SQL Server: Trasferire le statistiche di una tabella

#### di [Sergio Govoni](https://mvp.support.microsoft.com/profile/Sergio.Govoni) – Microsoft MVP ([blog](http://community.ugiss.org/blogs/sgovoni/))

* 1. 

*Maggio 2012*

## Introduzione

In questo articolo cercheremo di rispondere alla domanda: Come posso simulare, in ambiente di test, le query che sono eseguite in produzione? Senza fare il restore del DB nell’ambiente di test?

E’ possibile esportare le Statistiche di un database in produzione e importarle in un database di test con lo stesso schema? Simulando quindi la densità e la distribuzione dei dati nelle tabelle in produzione?

Prima di descrivere la soluzione, pensiamo a quello che avviene “dietro le quinte” quando chiediamo a SQL Server di eseguire una query. Le tre fasi fondamentali compiute, dall’Engine di SQL Server, durante l’esecuzione di una query sono:

* + [Query Parsing](#_Query_Parsing)
  + [Query Optimizer](#_Query_Optimizer)
  + [Query Execution](#_Query_Execution)

## Query Parsing

Durante la prima fase denominata Query Parsing, SQL Server verifica la sintassi del comando T-SQL, controlla che gli elementi (operatori) del linguaggio siano stati scritti e specificati correttamente; l’output di questa fase è noto come Logical Operator Tree e rappresenta l’input per la fase successiva svolta da un componente chiamato Query Optimizer.

## Query Optimizer

Query Optimizer ha il compito di trasformare una query T-SQL in un piano di esecuzione; genera e valuta molteplici piani di esecuzione con l’obiettivo di sceglierne uno, quello che meglio permette di raggiungere ai dati. La scelta del miglior piano di esecuzione passa attraverso la valutazione delle statistiche che permettono di conoscere la selettività (valore di densità) degli indici definiti nelle tabelle cui SQL Server dovrà accedere. Le statistiche consentono al Query Optimizer di valutare correttamente il costo dei possibili piani di esecuzione e di sceglierne uno.

L’output di questa fase è noto come Estimated Execution Plan e rappresenta l’input per il Query Execution.

## Query Execution

Query Execution esegue la query nello Storage Engine in accordo con il piano di esecuzione stimato.

## Import/Export delle Statistiche di una tabella

Le Statistiche influenzano quindi il Query Optimizer nella scelta del piano di esecuzione ottimale, è per questo motivo che vogliamo esportarle da un database in produzione per importarle in un database di test, al fine di simulare la presenza dei dati all’interno delle tabelle e analizzare in tutta tranquillità (offline) i piani di esecuzione scelti da SQL Server per determinate query.

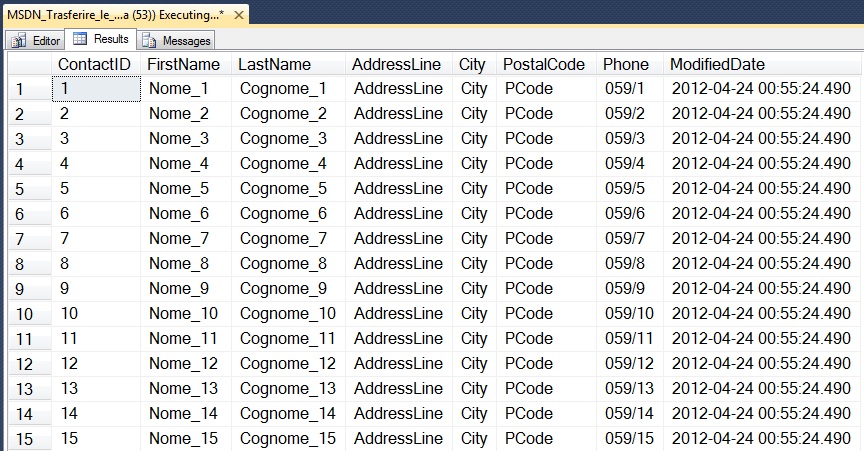
Ipotizziamo di avere un database in produzione chiamato DB\_Export\_Stats che contiene, per semplicità, soltanto una tabella denominata dbo.Contact che rappresenta l’anagrafica contatti. Il seguente frammento di codice in linguaggio T-SQL permette di eseguire il setup del database DB\_Export\_Stats:

* 1. ------------------------------------------------------------------------
  2. -- Setup DB for export statistics
  3. ------------------------------------------------------------------------
  4. use [master];
  5. go
  6. if exists(select \* from sys.databases where name = 'db\_export\_stats')
  7. begin
  8. alter database [db\_export\_stats]
  9. set single\_user with rollback immediate;
  10. drop database [db\_export\_stats];
  11. end
  12. go
  13. -- Create database
  14. create database [db\_export\_stats];
  15. go

Dopo aver creato il database, eseguiamo il seutp della tabella dbo.Contact e il caricamento di 1.000.000 di record (contatti di test) utilizzando il seguente codice T-SQL:

* 1. ------------------------------------------------------------------------
  2. -- Setup table and insert data
  3. ------------------------------------------------------------------------
  4. -- Change database context
  5. use [db\_export\_stats];
  6. go
  7. -- Create table dbo.Contact (drop if exists)
  8. if object\_id('dbo.Contact', 'U') is not null
  9. drop table dbo.Contact;
  10. go
  11. create table dbo.Contact
  12. (
  13. ContactID int identity(1, 1) not null
  14. ,FirstName varchar(128) not null
  15. ,LastName varchar(128) not null
  16. ,AddressLine varchar(40) default 'AddressLine'
  17. ,City varchar(40) default 'City'
  18. ,PostalCode varchar(5) default 'PCode'
  19. ,Phone varchar(20) not null
  20. ,ModifiedDate datetime default getdate()
  21. );
  22. go
  23. -- Create function dbo.udf\_GetNums
  24. if (object\_id('dbo.udf\_GetNums') is not null)
  25. drop function dbo.udf\_GetNums;
  26. go
  27. create function dbo.udf\_GetNums(@m as bigint) returns table
  28. as
  29. return
  30. with
  31. cte0 as (select n = 1 union all select n = 1),
  32. cte1 as (select n = 1 from cte0 as a, cte0 as b),
  33. cte2 as (select n = 1 from cte1 as a, cte1 as b),
  34. cte3 as (select n = 1 from cte2 as a, cte2 as b),
  35. cte4 as (select n = 1 from cte3 as a, cte3 as b),
  36. cte5 as (select n = 1 from cte4 as a, cte4 as b),
  37. cteres as (select row\_number() over (order by n) as n from cte5)
  38. select n from cteres where n <= @m;
  39. go
  40. with cte\_rows as
  41. (
  42. select
  43. n
  44. from
  45. dbo.udf\_GetNums(1000000)
  46. )
  47. insert into dbo.Contact
  48. (
  49. FirstName,
  50. LastName,
  51. Phone
  52. )
  53. select
  54. 'Nome\_' + ltrim(str(n)),
  55. 'Cognome\_' + ltrim(str(n)),
  56. '059/' + ltrim(str(n))
  57. from
  58. cte\_rows;
  59. go

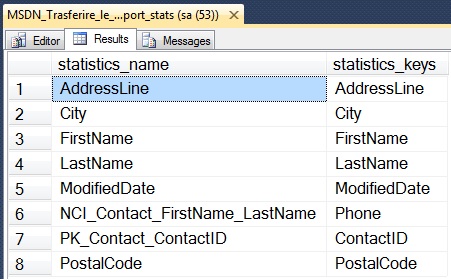
Familiarizziamo con i dati di prova inseriti nella tabella dbo.Contact, la Figura 1 illustra l’output che si ottiene eseguendo la SELECT riportata di seguito:

* 1. select \* from dbo.Contact;
  2. go
  3. 
  4. Figura 1 – Contenuto della tabella dbo.Contact

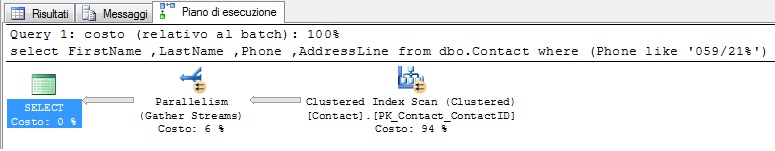
Applichiamo ora la Primary Key alla tabella dbo.Contact definita sulla colonna ContactID; la creazione di questo Constraint, in questo scenario, sulla tabella dbo.Contact ha come effetto la creazione dell’indice cluster sulla tabella stessa. Poi eseguiamo la creazione di un indice non cluster per migliorare le performance delle query che interrogano i contatti applicando un filtro sulla colonna Phone. Osserviamo anche l’utilizzo della stored procedure di sistema [sp\_createstats](http://msdn.microsoft.com/en-us/library/ms186834.aspx) che permette di creare le Statistiche per ogni colonna nelle tabelle utente del database corrente.

