraverso-diverse-versioni--2000---2016-

Capitolo 99: SQL Server Management Studio (SSMS)

introduzione

SQL Server Management Studio (SSMS) è uno strumento per gestire e amministrare SQL Server e il database SQL.

SSMS è offerto gratuitamente da Microsoft.

La documentazione SSMS è disponibile.

Examples

Aggiornamento della cache IntelliSense

Quando gli oggetti vengono creati o modificati, non sono automaticamente disponibili per IntelliSense. Per renderli disponibili a IntelliSense, è necessario aggiornare la cache locale.

All'interno di una finestra dell'editor di query premi `Ctrl + Shift + R` o seleziona `Edit | IntelliSense | Refresh Local Cache` dal menu.

Dopo questo, tutte le modifiche dall'ultimo aggiornamento saranno disponibili per IntelliSense.

Leggi SQL Server Management Studio (SSMS) online: https://riptutorial.com/it/sql-server/topic/10642/sql-server-management-studio--ssms-

Capitolo 100: SQLCMD

Osservazioni

È necessario essere nel percorso in cui SQLCMD.exe esiste o aggiungerlo alla variabile di ambiente PATH.

Examples

SQLCMD.exe chiamato da un file batch o da una riga di comando

```
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"
```

I file batch di questo tipo possono essere utilizzati per automatizzare le attività, ad esempio per eseguire backup di database in un momento specifico (può essere pianificato con Task Scheduler) per una versione di SQL Server Express in cui non è possibile utilizzare i lavori dell'agente.

Leggi SQLCMD online: https://riptutorial.com/it/sql-server/topic/5396/sqlcmd

Capitolo 101: String Aggregate funzioni in SQL Server

Examples

Utilizzo di STUFF per l'aggregazione di stringhe

Abbiamo una tabella degli studenti con SubjectId. Qui il requisito è di concatenare basato su subjectId.

Tutte le versioni di SQL Server

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

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

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

String_Agg per String Aggregation

In caso di SQL Server 2017 o vnext, possiamo utilizzare STRING_AGG incorporato per questa aggregazione. Per lo stesso tavolo dello studente,

```
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
```

Leggi String Aggregate funzioni in SQL Server online: https://riptutorial.com/it/sql-server/topic/9892/string-aggregate-funzioni-in-sql-server

Capitolo 102: subquery

Examples

subquery

Una sottoquery è una query all'interno di un'altra query SQL. Una sottoquery viene anche chiamata query interna o selezione interna e l'istruzione contenente una sottoquery viene chiamata query esterna o selezione esterna.

Nota

1. Le sottoquery devono essere racchiuse tra parentesi,
2. Un ORDER BY non può essere utilizzato in una sottoquery.
3. Il tipo di immagine come BLOB, array, tipi di dati di testo non sono consentiti nelle sottoquery.

Le sottoquery possono essere utilizzate con select, insert, update e delete statement all'interno di where, from, select clausola insieme a IN, operatori di confronto, ecc.

Abbiamo una tabella denominata ITCompanyInNepal su cui eseguiremo le query per mostrare esempi di subquery:

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

Esempi: **Sottocartelle con istruzione Select**

In con operatore e clausola **where:**

```
SELECT *
FROM ITCompanyInNepal
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'USA');
```

con **operatore di confronto** e **dove** clausola

```
SELECT *
FROM ITCompanyInNepal
```

```
WHERE NumberOfEmployee < (SELECT AVG(NumberOfEmployee)
                          FROM ITCompanyInNepal
                        )
```

con clausola **select**

```
SELECT   CompanyName,
         CompanyAddress,
         Headquarter,
         (Select SUM(NumberOfEmployee)
         FROM ITCompanyInNepal
         Where Headquarter = 'USA') AS TotalEmployeeHiredByUSAInKathmandu
FROM     ITCompanyInNepal
WHERE    CompanyAddress = 'Kathmandu' AND Headquarter = 'USA'
```

Sottoquery con istruzione di inserimento

Dobbiamo inserire i dati dalla tabella IndianCompany a ITCompanyInNepal. La tabella per IndianCompany è la seguente:

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
```

Sottoquery con la dichiarazione di aggiornamento

Supponiamo che tutte le società che hanno sede negli Stati Uniti decidano di licenziare 50 dipendenti da tutte le società statunitensi del Nepal a causa di qualche cambiamento nella politica delle società statunitensi.

```
UPDATE ITCompanyInNepal
SET NumberOfEmployee = NumberOfEmployee - 50
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'USA')
```

Sottoquery con istruzione di cancellazione

Supponiamo che tutte le società la cui sede è la Danimarca abbiano deciso di chiudere le loro aziende dal Nepal.

