لغو نرمالسازی (Denormalize) برای اجرا

نمودار denormalize گاهی اجرای بهتری برای query ارائه می دهد، گرچه ممکن است به معنای ذخیره ی داده های غیرضروری باشد. تنها سوال که مطرح می شود این است که فضای اضافه استفاده شده به سود اجرا می افزاید.

توضیح

قبل از اینکه به مثال بپردازیم، اجازه بدهید اطمینان حاصل کنیم که ستون های متصل ما ایندکس شده هستند (همانطور که در بحث های پیشین مشاهده کردیم)، بنابراین اجرای نتایج به وسیله ی اسکن ها تغییری نمی کند. در اینجا عبارت های SQL برای ایجاد این ایندکس ها را مشاهده می کنید:

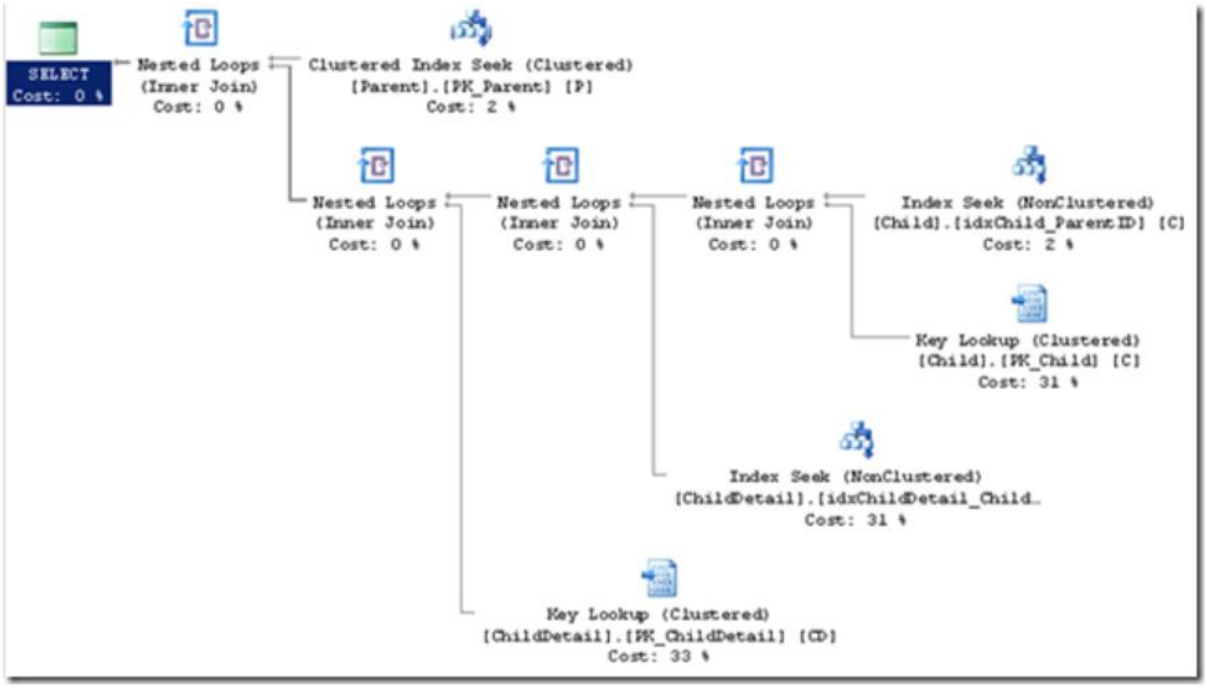
```
CREATE NONCLUSTERED INDEX idxChild_ParentID
ON [dbo].[Child] ([ParentID])

CREATE NONCLUSTERED INDEX idxChildDetail_ChildID
ON [dbo].[ChildDetail] ([ChildID])
```

برای تست اجرای هر دو نمودار denormalize و normalize از query ساده ی زیر استفاده می کنیم.

```
SELECT *
  FROM [dbo].[Parent] P INNER JOIN
     [dbo].[Child] C ON P.ParentID=C.ParentID INNER JOIN
     [dbo].[ChildDetail] CD ON C.ChildID=CD.ChildID
WHERE P.ParentID=32433
```

با دقت به explain plan متوجه می شویم که همانطوری رفتار می کند که مرد تردید ما بود، که در هنگام اتصال هر جدول در query از جستجوهای ایندکس (index seeks) و lookup ها استفاده می کند.



