Home
Tags
Articles
Editorials
Stairways
Forums
Scripts
Videos
Blogs
QotD
Books
Ask SSC
SQL Jobs
Training
Authors
Active Threads
About us
Contact us
Newsletters
Write for us

Write For Us
Find out more

Rate this | Join the discussion | Add to briefcase | Print

Thank this author by sharing: 10

A Check and Foreign Key Constraint Improves Query Performance

By Sarvesh Singh, 2013/04/26 (first published: 2010/10/21)

In one of my training sessions I was asked, can check and foreign key constrain improve query performance? The answer to this question is 'Yes' and I will show you how in this article.

The optimizer uses foreign key constrains and check constraints to create more efficient query plan by skipping some part of the query plan because the optimizer can see that there is a foreign key constraint so it is not necessary to execute that part of the plan. Let's prove this with the help of an example. Consider the two tables below.

Related Articles

FORUM **Foreign Key Constraints and Performace!**
Do foreign keys constraints impact performance?

SCRIPT **Fix Foreign Key Constraint Naming**
This stored procedure can be used for applying a custom foreign key constraint naming

FORUM **Disable foreign constraint**
Disable foreign constraint

SCRIPT **Rename Foreign Key Constraints**
Rename Foreign Key Constraints in a consistent manner. Ideal for preventing constraint errors durin...

FORUM **Suggestion on Foreign Key Constraint**
Suggestion on whether to add Foreign Key Constraint or Not

Tags
foreign keys (fk)
performance tuning

Write For Us
Find out more

sql SOURCE CONTROL

Get your SQL database under version control now!

Learn why

redgate

```
Create Table sales
        (
               CustomerID INT Primary key
        );

Create Table SalesOrderdetail
        (
               SalesOrderID int Primary key,
               CustomerID int Not Null
                       Constraint FTCustomerID
                               References sales (Custome
        );
```

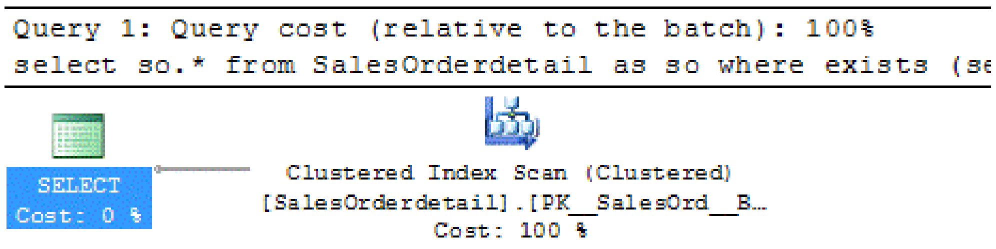
Insert Some values in the tables.

```
stomerID)
,3),(8,3),(9,3),(10,4),(11,4),(12,4),(13,5),(14,5),(15,5)
```

Let's look at what SQL Server does when there's a foreign key. Consider the following select query. Please include the actual execution plan by pressing Cntrl-M, or from the Query Menu, choose include actual execution plan.

```
select so.* from SalesOrderdetail as so
where exists (select * from sales as s
 where so.CustomerID=s.CustomerID)
```

If you see the execution plan below after you've run the above select statement, you will see that the optimizer doesn't even look at the sales table. Only salesorderdetail table is accessed. This is because the optimizer knows that it is no point executing the EXISTS operator as the foreign key constrain requires all sales to refer to an existing customer which is what the WHERE clause is doing.



```
Query 1: Query cost (relative to the batch): 100%
select so.* from SalesOrderdetail as so where exists (se
```

SELECT
Cost: 0 %
Clustered Index Scan (Clustered)
[SalesOrderdetail].[PK__SalesOrd__B…
Cost: 100 %

Fig. 1

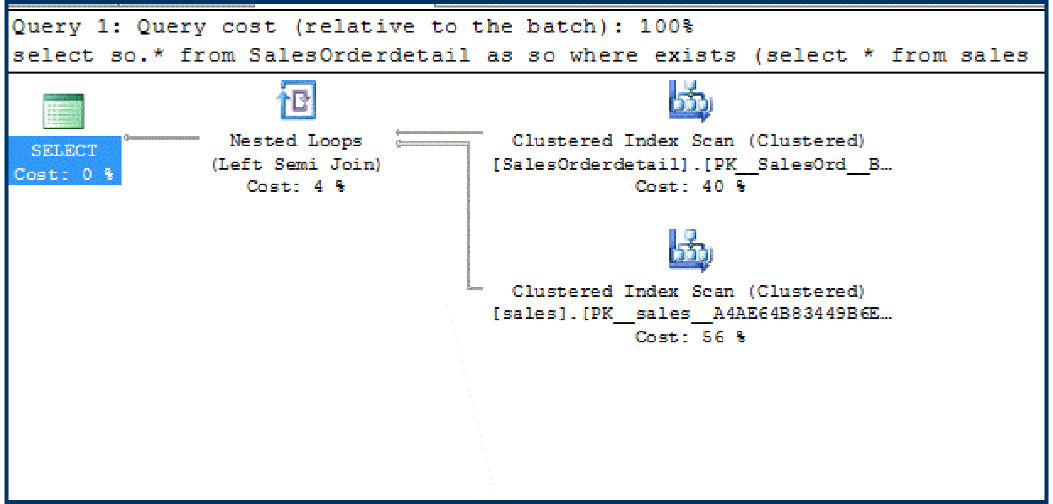
Now, let's turn off the foreign key constraint.

```
Alter Table SalesOrderdetail nocheck constraint FTCustome
```