```
DELETE FROM ITCompanyInNepal
WHERE Headquarter IN (SELECT Headquarter
                      FROM ITCompanyInNepal
                      WHERE Headquarter = 'Denmark')
```

Leggi subquery online: https://riptutorial.com/it/sql-server/topic/5629/subquery

Capitolo 103: Suggerimenti per la ricerca

Examples

ISCRIVITI

Quando si uniscono due tabelle, Query Optimizer (QO) di SQL Server può scegliere diversi tipi di join che verranno utilizzati nella query:

- HASH join
- LOOP join
- Unisciti a MERGE

QO esplorerà i piani e sceglierà l'operatore ottimale per unire le tabelle. Tuttavia, se sei sicuro di sapere quale sarebbe l'operatore di join ottimale, puoi specificare quale tipo di JOIN dovrebbe essere usato. Inner LOOP join costringerà QO a scegliere Nested loop join durante l'unione di due tabelle:

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

l'unione interna di unione forzerà l'operatore di join MERGE:

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

l'hash join interno costringerà l'operatore di HASH join:

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

GRUPPO PER CONSIGLI

Quando si utilizza la clausola GROUP BY, SQL Server Query Optimizer (QO) può scegliere diversi tipi di operatori di raggruppamento:

- HASH Aggregate che crea hash-map per il raggruppamento delle voci
- Stream Aggregate che funziona bene con gli input preordinati

È possibile richiedere esplicitamente che QO scelga uno o un altro operatore aggregato se si sa quale sarebbe l'ottimale. Con OPTION (ORDER GROUP), QO sceglierà sempre Stream

aggregate e aggiungerà l'operatore Sort davanti a Stream aggregate se l'input non è ordinato:

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

Con OPTION (GRUPPO HASH), QO sceglierà sempre l'aggregato hash:

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

Suggerimento di righe VELOCI

Specifica che la query è ottimizzata per il recupero rapido dei primi number_rows. Questo è un numero intero non negativo. Dopo che sono stati restituiti i primi number_rows, la query continua l'esecuzione e produce il set di risultati completo.

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

UNION suggerisce

Quando si utilizza l'operatore UNION su due risultati di query, Query Optimizer (QO) può utilizzare gli operatori seguenti per creare un'unione di due set di risultati:

- Unione (unione)
- Concat (Union)
- Hash Match (Union)

È possibile specificare esplicitamente quale operatore deve essere utilizzato utilizzando il suggerimento OPTION ():

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

Opzione MAXDOP

Specifica il grado massimo di parallelismo per la query che specifica questa opzione.

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

Questa opzione sovrascrive l'opzione di configurazione MAXDOP di sp_configure e Resource Governor. Se MAXDOP è impostato su zero, il server sceglie il grado massimo di parallelismo.

INDICE Suggerimenti

I suggerimenti dell'indice vengono utilizzati per forzare una query a utilizzare un indice specifico, invece di consentire allo Strumento di ottimizzazione query di SQL Server di scegliere ciò che ritiene l'indice migliore. In alcuni casi è possibile ottenere vantaggi specificando l'indice che deve essere utilizzato da una query. Generalmente, Query Optimizer di SQL Server sceglie l'indice migliore adatto per la query, ma a causa di statistiche mancanti o obsolete o di esigenze specifiche è possibile forzarlo.

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

Leggi Suggerimenti per la ricerca online: https://riptutorial.com/it/sql-server/topic/6881/suggerimenti-per-la-ricerca

Capitolo 104: Tabelle temporali

Osservazioni

SQL Server 2016 introduce il supporto per le tabelle temporali con versione di sistema come funzionalità di database che offre supporto integrato per fornire informazioni sui dati archiviati nella tabella in qualsiasi momento, anziché solo i dati corretti al momento attuale.

Una tabella temporale con versione di sistema è un nuovo tipo di tabella utente in SQL Server 2016, progettata per conservare una cronologia completa delle modifiche dei dati e consentire un'analisi puntuale semplice. Questo tipo di tabella temporale viene indicato come tabella temporale con versione di sistema poiché il periodo di validità per ogni riga è gestito dal sistema (ad es. Il motore di database). Ogni tabella temporale ha due colonne definite in modo esplicito, ciascuna con un tipo di dati datetime2. Queste colonne sono indicate come colonne del periodo. Queste colonne del periodo vengono utilizzate esclusivamente dal sistema per registrare il periodo di validità per ogni riga ogni volta che viene modificata una riga.

Examples

CREA tabelle temporali

```
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));
```

INSERTI: su un **INSERT** , il sistema imposta il valore per la colonna **ValidFrom sull'ora** di inizio della transazione corrente (nel fuso orario UTC) in base all'orologio di sistema e assegna il valore per la colonna **ValidTo** al valore massimo di 9999- 12-31. Questo contrassegna la riga come aperta.

AGGIORNAMENTI: in un **UPDATE** , il sistema memorizza il valore precedente della riga nella tabella cronologia e imposta il valore per la colonna **ValidTo sull'ora** di inizio della transazione corrente (nel fuso orario UTC) in base all'orologio di sistema. Questo contrassegna la riga come chiusa, con un periodo registrato per il quale la riga era valida. Nella tabella corrente, la riga viene aggiornata con il suo nuovo valore e il sistema imposta il valore per la colonna **ValidFrom** per l'ora di inizio della transazione (nel fuso orario UTC) in base all'orologio di sistema. Il valore per la riga aggiornata nella tabella corrente per la colonna **ValidTo** rimane il valore massimo di 9999-12-

31.

DELETE : su un **CANC** , il sistema memorizza il valore precedente della riga nella tabella cronologia e imposta il valore per la colonna **ValidTo sull'ora** di inizio della transazione corrente (nel fuso orario UTC) in base all'orologio di sistema. Questo contrassegna la riga come chiusa, con un periodo registrato per il quale la riga precedente era valida. Nella tabella corrente, la riga viene rimossa. Le query della tabella corrente non restituiranno questa riga. Solo le query che trattano i dati della cronologia restituiscono i dati per i quali viene chiusa una riga.

MERGE : su **MERGE** , l'operazione si comporta esattamente come se fossero eseguite fino a tre istruzioni (un **INSERT** , un **UPDATE** e / o un **DELETE**), a seconda delle azioni specificate **nell'istruzione MERGE** .

Suggerimento: i tempi registrati nelle colonne datetime2 del sistema si basano sull'ora di inizio della transazione stessa. Ad esempio, tutte le righe inserite all'interno di una singola transazione avranno la stessa ora UTC registrata nella colonna corrispondente all'inizio del periodo **SYSTEM_TIME** .

Come posso interrogare i dati temporali?

```
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;
```

Restituisce il valore attuale specificato nel tempo (FOR SYSTEM_TIME AS OF)

Restituisce una tabella con righe contenenti i valori effettivi (correnti) nel momento specifico nel passato.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME AS  OF '2016-08-06 08:32:37.91'
```

PER SYSTEM_TIME TRA E

Come sopra indicato in FOR SYSTEM_TIME FROM <start_date_time> TO <end_date_time> description, ad eccezione della tabella di righe restituite include le righe che sono diventate attive sul limite superiore definito dall'endpoint <end_date_time>.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME BETWEEN  '2015-01-01' AND '2015-12-31'
```

PER SYSTEM_TIME FROM A

Restituisce una tabella con i valori per tutte le versioni di riga attive nell'intervallo di tempo specificato, indipendentemente dal fatto che inizino a essere attive prima che il valore del

parametro <start_date_time> per l'argomento FROM sia cessato dopo il valore del parametro <end_date_time> per A argomento. Internamente, viene eseguita un'unione tra la tabella temporale e la sua tabella cronologia ei risultati vengono filtrati per restituire i valori per tutte le versioni di riga attive in qualsiasi momento durante l'intervallo di tempo specificato. Le righe che sono diventate attive esattamente sul limite inferiore definito dall'endpoint FROM sono incluse e i record che sono diventati attivi esattamente sul limite superiore definito dall'endpoint TO non sono inclusi.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME FROM '2015-01-01' TO '2015-12-31'
```

PER SYSTEM_TIME CONTENUTO IN (,)

Restituisce una tabella con i valori per tutte le versioni di riga aperte e chiuse nell'intervallo di tempo specificato definito dai due valori datetime per l'argomento CONTAINED IN. Le righe che sono diventate attive esattamente sul limite inferiore o sono state disattivate esattamente al limite superiore sono incluse.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME CONTAINED IN ('2015-04-01', '2015-09-25')
```

PER SYSTEM_TIME ALL

Restituisce l'unione di righe che appartengono alla tabella corrente e alla cronologia.

```
SELECT * FROM Employee
    FOR SYSTEM_TIME ALL
```

Creazione di una tabella temporale con versione di sistema ottimizzata per la memoria e pulizia della tabella di cronologia di SQL Server

La creazione di una tabella temporale con una tabella cronologica predefinita è un'opzione utile quando si desidera controllare la denominazione e continuare a fare affidamento sul sistema per creare la tabella cronologica con la configurazione predefinita. Nell'esempio seguente, una nuova tabella temporale ottimizzata per la memoria con versione di sistema collegata a una nuova tabella di cronologia basata su disco.

