Database$_2$ Project

Correspondence System

محمد باسل الشمالي

محمد العلوه

محمد غانم

آليات العمل:

- المراسلات تكون بين الأقسام والإدارات فقط، هذه المراسلات يُرسل بعضها كمراسلات شخصية للموظفين أو تعمم على كل الموظفين في الإدراة أو القسم.
- الموظف يعمل لدى قسم أو ادراة واحدة.
- المراسة الواردة ورقياً يمكن حفظها كصور.
- المراسة المرسلة والمراسة الواردة يمكن أن يكون لها خمس حالات: القبول-الرفض-التدقيق-الحذف-الأرشفة.
- المراسة الواردة الى موظف تمر بحالتين: الحذف-الأرشفة.

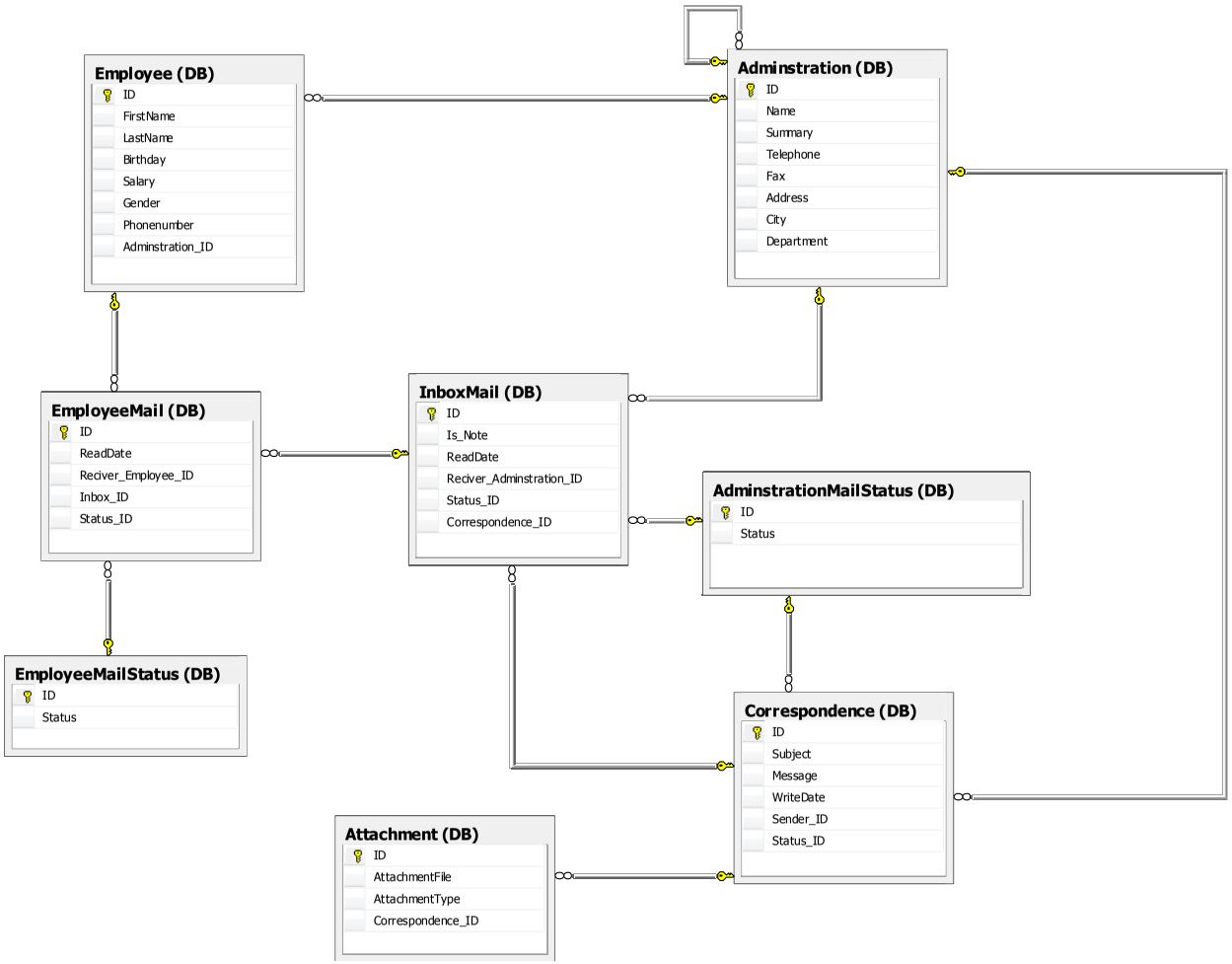
1. الطلب الأول جدول قاعدة البيانات ERD:

الادارة تحوي على عدة أقسام فالعلاقة بينهما واحد-كثير ولكن جمعناهما في جدول واحد ووضعنا في جدول الأدراة حقل اذا كانت قيمة هذا الحقل null فهذا يعني انه إدارة أما اذا لم يكن كذلك فهو قسم ويتبع إلى الادراة التي رقمها قيمة هذا الحقل.

الادراة ترسل أكثر من مراسلة فالعلاقة بينها وبين المراسلة هي واحد-كثير، المراسلة يمكن ان ترسل الى اكثر من إدراة يوجد جدول يجمع المرسل والمستقبل والمراسلة وهو الجدول Inbox Mail يحث يحوي رقم المرسل ورقم المستقبل ورقم المراسلة، حيث يعبر هذا الجدول عن مراسلة واردة إلى الإدارة أو القسم ويمكن تعميمها أو إرسالها إلى موظفين محددين.

المراسة المرسلة والمراسة المستقبلة ترتبط مع جدول الحالات، يساعد هذه الجدول باضافة حالات جديدة للمراسلات لاحقاً.