اکنون اجازه بدهید با انتقال داده ی جدول ChildDetail به جدول Child نوعی denormalization اجرا کنیم. ابتدا نیاز خواهیم داشت ستون های مورد نیاز را به جدول Child اضافه کنیم. سپس قبل از انتقال هر داده، نیاز به حذف محدودیت های کلید اولیه و کلید خارجی داریم و زمانی که داده انتقال داده می شود، می توانیم مجددا آن را ایجاد کنیم. عبارت های SQL زیر این عملکردها را اجرا می کنند.

```
ALTER TABLE [dbo].[Child] ADD [ChildDetailID] [bigint] NOT NULL
DEFAULT 0,[ExtraDataColumn] [bigint]

ALTER TABLE [dbo].[ChildDetail] DROP CONSTRAINT [FK_ChildDetail_Child]

ALTER TABLE [dbo].[Child] DROP CONSTRAINT [PK_Child]

INSERT INTO [dbo].[Child]
  SELECT C.ChildID,C.ParentID,C.IntDataColumn,C.VarcharDataColumn,
      CD.ChildDetailID,CD.ExtraDataColumn
   FROM [dbo].[Child] C INNER JOIN
     [dbo].[ChildDetail] CD ON C.ChildID=CD.ChildID

DELETE FROM Child where ChildDetailID=0
```

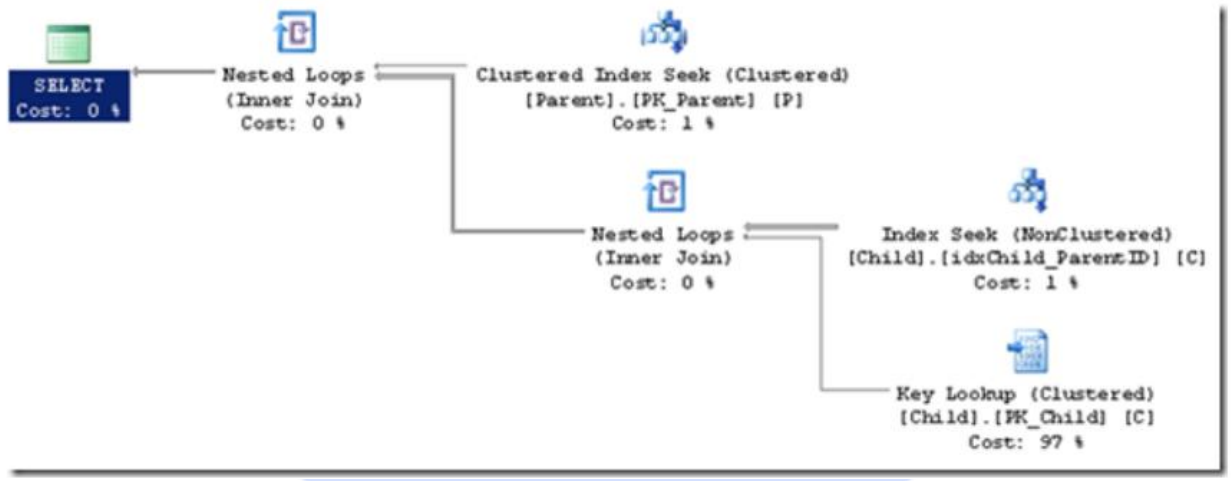
```
ALTER TABLE [dbo].[Child] ADD  CONSTRAINT [PK_Child] PRIMARY KEY CLUSTERED
  ([ChildID] ASC, [ChildDetailID] ASC)

DROP TABLE [dbo].[ChildDetail]
```

همچنین لازم خواهیم داشت query استفاده شده در بالا را آپدیت کنیم، چرا که دیگر نیازی به دسترسی به جدول ChildDetail نداریم. در اینجا query آپدیت شده را مشاهده می کنید:

```
SELECT *
  FROM [dbo].[Parent] P INNER JOIN
     [dbo].[Child] C ON P.ParentID=C.ParentID
 WHERE P.ParentID=32433
```

explain plan برای این query خیلی شبیه به نمونه ی اصلی آن می باشد، فقط در این مورد که دیگر اتصالی با جدول ChildDetail نداریم.



با نگاه به نتایج SQL Profiler از این query ها، یک مزیت بزرگ از برطرف کردن اتصال جدول ChildDetail مشاهده می کنیم. SQL Server خواندن های کمتری را اجرا می کند و زمان کل اجرا نیز بهبود پیدا کزده است.

	CPU	Reads	Writes	Duration
Normalized	0	365	0	75
Denormalized	0	250	0	5

ما باید نگاهی به مقدار فضای اضافه ای که استفاده می کنیم داشته باشیم، چرا که این امر در تصمیم گیری در مورد اجرای چنین تغییری مهم می باشد. عبارت SQL زیر مقدار فضایی را که هر کدام از جدول های شما روی دیسک اشغال می کنند را نشان می دهد.

```
SELECT o.name,SUM(reserved_page_count) * 8.0 / 1024 AS 'Size (MB)'
 FROM sys.dm_db_partition_stats ddps INNER JOIN
    sys.objects o ON ddps.object_id=o.object_id
WHERE o.name in ('Parent','Child','ChildDetail')
GROUP BY o.name
```

جدول زیر نتایج query بالا برای هر دو نمودار جدول normalize و denormalize نشان می دهد. همانطور که می بینید نمودار جدول denormalize فضایی بیشتر از 18mb استفاده نکرده است. تنها سوالی که مطرح می شود ارزش سود اجرایی فضای داده ی برکنار شده می باشد.

Table	Normalized Size (MB)	Denormalized Size (MB)
Parent	5.9	5.9
Child	151.6	679.2
ChildDetail	509.6	N/A
Total	667.1	685.1