```
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 )
    );
```

Pulizia della tabella della cronologia di SQL Server Nel corso del tempo la tabella della cronologia può aumentare in modo significativo. Poiché l'inserimento, l'aggiornamento o l'eliminazione dei dati dalla tabella della cronologia non sono consentiti, l'unico modo per ripulire la tabella della cronologia è innanzitutto disabilitare il controllo delle versioni del sistema:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = OFF); PARTIRE

Elimina i dati non necessari dalla tabella della cronologia:

```
    DELETE FROM dbo.EmployeeHistory
```

WHERE EndTime <= '2017-01-26 14:00:29';

e quindi riattivare il controllo delle versioni del sistema:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = ON (HISTORY_TABLE = [dbo]. [EmployeeHistory], DATA_CONSISTENCY_CHECK = ON));

La pulizia della tabella della cronologia nei database SQL di Azure è leggermente diversa, poiché i database SQL di Azure hanno il supporto integrato per la pulizia della tabella della cronologia. Innanzitutto, è necessario abilitare la pulizia della cronologia temporale a livello di database:

```
ALTER DATABASE CURRENT
```

Imposta TEMPORAL_HISTORY_RETENTION ON GO

Quindi imposta il periodo di conservazione per tabella:

```
ALTER TABLE dbo.Employee
```

SET (SYSTEM_VERSIONING = ON (HISTORY_RETENTION_PERIOD = 90 DAYS));

Questo eliminerà tutti i dati nella tabella della cronologia più vecchi di 90 giorni. I database on-premise di SQL Server 2016 non supportano TEMPORAL_HISTORY_RETENTION e HISTORY_RETENTION_PERIOD e una delle due query precedenti viene eseguita nei database on-premise di SQL Server 2016 si verificheranno i seguenti errori.

Per TEMPORAL_HISTORY_RETENTION l'errore sarà:

```
Msg 102, Level 15, State 6, Line 34
```

Sintassi errata vicino a "TEMPORAL_HISTORY_RETENTION".

L'errore HISTORY_RETENTION_PERIOD sarà:

```
Msg 102, Level 15, State 1, Line 39
```

Sintassi errata in prossimità di "HISTORY_RETENTION_PERIOD".

Leggi Tabelle temporali online: https://riptutorial.com/it/sql-server/topic/5296/tabelle-temporali

Capitolo 105: Tasti di scelta rapida di Microsoft SQL Server Management Studio

Examples

Esempi di scelta rapida

1. Apri una nuova finestra di query con la connessione corrente (`Ctrl` + `N`)
2. Passa tra le schede aperte (`Ctrl` + `Tab`)
3. Mostra / nascondi riquadro Risultati (`Ctrl` + `R`)
4. Esegui query evidenziata (`Ctrl` + `E`)
5. Rendi il testo selezionato in maiuscolo o minuscolo (`Ctrl` + `Maiusc` + `U` , `Ctrl` + `Maiusc` + `L`)
6. Membro della lista Intellisense e parola completa (`Ctrl` + `Spazio` , `Tab`)
7. Vai alla riga (`Ctrl` + `G`)
8. chiudi una scheda in SQL Server Managament Studio (`Ctrl` + `F4`)

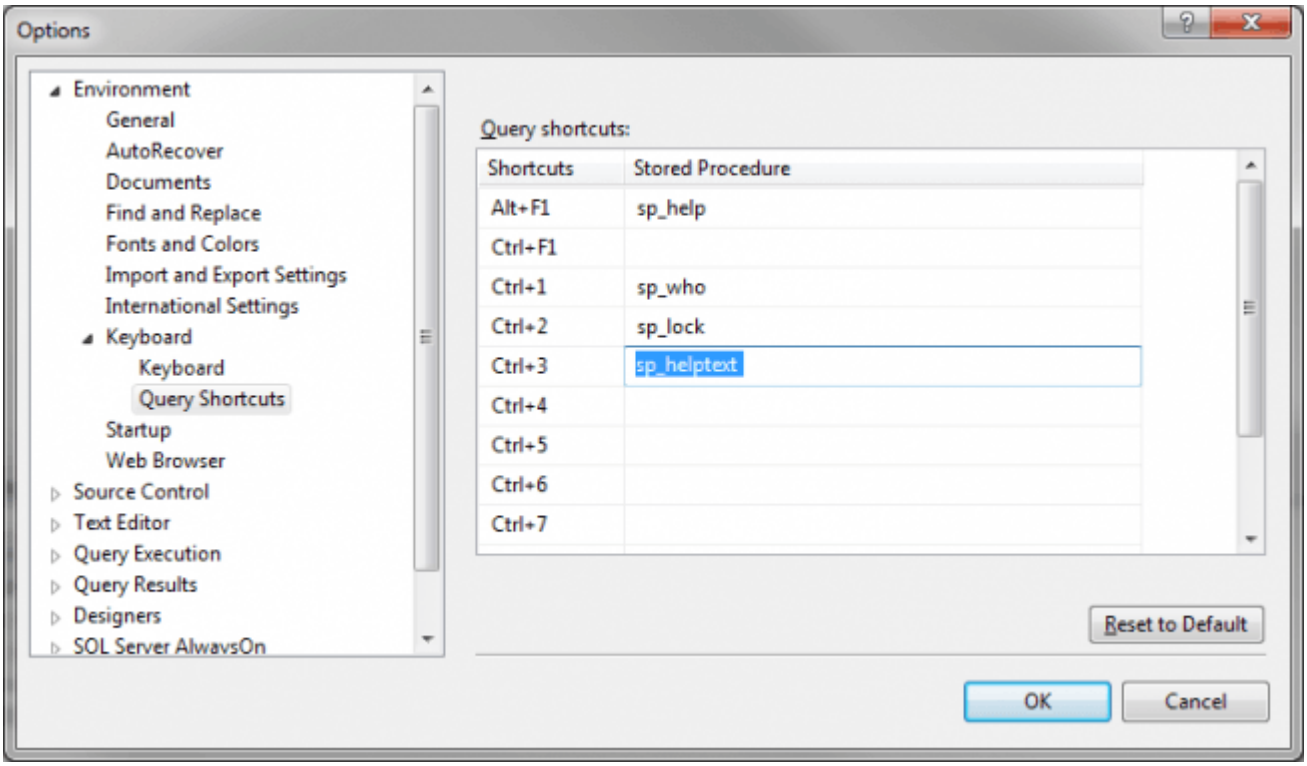
Menu Activation Tasti di scelta rapida

1. Passare alla barra dei menu di SQL Server Management Studio (`ALT`)
2. Attiva il menu per un componente strumento (`ALT` + `HYPHEN`)
3. Visualizza il menu di scelta rapida (`MAIUSC` + `F`)
4. Visualizza la finestra di dialogo Nuovo file per creare un file (`CTRL` + `N`)
5. Visualizza la finestra di dialogo Apri progetto per aprire un progetto esistente (`CTRL` + `MAIUSC` + `O`)
6. Visualizza la finestra di dialogo Aggiungi nuovo elemento per aggiungere un nuovo file al progetto corrente (`CTRL` + `MAIUSC` + `A`)
7. Visualizza la finestra di dialogo Aggiungi elemento esistente per aggiungere un file esistente al progetto corrente (`CTRL` + `MAIUSC` + `A`)
8. Visualizza la Progettazione query (`CTRL` + `MAIUSC` + `Q`)
9. Chiudere un menu o una finestra di dialogo, annullando l'azione (`ESC`)

Scorciatoie da tastiera personalizzate

Vai a Strumenti -> Opzioni. Vai a Ambiente -> Tastiera -> Scorciatoie delle query

Sul lato destro puoi vedere alcune scorciatoie che sono di default in SSMS. Ora se hai bisogno di aggiungerne uno, fai clic su qualsiasi colonna nella colonna Stored Procedure.



Clicca OK. Ora si prega di andare a una finestra di query e selezionare la stored procedure quindi premere CTRL + 3, mostrerà il risultato della stored procedure.

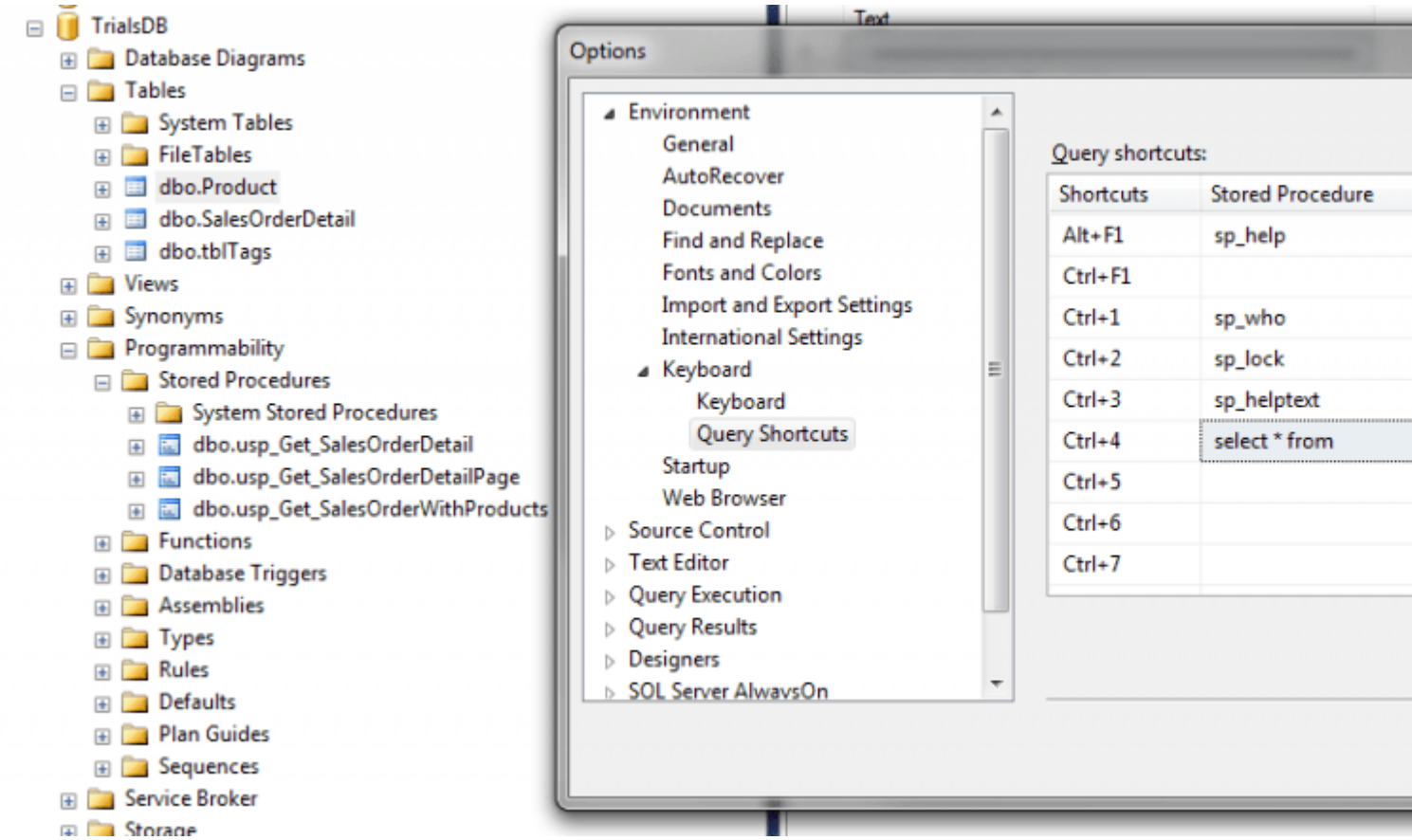
```
usp_Get_SalesOrderDetail
```

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

Ora se è necessario selezionare tutti i record da una tabella quando si seleziona la tabella e si preme CTRL + 5 (è possibile selezionare qualsiasi chiave). È possibile effettuare la scorciatoia come segue.



Ora vai avanti e seleziona il nome della tabella dalla finestra della query e premi CTRL + 4 (il tasto che abbiamo selezionato), ti darà il risultato.

Leggi Tasti di scelta rapida di Microsoft SQL Server Management Studio online: https://riptutorial.com/it/sql-server/topic/7749/tasti-di-scelta-rapida-di-microsoft-sql-server-management-studio

Capitolo 106: Tipi di dati

introduzione

In questa sezione vengono descritti i tipi di dati che possono essere utilizzati da SQL Server, inclusi l'intervallo di dati, la lunghezza e le limitazioni (se presenti).

Examples

Numeri esatti

Esistono due classi base di tipi di dati numerici esatti: **Integer** , **Fixed Precision e Scale** .

Tipi di dati interi

- po
- tinyint
- smallint
- int
- bigint

Gli interi sono valori numerici che non contengono mai una parte frazionaria e utilizzano sempre una quantità fissa di spazio di archiviazione. L'intervallo e le dimensioni di archiviazione dei tipi di dati interi sono mostrati in questa tabella:

Tipo di dati	Gamma	Conservazione
po	0 o 1	1 bit **
tinyint	Da 0 a 255	1 byte
smallint	-2 ^ 15 (-32.768) a 2 ^ 15-1 (32.767)	2 byte
int	-2 ^ 31 (-2.147.483.648) a 2 ^ 31-1 (2.147.483.647)	4 byte
bigint	-2 ^ 63 (-9.223.372.036.854.775.808) a 2 ^ 63-1 (9.223.372.036.854.775.807)	8 byte

Risolti i tipi di dati di precisione e scala

- numerico
- decimale
- smallmoney
- i soldi

Questi tipi di dati sono utili per rappresentare esattamente i numeri. Finché i valori possono rientrare nell'intervallo dei valori memorizzabili nel tipo di dati, il valore non avrà problemi di arrotondamento. Questo è utile per qualsiasi calcolo finanziario, in cui gli errori di arrotondamento si tradurranno in follia clinica per i contabili.

Si noti che **decimale** e **numerico** sono sinonimi per lo stesso tipo di dati.

Tipo di dati	Gamma	Conservazione
Decimale [(p [, s])] o Numerico [(p [, s])]	-10 ^ 38 + 1 a 10 ^ 38 - 1	Vedi tabella di **precisione**

Quando si definisce un tipo di dati *decimale* o *numerico* , potrebbe essere necessario specificare Precisione [p] e Scala [s].