مراسلات الموظف تبنى على وارد الإدارة فهناك علاقة واحد-كثير بين وارد الإدراة ووارد الموظف، حيث أن الوارد إلى الإدارة يمكن أن يعمم على الموظفين ككل أو يمكن أن يرسل إلى أشخاص محددين.


Employee (DB)
ID
FirstName
LastName
Birthday
Salary
Gender
Phonenumber
Adminstration_ID
Adminstration (DB)
ID
Name
Summary
Telephone
Fax
Address
City
Department
EmployeeMail (DB)
ID
ReadDate
Reciver_Employee_ID
Inbox_ID
Status_ID
InboxMail (DB)
ID
Is_Note
ReadDate
Reciver_Adminstration_ID
Status_ID
Correspondence_ID
AdminstrationMailStatus (DB)
ID
Status
EmployeeMailStatus (DB)
ID
Status
Correspondence (DB)
ID
Subject
Message
WriteDate
Sender_ID
Status_ID
Attachment (DB)
ID
AttachmentFile
AttachmentType
Correspondence_ID

2. الطلب الثاني إنشاء قاعدة البيانات:

حجوم الملفات ومعدل الزيادة يعتمد على البيئة التي تستخدم قاعدة البيانات، سوف ننشئ مجموعة ملفات إضافة اسمها BigFiles لتخزين الملفات الخاصة بالمراسة.

```
CREATE DATABASE Correspondences_DB
ON PRIMARY
      ( NAME       = 'Correspondences System DB',
        FILENAME   = 'D:\Database\Project\DatabaseFiles\CSDB.mdf',
        SIZE       = 4 MB,
        FILEGROWTH = 1 MB
      ),

      (
        NAME        = 'HW_file1',
        FILENAME    = 'D:\Database\Project\DatabaseFiles\HW_file1.ndf,
        SIZE        = 2 MB,
        FILEGROWTH = 1 MB
      ),

      (
        NAME        = 'HW_file2',
        FILENAME    = 'D:\Database\Project\DatabaseFiles\HW_file2.ndf,
        SIZE        = 2 MB,
        FILEGROWTH = 1 MB
      ),

FILEGROUP HW_FileGroup
      (
        NAME        = 'HW_file3',
        FILENAME    = 'D:\Database\Project\DatabaseFiles\HW_file3.ndf,
        SIZE        = 2 MB,
        FILEGROWTH = 1 MB
      ),

      (
        NAME        = 'HW_file4',
        FILENAME    = 'D:\Database\Project\DatabaseFiles\HW_file4.ndf,
        SIZE        = 2 MB,
        FILEGROWTH = 1 MB
           ),
FILEGROUP BigFiles_FileGroup
      (
        NAME        = 'BigFiles',
        FILENAME   = 'D:\Database\Project\DatabaseFiles\BigFiles.ndf',
        SIZE        = 512 MB,
        FILEGROWTH = 50 MB
      )
LOG ON
      (
```

```
        NAME        = 'HW_log1',
        FILENAME    = 'D:\Database\Project\DatabaseFiles\HW_log1.ldf',
        SIZE        = 1 MB,
        FILEGROWTH  = 512 KB
      ),

      (
        NAME        = 'HW_log2',
        FILENAME    = 'D:\Database\Project\DatabaseFiles\HW_log2.ldf',
        SIZE        = 1 MB,
        FILEGROWTH  = 512 KB
      )
GO
CREATE SCHEMA DB;
```

3. الطلب الثالث إنشاء الجداول:

```
CREATE TABLE DB.Adminstration(
      ID                    INT IDENTITY(1,1),
      Name                  VARCHAR(50) not null,
      Summary               TEXT,
      Telephone             VARCHAR(15),
      Fax                   VARCHAR(15),
      Address               VARCHAR(255),
      City                  VARCHAR(50),
      Department            INT

      CONSTRAINT FK_Department
        FOREIGN KEY REFERENCES DB.Adminstration(ID) ,

      CONSTRAINT PK_Adminstration PRIMARY KEY (ID)
) ON HW_FileGroup
GO
-----------------------------------------------------------------
CREATE TABLE DB.Employee(
      ID                    INT IDENTITY(1,1),
      FirstName             VARCHAR(50) not null,
      LastName              VARCHAR(50) not null,
      Birthday              DATE,
      Salary                INT,
      Gender                CHAR,
      Phonenumber           VARCHAR(15),
      Adminstration_ID      INT not null

      CONSTRAINT FK_Adminstration_ID
       FOREIGN KEY DB.Adminstration(ID) ON DELETE CASCADE
```

```
    CONSTRAINT PK_Employee
       PRIMARY KEY (ID),

    CONSTRAINT CK_Gender_Employee
       CHECK (Gender in ('M','F'))


     CONSTRAINT CH_SALARY
       CHECK (Salary >= 0)
) ON HW_FileGroup
GO
-----------------------------------------------------------------
CREATE TABLE DB.AdminstrationMailStatus(
     ID                        INT IDENTITY(1,1),
     Status                    VARCHAR(50) not null,

     CONSTRAINT PK_AdminstrationMailStatus
       PRIMARY KEY (ID)
) ON HW_FileGroup
GO
-----------------------------------------------------------------
CREATE TABLE DB.Correspondence(
     ID              INT IDENTITY(1,1),
     Subject         VARCHAR(255) not null,
     Message         VARCHAR(max) not null,
     WriteDate       DATETIME not null,
     Sender_ID       INT not null

     CONSTRAINT FK_Sender_ID
       FOREIGN KEY DB.Adminstration (ID) ON DELETE CASCADE,

     Status_ID       INT not null

     CONSTRAINT FK_Corrs_Status_ID
       FOREIGN KEY REFERENCES DB.AdminstrationMailStatus (ID)

     CONSTRAINT PK_Correspondence
       PRIMARY KEY (ID)

) ON HW_FileGroup
GO
-----------------------------------------------------------------
CREATE TABLE DB.Attachment (
     ID                   INT IDENTITY(1,1),
     AttachmentFile       VARBINARY,
     AttachmentType       VARCHAR(50),
     Correspondence_ID    INT not null
```

```
    CONSTRAINT FK_Corr_ID
       FOREIGN KEY DB.Correspondence (ID) ON DELETE CASCADE
     CONSTRAINT PK_Attachment
        PRIMARY KEY (ID)
) ON BigFiles_FileGroup
GO
-----------------------------------------------------------------
CREATE TABLE DB.InboxMail(
     ID                          INT IDENTITY(1,1),
     Is_Note                     BIT not null,
     ReadDate                    DATETIME,

     Reciver_Adminstration_ID INT not null

     CONSTRAINT FK_Reciver_ID
       FOREIGN KEY REFERENCES DB.Adminstration(ID),

     Status_ID                   INT not null

     CONSTRAINT FK_AInbox_Status_ID
       FOREIGN KEY REFERENCES DB.AdminstrationMailStatus(ID),

     Correspondence_ID        INT not null

     CONSTRAINT FK_Correspondence_ID
       FOREIGN KEY REFERENCES DB.Correspondence(ID)

     CONSTRAINT PK_InboxMail
       PRIMARY KEY (ID)
)  ON HW_FileGroup
GO
-----------------------------------------------------------------
CREATE TABLE DB.EmployeeMailStatus(
     ID                  INT IDENTITY(1,1),
     Status              VARCHAR(50) not null

     CONSTRAINT PK_EmployeeMailStatus
       PRIMARY KEY (ID)
)  ON HW_FileGroup
GO
-----------------------------------------------------------------
CREATE TABLE DB.EmployeeMail (
     ID                          INT IDENTITY(1,1),
     ReadDate                    DATETIME,
     Reciver_Employee_ID      INT not null
```

```
      CONSTRAINT FK_Empolyee_ID
        FOREIGN KEY REFERENCES DB.Employee(ID),

      Inbox_ID                  INT not null

      CONSTRAINT FK_EInobx_ID
        foreign key references DB.InboxMail(ID),

      Status_ID                 INT not null

      CONSTRAINT FK_Status_ID
        FOREIGN KEY REFERENCES DB.EmployeeMailStatus(ID)

      CONSTRAINT PK_EmployeeMail
        PRIMARY KEY (ID)
)  ON HW_FileGroup
GO
```