Run the same select statement

```
select so.* from SalesOrderdetail as so
where exists (select * from sales as s
                                   where so.CustomerID=s.Cus
```

The execution plan below shows that the optimizer now executes the EXISTS operator to return only those salesorder rows that have a reference to the customer. Since we switched off the foreign key constraint, the optimizer wasn't sure whether salesorder had valid customer references. That's the reason why it had to execute the EXISTS operator. If you have a big table, this can make a big difference as far as performance is concerned.


```
Query 1: Query cost (relative to the batch): 100%
select so.* from SalesOrderdetail as so where exists (select * from sales
```
SELECT
Cost: 0 %
Nested Loops
(Left Semi Join)
Cost: 4 %
Clustered Index Scan (Clustered)
[SalesOrderdetail].[PK__SalesOrd__B…
Cost: 40 %
Clustered Index Scan (Clustered)
[sales].[PK__sales__A4AE64B83449B6E…
Cost: 56 %

Zoom in | Open in new window

Fig. 2

Let's turn on the foreign key again by executing the following statement.

```
.ter table SalesOrderdetail check constraint FTCustomerID;
```

Now if you run the same select statement again you will still see the execution plan shown in Fig. 2. Why is this happening? You have switched on the foreign key constraint, and it should now execute the plan in Fig. 1. But it's not doing that. This is because the foreign key constraint is 'not trusted'. The optimizer does not take into account a constraint that is not trusted. The foreign key is not trusted because a user might have inserted or updated a sales row with an invalid CustomerID. When you switch it on, it does not verify the previous data by default.

You can check the status of your foreign key constraint by running the following query.

```
_trusted from sys.foreign_keys where name= 'FTCustomerID'
```

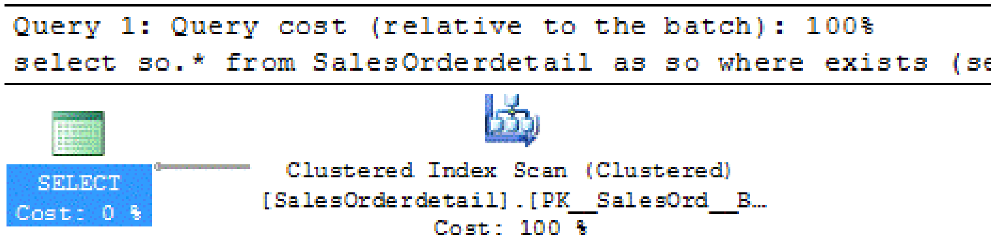
This will show you that the is_not_trusted column = 1 indicating that your foreign key constraint is not trusted. To make it trusted include the WITH CHECK option in your query as shown below.

```
SalesOrderdetail WITH CHECK check constraint FTCustomerID;
```

This checks that all rows in the table comply with the constraint before turning it on. If any rows do not comply with the constraint then an error message is returned and the Alter Statement is rolled back.

If you check the status of the constraint again, you will see that the is_not_trusted column shows 0. Now, when we run the first select statement we get the first execution plan as shown in Fig. 1.

```
select so.* from SalesOrderdetail as so
where exists (select * from sales as s
                                where so.CustomerID=s.Cus
```



```
Query 1: Query cost (relative to the batch): 100%
select so.* from SalesOrderdetail as so where exists (se
```

SELECT
Cost: 0 %
Clustered Index Scan (Clustered)
[SalesOrderdetail].[PK__SalesOrd__B...
Cost: 100 %

Let's make it a bit more interesting. Let's change the table structure of the members table such that TeamID column allows NULLs.

```
drop table SalesOrderdetail
go
Create Table SalesOrderdetail
         (
                 SalesOrderID int Primary key,
                 CustomerID int Null
                         Constraint FTCustomerID
                                 References sales (Custome
         );
```

Now if you run the same query against this new table structure, you will get execution plan shown in Fig. 2. This means that the optimizer is executing the EXISTS operator even if the foreign key constraint is trusted. To get back to the first execution plan (Fig. 1), which did not execute the EXISTS operator you will need to change the query as shown below.

```
select so.* from SalesOrderdetail as so
where exists (select * from sales as s
                                 where so.CustomerID=s.Cus
and so.customerID is not null
```

This informs the SQL server that no member with TeamID of NULL should be returned. This again brings you back to the first execution plan (Fig. 1) where it doesn't execute the EXISTS operator.

Conclusion

In this article I have showed that check and foreign constraints do not degrade performance but actually improves performance.

Rate this | Thank this author by sharing: 10
Join the discussion | Add to briefcase | Print

Total article view s: 21169 | View s in the last 30 days: 7970

Copyright © 2002-2013 Simple Talk Publishing. All Rights Reserved. Privacy Policy. Terms of Use. Report Abuse.