La precisione è il numero di cifre che possono essere memorizzate. Ad esempio, se è necessario memorizzare valori compresi tra 1 e 999, è necessaria una precisione di 3 (per contenere le tre cifre su 100). Se non si specifica una precisione, la precisione predefinita è 18.

Scala è il numero di cifre dopo il punto decimale. Se è necessario memorizzare un numero compreso tra 0,00 e 999,99, è necessario specificare una precisione di 5 (cinque cifre) e una scala di 2 (due cifre dopo il punto decimale). È necessario specificare una precisione per specificare una scala. La scala predefinita è zero.

La precisione di un tipo di dati *decimale* o *numerico* definisce il numero di byte necessari per memorizzare il valore, come mostrato di seguito:

Tabella di precisione

Precisione	Byte di archiviazione
1 - 9	5
10-19	9
20-28	13
29-38	17

Tipi di dati fissi monetari

Questi tipi di dati sono destinati specificamente alla contabilità e ad altri dati monetari. Questi tipi hanno una scala fissa di 4 - vedrai sempre quattro cifre dopo la cifra decimale. Per la maggior parte dei sistemi che funzionano con la maggior parte delle valute, sarà sufficiente utilizzare un valore *numerico* con una Scala di 2. Si noti che nessuna informazione sul tipo di valuta rappresentata viene memorizzata con il valore.

Tipo di dati	Gamma	Conservazione
i soldi	-922,337,203,685,477,5808 a 922,337,203,685,477,5807	8 byte
smallmoney	Da -214.748.3648 a 214.748.3647	4 byte

Numeri approssimativi

- float [(**n**)]
- vero

Questi tipi di dati vengono utilizzati per memorizzare numeri in virgola mobile. Poiché questi tipi sono destinati a contenere solo valori numerici approssimativi, questi non dovrebbero essere utilizzati nei casi in cui qualsiasi errore di arrotondamento non è accettabile. Tuttavia, se è necessario gestire numeri molto grandi o numeri con un numero indeterminato di cifre dopo la cifra decimale, queste potrebbero essere la soluzione migliore.

Tipo di dati	Gamma	Taglia
galleggiante	-1.79 E + 308 a -2.23E-308, 0 e 2.23E-308 a 1.79E + 308	dipende da **n** nella tabella sottostante
vero	-3.40E + 38 a -1.18E - 38, 0 e 1.18E - 38 a 3.40E + 38	4 byte

n tabella dei valori per i numeri *float* . Se non viene specificato alcun valore nella dichiarazione del float, verrà utilizzato il valore predefinito 53. Si noti che *float (24)* è l'equivalente di un valore *reale* .

n valore	Precisione	Taglia
1-24	7 cifre	4 byte
25-53	15 cifre	8 byte

Data e ora

Questi tipi sono in tutte le versioni di SQL Server

- appuntamento
- smalldatetime

Questi tipi sono disponibili in tutte le versioni di SQL Server dopo SQL Server 2012

- Data
- datetimeoffset
- datetime2
- tempo

Stringhe di caratteri

- carbonizzare
- varchar
- testo

Stringhe di caratteri Unicode

- nchar
- nvarchar
- ntext

Stringhe binarie

- binario
- varbinary
- Immagine

Altri tipi di dati

- cursore
- timestamp
- hierarchyid
- identificativo unico
- sql_variant
- xml
- tavolo
- Tipi spaziali

Leggi Tipi di dati online: https://riptutorial.com/it/sql-server/topic/5260/tipi-di-dati

Capitolo 107: Tipi di tabella definiti dall'utente

introduzione

I tipi di tabella definiti dall'utente (in breve UDT) sono tipi di dati che consentono all'utente di definire una struttura di tabella. I tipi di tabella definiti dall'utente supportano chiavi primarie, vincoli univoci e valori predefiniti.

Osservazioni

Gli UDT hanno le seguenti restrizioni:

- non può essere usato come una colonna in una tabella o in un campo in tipi strutturati definiti dall'utente
- non è possibile creare un indice non cluster in un UDT a meno che l'indice non sia il risultato della creazione di un vincolo PRIMARY KEY o UNIQUE sull'UDT
- La definizione UDT NON PU be essere modificata dopo la sua creazione

Examples

creando un UDT con una singola colonna int che è anche una chiave primaria

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

Creazione di un UDT con più colonne

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

Creazione di un UDT con un vincolo univoco:

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

Nota: i vincoli nei tipi di tabella definiti dall'utente non possono essere nominati.

Creazione di un UDT con una chiave primaria e una colonna con un valore predefinito:

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

Leggi Tipi di tabella definiti dall'utente online: https://riptutorial.com/it/sql-server/topic/5280/tipi-di-tabella-definiti-dall-utente

Capitolo 108: Ultima identità inserita

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();
```