المفاتيح الرئيسية:

جميع الجداول تحوي حقل ID هو عبارة عن مفتاح رئيسي PK.

المفاتيح الثانوية:

- جدول الإدارة يحوي مفتاح ثانوي هو رقم الإدارة الذي يتبع له القسم.
- جدول الموظف يحوي مفتاح ثانوي هو رقم للإدارة أو القسم.
- جدول المراسلة يحوي 2 حقول ثانوية هم: رقم القسم أو الإدارة المرسلة، رقم حالة المراسلة.
- جدول الوارد للإدارة أو القسم يحوي 3 مفاتيح ثانوية: رقم المراسلة، رقم الادراة الوراد إليها، رقم حالة المراسلة.
- جدول الوارد للموظف يحوي 3 مفاتيح ثانوية: رقم الوارد الى الإدارة، رقم الموظف، رقم حالة الوارد إلى الموظف.

4. الطلب الرابع Indexes:

1. Non Clustered Index:

الاستعلام هو عرض جميع أرقام هواتف الإدارات والأقسام التي تبدأ بالرقمين "34"

```
select Telephone from DB.Adminstration  where (Telephone like
'34%') order by (Telephone);
```

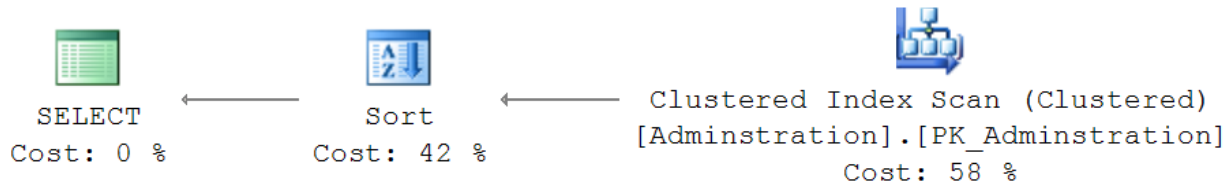
عند تنفيذ الاستعلام السابق ينتج:

Physical Logial Reads

```
Table 'Adminstration'. Scan count 1, logical reads 20, physical reads
6, read-ahead reads 8, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

SQL Server Execution Times:
   CPU time = 15 ms,  elapsed time = 42 ms.
```

Exuction Plan



نضيف فهرس مع معامل ملأ % 60 :

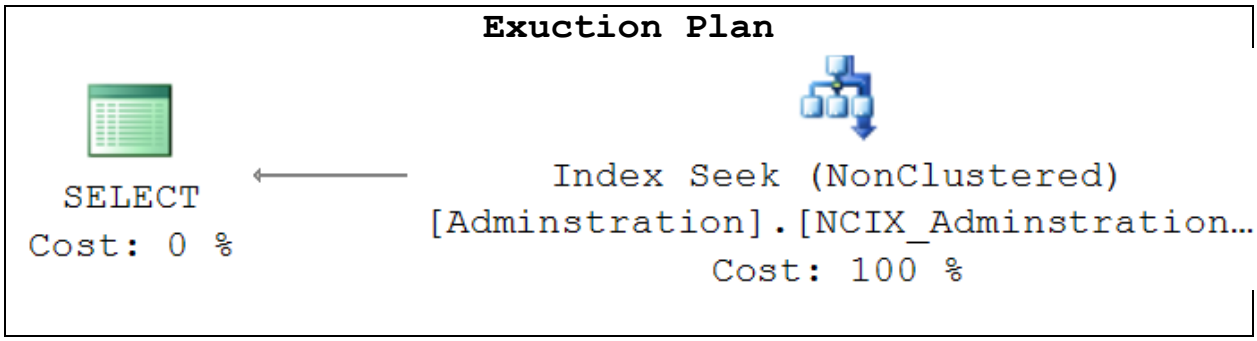
```
create nonclustered index NCIX_Adminstration_Telephone on
DB.Adminstration(Telephone) with (fillfactor = 60);
```

وننفذ الاستعلام السابق ينتج:

Physical Logical Reads

```
Table 'Adminstration'. Scan count 1, logical reads 2, physical reads
2, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

SQL Server Execution Times:
   CPU time = 15 ms,  elapsed time = 34 ms.
```



التوضيح:

قبل انشاء الفهرس استخدم Optimizer المفتاح الأولي كفهرس وبعدها عملية فرز وعدد العمليات الفيزيائية 6 والمنطقية 20.