* 1. ------------------------------------------------------------------------
  2. -- Setup constraint and statistics
  3. ------------------------------------------------------------------------
  4. alter table dbo.Contact
  5. add constraint PK\_Contact\_ContactID primary key (ContactID);
  6. go
  7. create index NCI\_Contact\_FirstName\_LastName on dbo.Contact
  8. (
  9. [Phone]
  10. )
  11. include
  12. (
  13. [FirstName]
  14. ,[LastName]
  15. );
  16. go
  17. exec sp\_createstats;
  18. go

Consultiamo ora le statistiche riguardanti la tabella dbo.Contact, l’output è illustrato in Figura 2:

* 1. exec sp\_helpstats
  2. @objname = 'dbo.Contact',
  3. @results = 'ALL';
  4. go
  5. 
  6. Figura 2 – Statistiche per la tabella dbo.Contact

Abbiamo allestito il database di produzione, ipotizziamo ora ci siano diversi utenti pronti a interrogare i dati attraverso un’applicazione, uno di loro desidera selezionare i contatti filtrandoli per la colonna Phone attraverso l’operatore LIKE e invia a SQL Server la query riportata di seguito. In Figura 3 osserviamo il piano di esecuzione effettivo generato per la query; consultandolo si osserva la parallelizzazione del task Clustered Index Scan.

* 1. -- Query data
  2. select
  3. FirstName
  4. ,LastName
  5. ,Phone
  6. ,AddressLine
  7. from
  8. dbo.Contact
  9. where
  10. (Phone like '059/21%')
  11. go
  12. 
  13. Figura 3 – Piano di esecuzione effettivo in ambiente di produzione

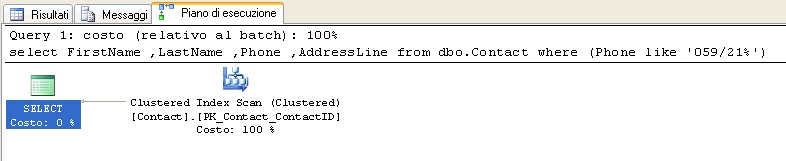
La query precedente rappresenta (ovviamente) un esempio, nel mondo reale l’istanza SQL Server si troverà a dover gestire migliaia di query al minuto contemporaneamente ad altre attività, le operazioni di monitoraggio e di tuning vorremmo venissero eseguite offline per non stressare ulteriormente la macchina e per lavorare in sicurezza su un database in ambiente di test. Come potrei simulare, in ambiente di test, la precedente query su dbo.Contact senza disporre dei dati veri e propri? Alcuni clienti (giustamente) non hanno piacere vengano presi i backup dei DB in produzione, oppure semplicemente il backup potrebbe essere centinaia di GB.

Una soluzione è rappresentata dall’esportazione di tutte le statistiche presenti sulla tabella dbo.Contact per la successiva importazione su un DB vuoto avente lo stesso schema del database DB\_Export\_Stats.

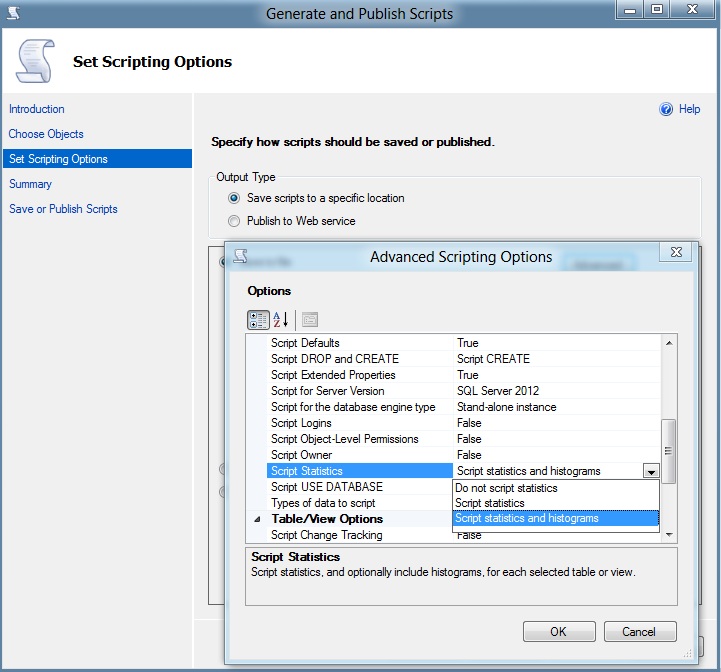
Procediamo quindi con il seutp del database DB\_Stats\_Only che conterrà la tabella dbo.Contact con lo stesso schema dell’omologa tabella sul database DB\_Export\_Stats, unica differenza: sul database DB\_Stats\_Only non verrà eseguito il caricamento dei contatti.