Ciò restituirà il valore di identità aggiunto più recente prodotto sulla stessa connessione, nell'ambito corrente. In questo caso, 1, per la prima riga nella tabella dbo.person.

@@IDENTITÀ

```
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;
```

Ciò restituirà l'identità aggiunta più di recente sulla stessa connessione, indipendentemente dall'ambito. In questo caso, qualunque sia il valore corrente della colonna Identity su logging_table, supponendo che non si stia verificando un'altra attività sull'istanza di SQL Server e

nessun altro trigger generato da questo inserimento.

IDENT_CURRENT ('tablename')

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

Questo selezionerà il valore dell'identità aggiunto più recentemente nella tabella selezionata, indipendentemente dalla connessione o dall'ambito.

@@ IDENTITY e 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
```

Gli ultimi due valori delle istruzioni SELECT sono gli stessi.

Leggi Ultima identità inserita online: https://riptutorial.com/it/sql-server/topic/5674/ultima-identita-inserita

Capitolo 109: UNIONE

Examples

Unione e unione tutti

L' operazione **Unione** combina i risultati di due o più query in un singolo set di risultati che include tutte le righe che appartengono a tutte le query nell'unione e ignorerà eventuali duplicati esistenti. **Anche Union** fa la stessa cosa ma include anche i valori duplicati. Il concetto di operazione sindacale sarà chiaro dall'esempio seguente. Poche cose da considerare durante l'utilizzo di unione sono:

1. Il numero e l'ordine delle colonne devono essere gli stessi in tutte le query.

2. I tipi di dati devono essere compatibili.

Esempio:

Abbiamo tre tabelle: Marksheet1, Marksheet2 e Marksheet3. Marksheet3 è la tabella duplicata di Marksheet2 che contiene gli stessi valori di Marksheet2.

Tabella1 : Marksheet1

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

Tabella2 : 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

Unione su tabelle Marksheet1 e Marksheet2

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
```

Nota: l'output per l'unione delle tre tabelle sarà uguale all'unione su Marksheet1 e Marksheet2 perché l'operazione di unione non assume valori duplicati.

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

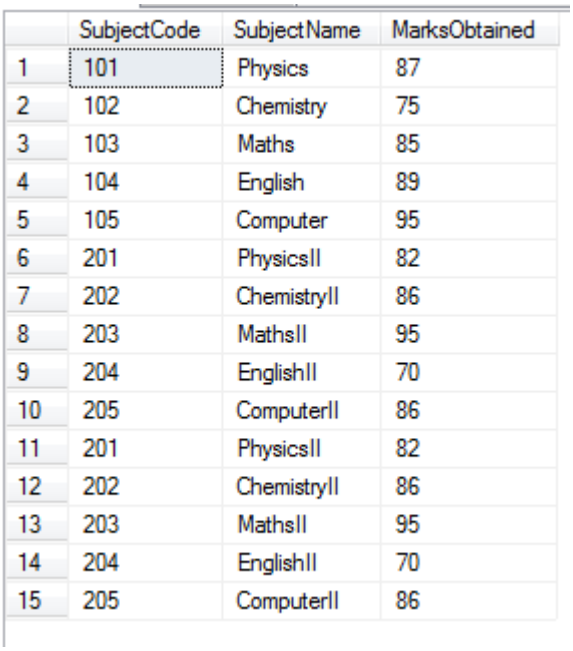
PRODUZIONE

	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

Unione tutti

```
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet1
UNION ALL
SELECT CourseCode, CourseName, MarksObtained
FROM Marksheet2
UNION ALL
SELECT SubjectCode, SubjectName, MarksObtained
FROM Marksheet3
```

PRODUZIONE



	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

Qui noterai che i valori duplicati di Marksheet3 vengono visualizzati anche utilizzando l'unione tutti.

Leggi UNIONE online: https://riptutorial.com/it/sql-server/topic/5590/unione

Capitolo 110: Uso della tabella TEMP

Osservazioni

Le tabelle temporanee sono davvero molto utili.

La tabella può essere creata in fase di esecuzione e può eseguire tutte le operazioni eseguite in una tabella normale.

Queste tabelle sono create in un database tempdb.

Usato quando?

1. Dobbiamo fare un'operazione di join complessa.
2. Facciamo un gran numero di manipolazioni di riga nelle stored procedure.
3. Può sostituire l'uso del cursore.

Così aumenta le prestazioni.

Examples

Tabella Temp locale

- Sarà disponibile fino a quando la connessione corrente persiste per l'utente.

  Eliminato automaticamente quando l'utente si disconnette.

  Il nome dovrebbe iniziare con # (#temp)

  ```
  CREATE TABLE #LocalTempTable(
                  StudentID      int,
                  StudentName    varchar(50),
                  StudentAddress varchar(150))
  ```

```
insert into #LocalTempTable values ( 1, 'Ram','India');

select * from #LocalTempTable
```

Dopo aver eseguito tutte queste istruzioni se chiudiamo la finestra della query e la apriamo di nuovo e proviamo ad inserirla e selezionarla mostrerà un messaggio di errore

```
“Invalid object name #LocalTempTable”
```

Tabella Temp globale

- Comincerà con ## (## temp).

  Verrà cancellato solo se l'utente disconnette tutte le connessioni.

  Si comporta come un tavolo permanente.

```
CREATE TABLE ##NewGlobalTempTable(
               StudentID      int,
               StudentName    varchar(50),
               StudentAddress varchar(150))

Insert Into ##NewGlobalTempTable values ( 1,'Ram','India');
Select * from ##NewGlobalTempTable
```

Nota: questi sono visualizzabili da tutti gli utenti del database, indipendentemente dal livello di autorizzazioni.

Eliminazione di tabelle temporanee

Le tabelle temporali devono avere ID univoci (all'interno della sessione, per le tabelle temporanee locali o all'interno del server, per le tabelle temporali globali). Cercando di creare una tabella utilizzando un nome già esistente, verrà restituito il seguente errore:

```
There is already an object named '#tempTable' in the database.
```

Se la query produce tabelle temporanee e si desidera eseguirla più di una volta, è necessario eliminare le tabelle prima di provare a generarle di nuovo. La sintassi di base per questo è:

```
drop table #tempTable
```

Cercando di eseguire questa sintassi prima che la tabella esista (ad es. Alla prima esecuzione della sintassi) causerà un altro errore:

```
Cannot drop the table '#tempTable', because it does not exist or you do not have permission.
```

Per evitare ciò, puoi verificare se la tabella esiste già prima di rilasciarla, in questo modo:

```
IF OBJECT_ID ('tempdb..#tempTable', 'U') is not null DROP TABLE #tempTable
```

Leggi Uso della tabella TEMP online: https://riptutorial.com/it/sql-server/topic/5328/uso-della-tabella-temp

Capitolo 111: Utility bcp (bulk copy program)

introduzione

Il programma di utilità bulk copy program (bcp) copia i dati tra un'istanza di Microsoft SQL Server e un file di dati in un formato specificato dall'utente. L'utilità bcp può essere utilizzata per importare un numero elevato di nuove righe in tabelle di SQL Server o per esportare i dati dalle tabelle in file di dati.

Examples

Esempio di importazione di dati senza un file di formato (utilizzando il formato nativo)

```
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;"
```

Leggi Utility bcp (bulk copy program) online: https://riptutorial.com/it/sql-server/topic/10942/utility-bcp--bulk-copy-program-

Capitolo 112: variabili

Sintassi

- DECLARE @VariableName DataType [= Value];
- SET @VariableName = Value;

Examples

Dichiarare una variabile di tabella

```
DECLARE @Employees TABLE
(
    EmployeeID INT NOT NULL PRIMARY KEY,
    FirstName NVARCHAR(50) NOT NULL,
    LastName NVARCHAR(50) NOT NULL,
    ManagerID INT NULL
)
```

Quando si crea una tabella normale, si utilizza la sintassi `CREATE TABLE Name (Columns)` . Quando si crea una variabile di tabella, si utilizza la `DECLARE @Name TABLE (Columns)` .