بعد انشاء الفهرس نلاحظ إنخفاض عدد العمليات الفيزيائية والمنطقية إلى 2 فقط نتيجة استخدام الفهرس على رقم الهاتف ولم يجري عملية فرز لأن المعطيات تم الوصول إليها بشكل مرتب.

أما عن تأثير معامل الملئ فكلما زاد، تزداد المعطيات المخزنة في ورقة الفهرس وبالتالي في حال اضافة أو تعديل سوف يؤدي ذلك إلى شطر الورقة وعملية الشطر مكلفة زمنياً، أما في حال نقصانه فان حجم الفهرس سوف يزداد وتقل المعطيات المخزنة في الورقة مما يتيح مساحة فارغة لاضافة معطيات جديدة.

ولمقارنة حجم الفهرس سوف نستخدم التعليمة التالية:

```
EXEC sp_spaceused 'DB.Adminstration'
```

معامل الملئ = 1

	name	rows	reserved	data	index_size	unused
1	Adminstration	1100	2664 KB	256 KB	2264 KB	144 KB

معامل الملئ = 60

	name	rows	reserved	data	index_size	unused
1	Adminstration	1100	408 KB	256 KB	88 KB	64 KB

معامل الملئ = 100

	name	rows	reserved	data	index_size	unused
1	Adminstration	1100	392 KB	256 KB	72 KB	64 KB

نلاحظ حجم المعطيات نفسه ولكن الاختلاف بحجم الفهرس.

2. Clustered Index:

الاستعلام هو:

```
SELECT * FROM DB.Adminstration WHERE (ID > 35 and ID < 100 and
not (ID in (1,2,3) or ID in(73)));
```

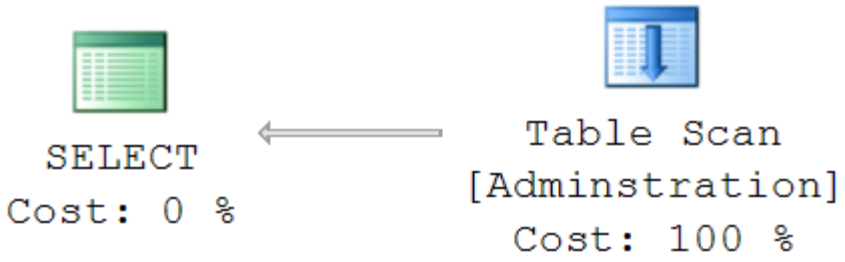
في حال لم يتم بناء أي PK سينتج ما يلي:

Physical Logical Reads

```
Table 'Adminstration'. Scan count 1, logical reads 19, physical reads
1, read-ahead reads 19, lob logical reads 126, lob physical reads 2,
lob read-ahead reads 0.

SQL Server Execution Times:
  CPU time = 15 ms,  elapsed time = 238 ms.
```

Exuction Plan



وبعد بناء الفهرس:

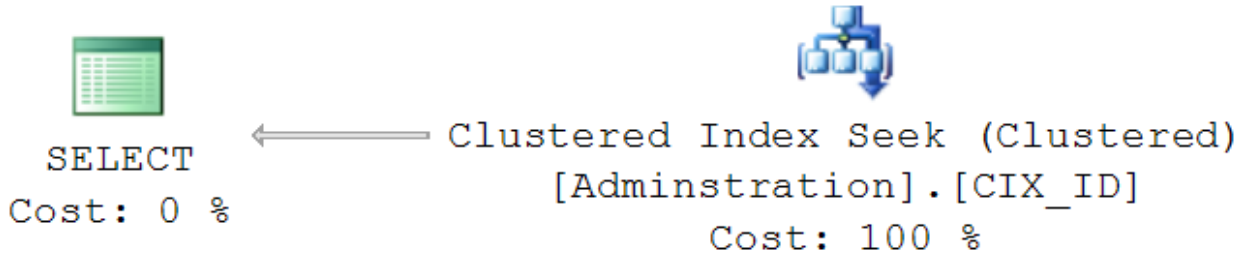
```
CREATE CLUSTERED INDEX CIX_ID ON DB.Adminstration (ID);
```

Physical Logical Reads

```
Table 'Adminstration'. Scan count 1, logical reads 4, physical reads
2, read-ahead reads 2, lob logical reads 126, lob physical reads 2,
lob read-ahead reads 0.

SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 189 ms.
```

Exuction Plan



تم اختيار حقل ID ليكون فهرس منعقد لأنه نوعه Int وبالتالي لا يأخذ حجم كبير ولا يعدل لانه في حال التعديل سوف يؤدي ذلك إلى تعديل شجرة الفهرس مما يعني كلفة أكثر ولأنه يزيد بمقدار ثابت identity مما يعني سهولة في الاضافة ضمن شجرة الفهرس فالاضافة تتم ضمن الصفحة نفسها.

وفي حال كان الاستعلام يحوي في عبارة where حقل أو أكثر غير الحقول التي بُنِيَ عليه الفهرس المعنقد فكلا الحالتين متكافئتين أي كلاهما Scan للجدول في حال عدم وجود ترتيب.