* 1. ------------------------------------------------------------------------
  2. -- Setup DB with statistics only
  3. ------------------------------------------------------------------------
  4. use [master];
  5. go
  6. if exists(select \* from sys.databases where name = 'db\_stats\_only')
  7. begin
  8. alter database [db\_stats\_only]
  9. set single\_user with rollback immediate;
  10. drop database [db\_stats\_only];
  11. end
  12. go
  13. -- Create database
  14. create database [db\_stats\_only];
  15. go
  16. ------------------------------------------------------------------------
  17. -- Setup table dbo.Contact
  18. ------------------------------------------------------------------------
  19. -- Change database context
  20. use [db\_stats\_only];
  21. go
  22. -- Create table dbo.Contact
  23. if object\_id('dbo.Contact', 'U') is not null
  24. drop table dbo.Contact;
  25. go
  26. create table dbo.Contact
  27. (
  28. ContactID int identity(1, 1) not null
  29. ,FirstName varchar(128) not null
  30. ,LastName varchar(128) not null
  31. ,AddressLine varchar(40) default 'AddressLine'
  32. ,City varchar(40) default 'City'
  33. ,PostalCode varchar(5) default 'PCode'
  34. ,Phone varchar(20) not null
  35. ,ModifiedDate datetime default getdate()
  36. );
  37. go
  38. ------------------------------------------------------------------------
  39. -- Setup constraint
  40. ------------------------------------------------------------------------
  41. alter table dbo.Contact
  42. add constraint PK\_Contact\_ContactID primary key (ContactID);
  43. go
  44. create index NCI\_Contact\_FirstName\_LastName on dbo.Contact
  45. (
  46. [Phone]
  47. )
  48. include
  49. (
  50. [FirstName]
  51. ,[LastName]
  52. );
  53. go

Eseguiamo, in ambiente di test, la stessa query eseguita in precedenza in ambiente di produzione (piano di esecuzione illustrato in Figura 3), come si può facilmente intuire il piano di esecuzione sarà diverso, il Query Optimizer, in questo caso (nessun record nella tabella dbo.Contact), sceglierà di eseguire una scansione dell’indice cluster, ma senza parallelizzare il task; il piano di esecuzione effettivo è illustrato in Figura 4.

* 1. use [db\_stats\_only];
  2. go
  3. -- Query data
  4. select
  5. FirstName
  6. ,LastName
  7. ,Phone
  8. ,AddressLine
  9. from
  10. dbo.Contact
  11. where
  12. (Phone like '059/21%');
  13. go
  14. 
  15. Figura 4 – Piano di esecuzione effettivo in ambiente di test

L’esportazione delle Statistiche può essere eseguita utilizzando il Wizard “Generazione guidata script di SQL Server” raggiungibile con un click destro del mouse sul database (in produzione) DB\_Export\_Stats, accedendo al menù attività e in seguito a “Genera script…”. Con alcuni semplici click, si accede alla finestra illustrata in Figura 5 dove si dovrà selezionare, per l’opzione “Script Statistiche”, il valore “Genera script per statistiche e istogrammi”.

* 1. 
  2. Figura 5 – Selezione del valore “Genera script per statistiche e istogrammi” per l’opzione “Script Statistiche”

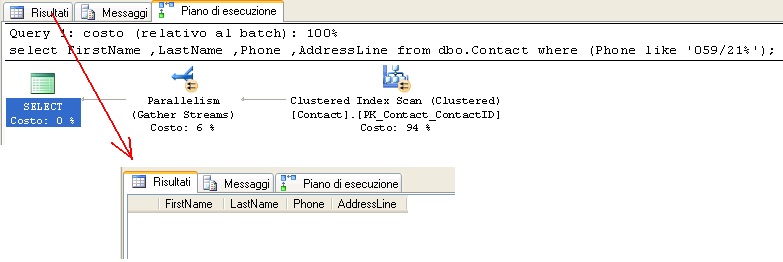
Scegliendo di ottenere l’output in una nuova finestra di query, SQL Server produrrà uno script simile a quello riportato di seguito:

* 1. SET ANSI\_PADDING OFF
  2. GO
  3. CREATE STATISTICS [AddressLine] ON [dbo].[Contact]([AddressLine]) WITH STATS\_STREAM = 0x01000...0F0000000000
  4. GO
  5. CREATE STATISTICS [City] ON [dbo].[Contact]([City]) WITH STATS\_STREAM = 0x01000...000000000
  6. GO
  7. CREATE STATISTICS [FirstName] ON [dbo].[Contact]([FirstName]) WITH STATS\_STREAM = 0x010000000100...00000000
  8. GO
  9. CREATE STATISTICS [LastName] ON [dbo].[Contact]([LastName]) WITH STATS\_STREAM = 0x010000...40420F0000000000
  10. GO
  11. CREATE STATISTICS [ModifiedDate] ON [dbo].[Contact]([ModifiedDate]) WITH STATS\_STREAM = 0x01000000010...000000000
  12. GO
  13. UPDATE STATISTICS [dbo].[Contact]([NCI\_Contact\_FirstName\_LastName]) WITH STATS\_STREAM = 0x010000000200...000040420F0000000000, ROWCOUNT = 1000000, PAGECOUNT = 6025
  14. GO
  15. UPDATE STATISTICS [dbo].[Contact]([PK\_Contact\_ContactID]) WITH STATS\_STREAM = 0x0100000001...0F0000000000, ROWCOUNT = 1000000, PAGECOUNT = 11061
  16. GO
  17. CREATE STATISTICS [PostalCode] ON [dbo].[Contact]([PostalCode]) WITH STATS\_STREAM = 0x010000000...420F0000000000
  18. GO

Osserviamo l’opzione non documentata STATS\_STREAM del comando CREATE STATISTICS.

STATS\_STREAM (di cui Microsoft non garantisce la compatibilità con le versioni future – è comunque presente in SQL Server 2012) accetta in input un valore esadecimale (troncato per semplicità nello script riportato in quest’articolo). Eseguendo i comandi CREATE e UPDATE STATISTICS, SQL Server sarà in grado di includere nelle statistiche, gli istogrammi che descrivono la distribuzione dei dati, il numero di righe contenute nelle tabelle ed il numero di pagine necessarie per la memorizzazione dei dati.

Dopo aver applicato le statistiche e aver resettato la Plan Cache, rieseguiamo la query descritta in precedenza in ambiente di test. Osserviamo che Query Optimizer cercherà di determinare, ancora una volta, il piano di esecuzione (sub)ottimale in funzione delle statistiche importate dal database DB\_Export\_Stats… deciderà quindi di parallelizzare gli operatori esattamente come avviene in produzione, lo illustra la Figura 6.

* 1. DBCC FREEPROCCACHE;
  2. go
  3. -- Query data
  4. select
  5. FirstName
  6. ,LastName
  7. ,Phone
  8. ,AddressLine
  9. from
  10. dbo.Contact
  11. where
  12. (Phone like '059/21%');
  13. go
  14. 
  15. 
  16. Figura 7 – Piano di esecuzione effettivo (grafico e XML) in ambiente di test

## Conclusioni

In quest’articolo abbiamo descritto come sia possibile analizzare il piano di esecuzione effettivo di una query eseguita in produzione, simulandola in ambiente di test, senza però fare il restore del DB o copiare il contenuto delle tabelle interessate.

## Pulizia dati

* 1. ------------------------------------------------------------------------
  2. -- Cleanup DB
  3. ------------------------------------------------------------------------
  4. use [master];
  5. go
  6. if exists(select \* from sys.databases where name = 'db\_export\_stats')
  7. begin
  8. alter database [db\_export\_stats]
  9. set single\_user with rollback immediate;
  10. drop database [db\_export\_stats];
  11. end
  12. go
  13. if exists(select \* from sys.databases where name = 'db\_stats\_only')
  14. begin
  15. alter database [db\_stats\_only]
  16. set single\_user with rollback immediate;
  17. drop database [db\_stats\_only];
  18. end
  19. go

#### di [Sergio Govoni](https://mvp.support.microsoft.com/profile/Sergio.Govoni) – Microsoft MVP ([blog](http://community.ugiss.org/blogs/sgovoni/))

* 1. [*Altri articoli di Sergio Govoni nella Libr*](http://sxp.microsoft.com/feeds/3.0/msdntn/TA_MSDN_ITA?contenttype=Article&author=Sergio%20Govoni)*ary* [](http://sxp.microsoft.com/feeds/3.0/msdntn/TA_MSDN_ITA?contenttype=Article&author=Sergio%20Govoni)