Per fare riferimento alla variabile di tabella all'interno di un'istruzione `SELECT` , SQL Server richiede che si dia un alias alla variabile di tabella, altrimenti si otterrà un errore:

> Deve dichiarare la variabile scalare "@TableVariableName".

vale a dire

```
DECLARE @Table1 TABLE (Example INT)
DECLARE @Table2 TABLE (Example INT)

/*
-- the following two commented out statements would generate an error:
SELECT *
FROM @Table1
INNER JOIN @Table2 ON @Table1.Example = @Table2.Example

SELECT *
FROM @Table1
WHERE @Table1.Example = 1
*/

-- but these work fine:
SELECT *
FROM @Table1 T1
INNER JOIN @Table2 T2 ON T1.Example = T2.Example

SELECT *
FROM @Table1 Table1
WHERE Table1.Example = 1
```

Aggiornamento di una variabile usando SET

```
DECLARE @VariableName INT
SET @VariableName = 1
PRINT @VariableName
```

1

Usando `SET` , puoi aggiornare solo una variabile alla volta.

Aggiornamento delle variabili usando SELECT

Utilizzando `SELECT` , è possibile aggiornare più variabili contemporaneamente.

```
DECLARE @Variable1 INT, @Variable2 VARCHAR(10)
SELECT @Variable1 = 1, @Variable2 = 'Hello'
PRINT @Variable1
PRINT @Variable2
```

1

Ciao

Quando si utilizza `SELECT` per aggiornare una variabile da una colonna della tabella, se ci sono più valori, utilizzerà l' *ultimo* valore. (Si applicano le normali regole dell'ordine - se non viene fornito alcun ordinamento, l'ordine non è garantito.)

```
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

Se non ci sono righe restituite dalla query, il valore della variabile non cambierà:

```
SELECT TOP 0 @Variable = Example
```

```
FROM #Test
ORDER BY Example ASC

PRINT @Variable
```

1

Dichiarare più variabili contemporaneamente, con valori iniziali

```
DECLARE
  @Var1 INT = 5,
  @Var2 NVARCHAR(50) = N'Hello World',
  @Var3 DATETIME = GETDATE()
```

Operatori di assegnazione composti

SQL Server 2008 R2

Operatori composti supportati:

- `+=` Aggiungi e assegna
- `-=` Sottrai e assegna
- `*=` Moltiplica e assegna
- `/=` Dividi e assegna
- `%=` Modulo e assegnare
- `&=` Bitwise AND e assegnare
- `^=` Bit XOR bit e assegnare
- `|=` Bitwise OR e assegnare

Esempio di utilizzo:

```
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
```

Aggiornamento delle variabili selezionando da una tabella

A seconda della struttura dei dati, è possibile creare variabili che si aggiornano in modo dinamico.