في حال كان الاستعلام يحوي بالإضافة الى حقل الفهرس حقول آخرى كما في الاستعلام التالي:

```
SELECT * FROM DB.Adminstration WHERE Telephone like '34%'  and
(ID > 35 and ID < 100 and not (ID in (1,2,3) or ID in(73)));
```

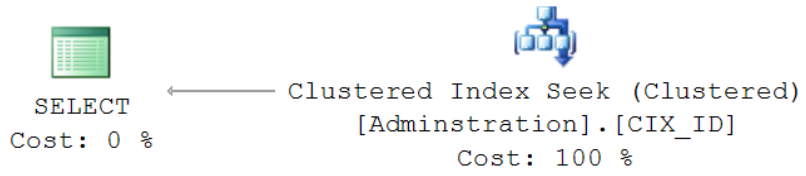
ومع وجود الفهرس السابق ينتج مايلي:

Physical Logical Reads

```
Table 'Adminstration'. Scan count 1, logical reads 4, physical reads
2, read-ahead reads 2, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 37 ms.
```

Exuction Plan



Predicate
[Correspondences_DB].[DB].[Adminstration].[Telephone] like '34%' AND ([Correspondences_DB].[DB].[Adminstration].[ID] <(73) OR [Correspondences_DB].[DB].[Adminstration].[ID]> (73))
Object
[Correspondences_DB].[DB].[Adminstration].[CIX_ID]
Output List
[Correspondences_DB].[DB].[Adminstration].Telephone
Seek Predicates
Seek Keys[1]: Start: [Correspondences_DB].[DB].[Adminstration].ID > Scalar Operator((35)), End: [Correspondences_DB].[DB].[Adminstration].ID < Scalar Operator((100))

التوضيح:

لم يقم Optimizer باختيار عملية Scan للجدول بل استخدم Seek على الفهرس وذلك لوجد شروط تساعده بالتجوال في شجرة الفهرس (الشروط على ID في الاستعلام السابق) ونلاحظ من الصورة الثانية أن الشرط المطبق في الوصل الى المعطيات من خلال الفهرس هو $35 < ID < 100$ أما الشرط المطبق للاختيار من نتائج الفهرس هو الذي يحوي Telephone like '34%'.

3. Covered Index:

الاستعلام يجلب الاسم الأول والأخير للموظفين الذين يبدأ اسمهم الأول بِ "D" واسمهم الأخير بِ "C" وراتبهم يزيد عن 5000.

```
SELECT FirstName + ' ' + LastName AS 'Full Name',Salary AS
'Employee Salary' FROM DB.Employee WHERE FirstName like 'D%' and
LastName like 'C%' and Salary > 5000 ORDER BY FirstName;
```

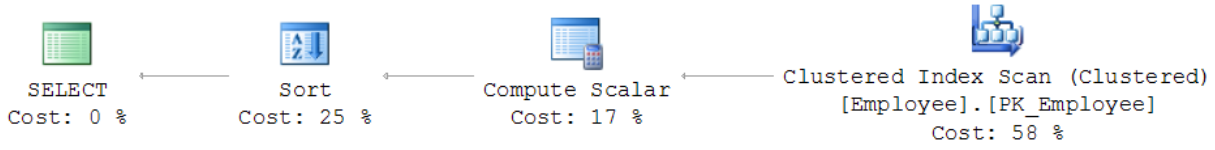
بعد تنفيذ الاستعلام ينتج ما يلي:

Physical Logical Reads

```
Table 'Employee'. Scan count 1, logical reads 28, physical reads 4,
read-ahead reads 26, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

SQL Server Execution Times:
  CPU time = 0 ms,  elapsed time = 4 ms.
```

Exuction Plan



سوف نضع سوف نضع الفهرس المغطي التالي:

```
CREATE NONCLUSTERED INDEX COIX_Name_Salary ON
DB.Employee(FirstName,lastName,Salary);
```

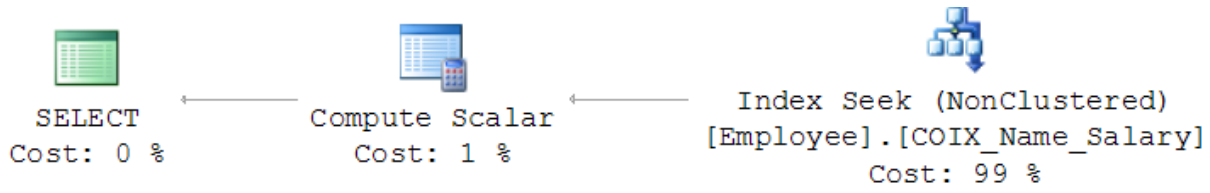
بعد وضع الفهرس وبعد تنفيذ الاستعلام:

Physical Logical Reads

```
Table 'Employee'. Scan count 1, logical reads 4, physical reads 2,
read-ahead reads 2, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 1 ms.
```

Exuction Plan



التوضيح:

- في الحالة الاولى العملية كانت Scan على كامل الجدول نلاحظ عدد العمليات كبير ووجود عملية فرز، في الحالة الثانية العملية عملية Seek على الفهرس دون عملية فرز.
- أن وجود حقل غير الحقول الواردة في الفهرس المغطى في عبارة Where أو عبارة Select سوف يؤدي إلى حتماً على عملية Key Lookup إضافية.
- في الفهرس المغطى يمكن إجراء عملية ترتيب دون أن يقوم ال Optimizer بعملية فرز مستقلة في حال كان الترتيب على جميع حقول الفهرس المغطى وبنفس الترتيب أو في حال الترتيب فقط على أول حقل من حقول الفهرس المغطى.
- في الاستعلام السابق كانت العلاقة بين التعابير المنطقية لعبارة Where هي And ولكن في حال كانت احداها هي Or نلاحظ أن ال Optimizer لا يعود باستطاعته استخدام الفهرس المغطى للحصول على معطيات الاستعلام وبالتالي عليه القيام بعملية Scan للجدول.
- في كانت عبارة Like تحوي المعامل % في البداية من أجل الحقل الأول من عبارة Where لا يمكن لل Optimizer أن يقوم بعملية المقارنة وبالتالي لا يمكنه استخدام الفهرس، هذه الحالة تنطبق على جميع أنواع الفهارس وليس فقط على الفهرس المغطى.

4. Including Index:

الاستعلام مشابه للاستعلام السابق:

```
SELECT FirstName,LastName,Salary FROM DB.Employee WHERE Salary
Between 5000 and 8000 and FirstName like 'A%';
```

ولكن الفهرس سوف يكون:

```
CREATE NONCLUSTERED INDEX IIX_Name ON DB.Employee(Salary)
include(FirstName,LastName);
```

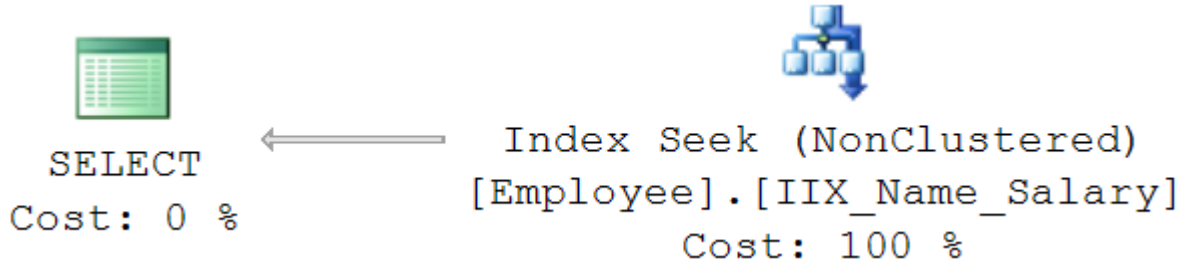
بعد انشاء الفهرس وتنفيذ الاستعلام ينتج ما يلي:

Physical Logical Reads

```
Table 'Employee'. Scan count 1, logical reads 17, physical reads 2,
read-ahead reads 15, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 3 ms.
```

Exuction Plan



التوضيح:

الفرق بين الفهرس المغطى والفهرس المتضمن هو أن الفهرس المغطى يخزن الأعمدة ضمن الشجرة في العقد الداخلية وفي الأوراق أما الفهرس المتضمن يخزن الأعمدة في أوراق شجرة البحث فقط.

وفي حال وجود حقل من الحقول المضافة ضمن عبارة Select لا يأثر هذه على كلفة الاستعلام تبقى العملية هي عملية Seek، أما في حال وجود الحقل ضمن عبارة Where (رقم الهاتف في الاستعلام السابق) أيضاً تبقى العملية عملية Seek ولكن يضاف Predicate لتحقيق ما هو مطلوب كما ذكرنا سابقاً في حالة مشابهة.

5. Filter Index:

الاستعلام يجلب المراسلات التي كتبت منذ بداية سنة 2015:

```
SELECT CONVERT(DATE,writeDate) AS 'Write Date',Message  FROM
DB.Correspondence WHERE  WriteDate > '2015-01-01';
```

لفهم أهمية هذا النوع من الفهارس سوف نفترض وجود الفهرس التالي:

```
CREATE NONCLUSTERED INDEX FIX_Coooespondence_Date On
DB.Correspondence(WriteDate) include(Sender_ID);
```

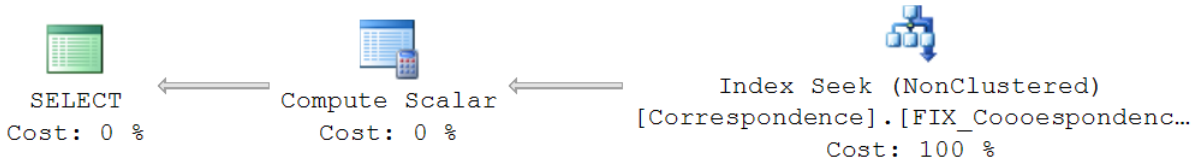
عند تنفيذ الاستعلام مع وجود الفهرس السابق ينتج مايلي:

Physical Logical Reads

```
Table 'Correspondence'. Scan count 1, logical reads 2, physical reads
2, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 56 ms.
```

Exuction Plan



استخدم Optimizer الفهرس الذي وضعناه ولكن الفهرس بُني على كل الجدول وبوجود الفهرس المرشح يمكننا بناء الفهرس على جزء من الجدول وبالتالي تقليل الحجم وتسريع الأداء.

نبني الفهرس المرشح فقط بإضافة عبارة Where على الفهرس السابق:

```
CREATE NONCLUSTERED INDEX FIX_Coooespondence_Date ON
DB.Correspondence(WriteDate) include(Sender_ID) WHERE (WriteDate
> '2010-01-01');
```

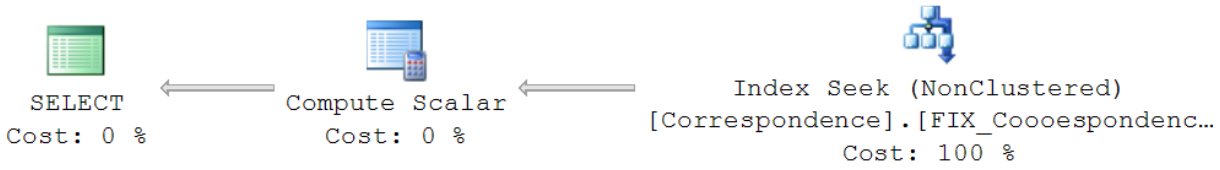
وبعد بناء الفهرس وتنفيذ الاستعلام:

Physical Logical Reads

```
Table 'Correspondence'. Scan count 1, logical reads 2, physical reads
2, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.
```

```
SQL Server Execution Times:
   CPU time = 0 ms,  elapsed time = 38 ms.
```

Exuction Plan



التوضيح:

نلاحظ الاختلاف في الأداء.