```
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)
```

Nella maggior parte dei casi, è necessario assicurarsi che la query restituisca un solo valore quando si utilizza questo metodo.

Leggi variabili online: https://riptutorial.com/it/sql-server/topic/2566/variabili

Capitolo 113: Visualizzazioni

Osservazioni

Le viste sono query memorizzate che possono essere interrogate come tabelle normali. Le viste non fanno parte del modello fisico del database. Eventuali modifiche applicate all'origine dati di una vista, ad esempio una tabella, si rifletteranno anche nella vista.

Examples

Crea una vista

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

Crea o sostituisci vista

Questa query eliminerà la vista, se già esiste, e ne creerà una nuova.

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

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

Creare una vista con associazione allo schema

Se viene creata una vista WITH SCHEMABINDING, le tabelle sottostanti non possono essere eliminate o modificate in modo tale da interrompere la visualizzazione. Ad esempio, una colonna di tabella a cui si fa riferimento in una vista non può essere rimossa.

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

Le viste *senza il* collegamento dello schema possono interrompersi se le loro tabelle sottostanti cambiano o vengono eliminate. Interrogare una vista interrotta genera un messaggio di errore. sp_refreshview può essere utilizzato per garantire che le viste esistenti senza il collegamento dello schema non siano interrotte.

Leggi Visualizzazioni online: https://riptutorial.com/it/sql-server/topic/5327/visualizzazioni

Titoli di coda

S. No	Capitoli	Contributors
1	Introduzione a 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	Aderire	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
3	Analizzando una query	DForck42
4	Attività pianificata o lavoro	Edathadan Chief aka Arun, Hadi
5	Autorizzazioni del database	Ben Thul
6	Autorizzazioni e sicurezza	5arx
7	Backup e ripristino del database	Jones Joseph, Jovan MSFT
8	Broker di servizi	Ken S., Matej, RamenChef
9	Chiavi esterne	Jovan MSFT
10	Chiavi primarie	Kritner
11	Ciclo WHILE	lord5et, Matas Vaitkevicius, podiluska, RamenChef, Wojciech Kazior

12	Clausola OVER	Athafoud, bluefeet, Brandon, DVT, gofr1, Lamak, Paul Bambury , RamenChef, Rowland Shaw, Sam
13	COALESCE	Bharat Prasad Satyal, Edathadan Chief aka Arun, Karthikeyan, Matej, scsimon, Tab Alleman
14	Colonne calcolate	cnayak, Kannan Kandasamy
15	COLONNERO CLUSTER	Jovan MSFT
16	Common Language Runtime Integration	Jovan MSFT
17	Con l'opzione Ties	TheGameiswar
18	Conversione di tipi di dati	Ben O, Edathadan Chief aka Arun
19	CREA VISTA	Almir Vuk, cteski, Edathadan Chief aka Arun, Hadi, Josh B, Robert Columbia, Tot Zam
20	crittografia	Rubenisme
21	croce si applicano	Hamza Rabah, Jovan MSFT, Tom V
22	Cursori	Kane, Phrancis
23	Database di sistema - TempDb	Anuj Tripathi, RamenChef
24	Date	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
25	Dati spaziali	cteski, Neil Kennedy, RamenChef, Vladimir Oselsky
26	DBCC	Jovan MSFT
27	dbmail	Phrancis
28	Delimitazione di caratteri speciali e parole riservate	bassrek
29	Dichiarazione CASE	Laughing Vergil, RamenChef, Vikas Vaidya
30	Elimina la parola chiave	Ignas, Jakub Ojmucianski, Justin Rohr, Max, scsimon

31	Esportare i dati nel file txt usando SQLCMD	sheraz mirza
32	Espressioni di tabella comuni	Arif, bbrown, cteski, DForck42, Jeffrey Van Laethem, Jovan MSFT, kafka, Keith Hall, Monty Wild, SQLMason
33	Filestream	Raghu Ariga
34	Funzione Split Split in Sql Server	Jibin Balachandran, Jovan MSFT, MasterBob, Pரதீ ப், RamenChef
35	Funzioni aggregate	Akshay Anand, cnayak, cteski, Jeffrey L Whitledge, Joe Taras, Vexator
36	Funzioni della finestra	andyabel, feetwet, MarmiK
37	Funzioni di classifica	cteski, kolunar, New
38	Funzioni di stringa	A_Arnold, anon, cteski, FoxyBOA, Hadi, hatchet, Igor Micev, Jibin Balachandran, Jovan MSFT, mtb, Phrancis, Raidri, Ricardo C, Ross Presser, takrl, Zohar Peled
39	Funzioni logiche	dd4711
40	Generare un intervallo di date	James, Siyual
41	Gestione del database SQL di Azure	Jovan MSFT
42	Gestione delle transazioni	Metanormal
43	grilletto	Oluwafemi, The_Outsider, Zohar Peled
44	Gruppo di file	Behzad
45	Importazione BULK	Jovan MSFT
46	Indice	Ahmad Aghazadeh, Akshay Anand, cteski, Henrik Staun Poulsen, Martin Smith, Tom V
47	Indicizzazione di testo completo	Edathadan Chief aka Arun
48	In-Memory OLTP (Hekaton)	Akshay Anand, Behzad, Brandon, Jovan MSFT, Martijn Pieters

49	Inserire	Randall
50	INSERIRE	Abubakar Riaz, barcanoj, DVJex, Hari K M, intox, martinshort, Matas Vaitkevicius, Max, Michael Stum, n00b, Piotr Nawrot, Robert Columbia, Tot Zam, woony
51	Installazione di SQL Server su Windows	Luis Bosquez
52	Interrogare i risultati per pagina	Pat
53	Istantanee del database	Akash, Daryl, Jovan MSFT, Wolfgang
54	JSON in SQL Server	Jovan MSFT, Mono
55	La funzione STUFF	Arthur D, bluefeet, Chetan Sanghani, dacohenii, Kiran Ukande, Luis Bosquez, MrE, user1690166
56	Limite risultato impostato	alalp, chrisb, cteski, ErikE
57	Livelli di isolamento delle transazioni	Phrancis
58	Livelli di isolamento e bloccaggio	RamenChef, sqlandmore.com
59	Mascheramento dinamico dei dati	Jovan MSFT
60	Memorizzazione di JSON nelle tabelle SQL	Ed Harper, Jovan MSFT, RamenChef
61	MERGE	Abhilash R Vankayala, Abubakar Riaz, Alex, David Kaminski, dd4711, Hari K M, Moshiour, Rogerio Soares, Serg
62	Migrazione	Matas Vaitkevicius
63	Modifica il testo JSON	Jovan MSFT
64	Moduli compilati in modo nativo (Hekaton)	bakedpatato, Jovan MSFT
65	Negozio di query	bakedpatato, Jovan MSFT

66	Nomi alias in Sql Server	Pரதீ ப்
67	NULL	Amir Pourmand, Hadi, Kannan Kandasamy, Kritner, Laughing Vergil, podiluska, Sean Branchaw, Zohar Peled
68	OPENJSON	James, Jovan MSFT
69	Operazioni DDL di base in MS SQL Server	Matt
70	Opzioni avanzate	Ahmad Aghazadeh
71	Ordinamento / ordine di file	APH, OzrenTkalcecKrznaric
72	ORDINATO DA	APH, beercohol, cteski, Gidil, RamenChef
73	paginatura	cteski, Jovan MSFT, Sender
74	Parametri valutati a tabella	Zohar Peled
75	ParseName	Mani
76	partizionamento	Dan Guzman, James Anderson, John Odom, Susang
77	PER IL PERCORSO XML	bluefeet, gotqn, Keith Hall, Wolfgang
78	PER JSON	James, Jovan MSFT
79	PHANTOM legge	Max
80	PIVOT / UNPIVOT	Athafoud, bluefeet, DhruvJoshi, kolunar
81	Pivot SQL dinamico	Jesse
82	Privilegi o permessi	Oluwafemi
83	Procedura di archiviazione	Bino Mathew Varghese, cnayak, cteski, Erik Oppedijk, Eugene Niemand, Hari K M, Jayasurya Satheesh, Matas Vaitkevicius, Nathan Skerl, Pirate X, scsimon
84	PROVA A PRENDERE	Jovan MSFT, ravindra, Uberzen1
85	Query con dati JSON	bakedpatato, James, Jovan MSFT

86	RAGGRUPPA PER	Andy, Edathadan Chief aka Arun, Jenism, juergen d, Julien Vavasseur, Kiran Ukande, Matas Vaitkevicius, ShlomiR
87	Recupera informazioni sul database	Andrea, Anuj Tripathi, Baodad, Brent Ozar, dario, feetwet, James Anderson, JamieA, Jasmin Solanki, Jeffrey Van Laethem, jyao, Kritner, Laughing Vergil, LowlyDBA, Mahendra, Moshiour, Phrancis, Rhumborl, scsimon, Shiva, spaghettidba, Tot Zam, TZHX, Umachandar - Microsoft
88	Recupera informazioni sulla tua istanza	Bino Mathew Varghese, feetwet, James Anderson, Kritner, LowlyDBA, S.Karras, scsimon
89	Resource Governor	Ako, RamenChef
90	schemi	Merenix
91	SCOPE_IDENTITY ()	Dheeraj Kumar
92	SE ALTRO	cteski, M.Ali, RamenChef
93	SELECT statement	cteski, Jovan MSFT
94	sequenze	Josh Morel
95	Sicurezza a livello di riga	Carsten Hynne, Jovan MSFT
96	Sposta e copia i dati intorno alle tabelle	Nick.McDermaid
97	SQL dinamico	Jovan MSFT
98	SQL Server Evolution attraverso diverse versioni (2000 - 2016)	Dan Guzman, M.Ali
99	SQL Server Management Studio (SSMS)	dd4711
100	SQLCMD	Eugene Niemand, Techie
101	String Aggregate funzioni in SQL Server	Kannan Kandasamy
102	subquery	cnayak

103	Suggerimenti per la ricerca	cteski, DARKOCEAN, Jovan MSFT, user_0
104	Tabelle temporali	Ahmad Aghazadeh, Akshay Anand, Ben O, Mspaja
105	Tasti di scelta rapida di Microsoft SQL Server Management Studio	Bino Mathew Varghese, cteski, Sibeesh Venu
106	Tipi di dati	Laughing Vergil, Matas Vaitkevicius
107	Tipi di tabella definiti dall'utente	Jivan, Zohar Peled
108	Ultima identità inserita	Jeffrey Van Laethem, sqluser, Tot Zam
109	UNIONE	cnayak
110	Uso della tabella TEMP	APH, New
111	Utility bcp (bulk copy program)	MarmiK
112	variabili	APH, Keith Hall, Phrancis
113	Visualizzazioni	Benjamin Hodgson, Daniel Lemke, Max