وبمقارنة حجم الفهرس، في الحالة الأولى يكون الحجم:

	name	rows	reserved	data	index_size	unused
1	Correspondence	1989	264 KB	176 KB	80 KB	8 KB

أما في الحالة الثانية يكون الحجم:

	name	rows	reserved	data	index_size	unused
1	Correspondence	1989	240 KB	176 KB	56 KB	8 KB

يجب ملاحظة أنه في حال استخدام الحقل أو مجموعة الحقول التي بنى فيها الفهرس المرشح بمجال غير المجال المحدد عند بناء الفهرس فان العملية تتحول من Seek إلى Scan.

5. السؤال الخامس + السادس:

سوف نختار استعلام مختلف وسوف نبني فهرس مناسب له، الاستعلام هو جلب جميع المراسلات الصادرة بعد تاريخ معين من الإدرات والأقسام التي عنوانها يساوي قيمة معينة:

```
SELECT a.Name AS 'Sender Name',a.Address AS 'Sender Address',c.WriteDate AS 'Send Date',c.Message AS 'Correspondence Message' from DB.Adminstration as a join DB.Correspondence AS c on (a.ID = c.Sender_ID) WHERE (Address = 'Address of Adminstration #4' and WriteDate > '2005-01-01');
```

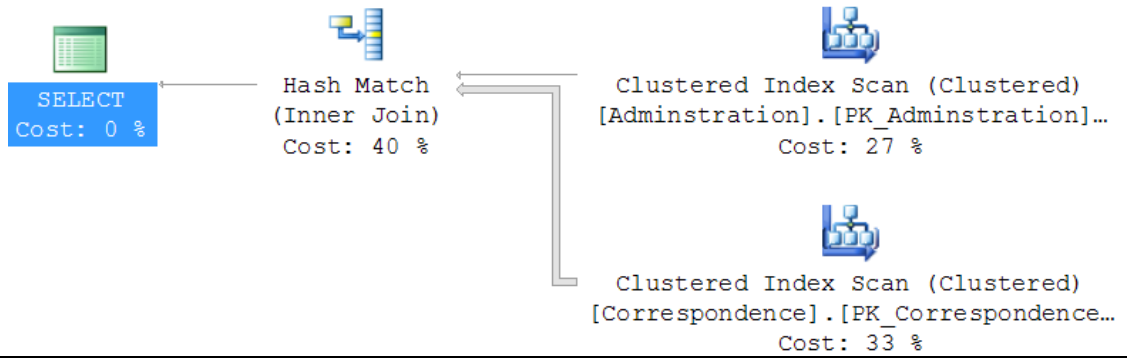
وقبل اضافة اي فهرس وتنفيذ الاستعلام ينتج ما يلي:

Physical Logical Reads

```
Table 'Correspondence'. Scan count 1, logical reads 24, physical
reads 4, read-ahead reads 22, lob logical reads 0, lob physical reads
0, lob read-ahead reads 0.
Table 'Adminstration'. Scan count 1, logical reads 20, physical reads
5, read-ahead reads 18, lob logical reads 0, lob physical reads 0,
lob read-ahead reads 0.

SQL Server Execution Times:
 CPU time = 16 ms,  elapsed time = 79 ms.
```

Exuction Plan



لجعل هذا الاستعلام أكثر فعالية يجب اضافة الفهرسين التالين:

```
CREATE NONCLUSTERED INDEX NCIX_Date_Corrspondence ON
DB.Correspondence(Sender_ID) include(Message,WriteDate);

CREATE NONCLUSTERED INDEX NCIX_Address_Adminstration ON
DB.Adminstration(Address) include(Name);
```

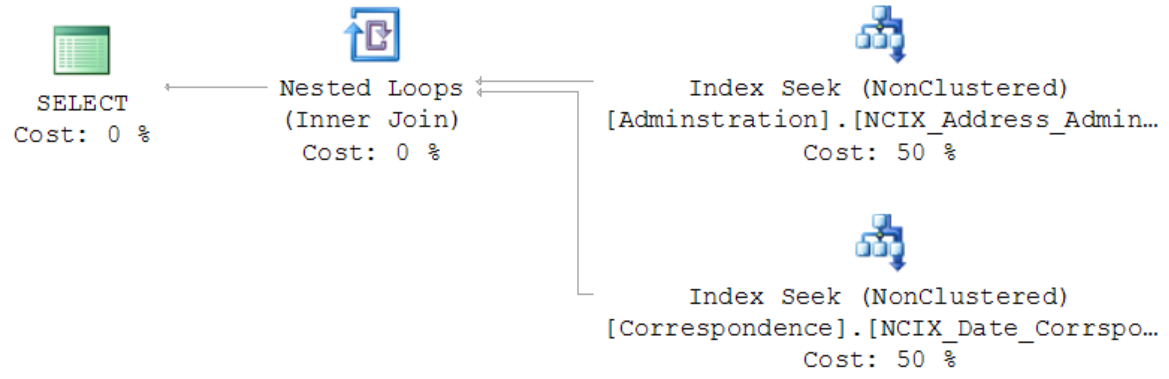
وعند تنفيذ الاستعلام ينتج ما يلي:

Physical Logical Reads

```
Table 'Correspondence'. Scan count 1, logical reads 2, physical reads
2, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.
Table 'Adminstration'. Scan count 1, logical reads 2, physical reads
2, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

SQL Server Execution Time:
    CPU time = 0 ms,  elapsed time = 2 ms.
```

Exuction Plan



التوضيح:

نلاحظ الفرق الواضح في الأداء بدون وجود الفهارس ومع وجود الفهارس، حيث تحولت عملية الدمج بين الجدولين من Merge ألى Nested وذلك لأن الفهارس أتاحت إمكانية إنتقاء الأسطر المطلوبة من كل جدول.

فكرة جلب أسطر معينة أسرع من غيرها هي كالتالي: في حال وجود استعلام يعرض معطيات إلى المستخدم ولا نريد جعل التطبيق ينتظر فترة معينة (مهما كانت صغيرة) لجلب كافة المعطيات، في هذه الحالة يهمنا هو أداء التطبيق وليس أداء قاعدة المعطيات عندها يمكن استخدام خيار Fast ارشاد Optimizer للتركيز جلب مقدار معين من المعطيات بأسرع وقت ممكن، لنأخذ الاستعلام السابق كمثال -سوف نحذف الفهارس المُنشأة عليه-وفي حال كنا نريد جلب أول سطرين فقط أسرع فأن الاستعلام يصبح كالتالي:

```
SELECT     a.Name AS 'Sender Name',
           a.Address AS 'Sender Address',
           c.WriteDate AS 'Send Date',
           c.Message AS 'Correspondence Message'
FROM DB.Adminstration AS a join DB.Correspondence AS c
     ON (a.ID = c.Sender_ID)
WHERE (Address = 'Address of Adminstration #4' and
WriteDate > '2005-01-01') OPTION (FAST 2);
```

وعند تنفيذه ومقارنة أداءه مع نفس الاستعلام دون خيار Fast ينتج ما يلي:

Physical Logical Reads without fast option

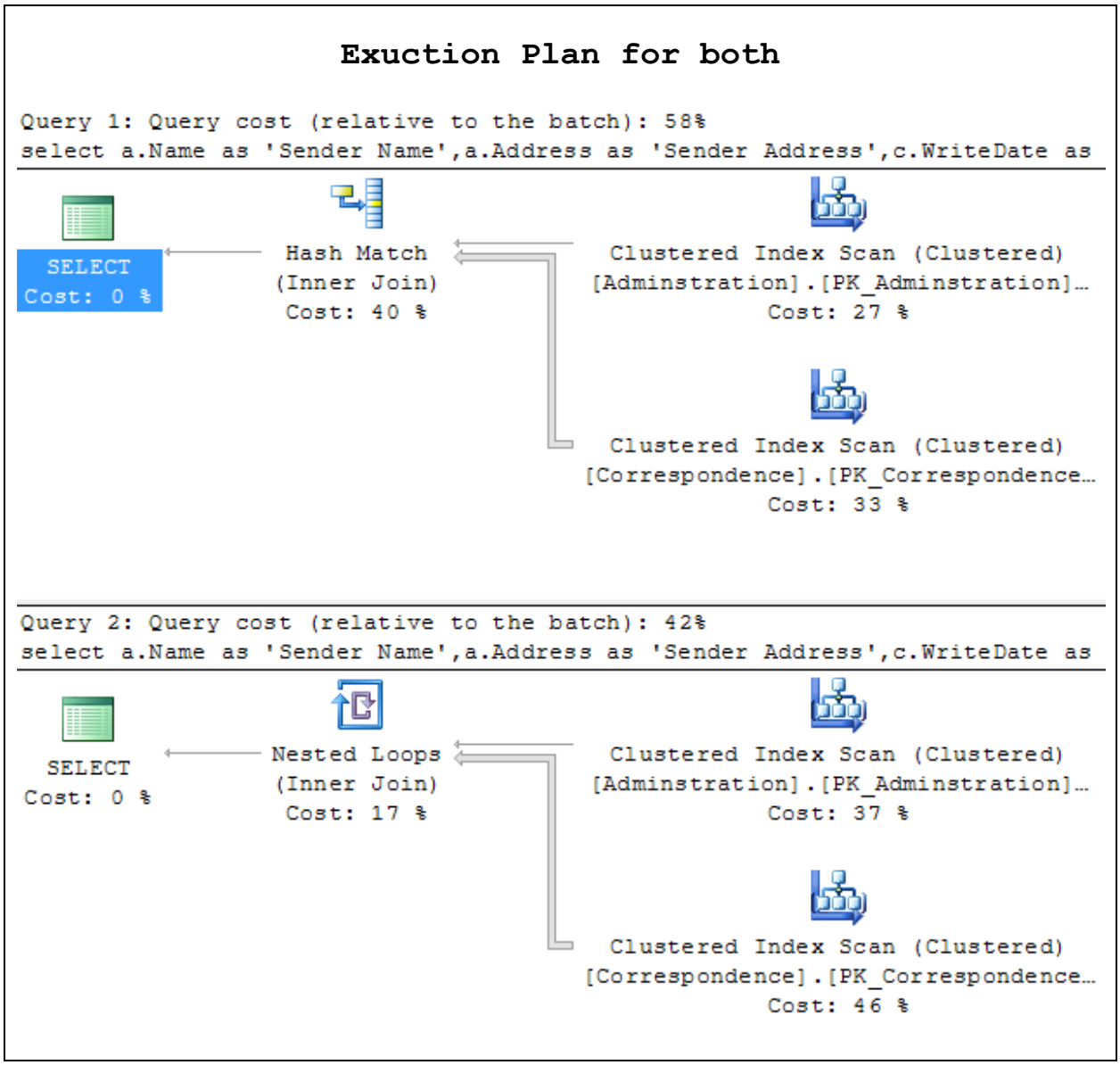
```
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0,
read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.
Table 'Correspondence'. Scan count 1, logical reads 24, physical
reads 0, read-ahead reads 0, lob logical reads 0, lob physical reads
0, lob read-ahead reads 0.
Table 'Adminstration'. Scan count 1, logical reads 20, physical reads
0, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 16 ms,  elapsed time = 6 ms.
```

Physical Logical Reads with fast option

```
Table 'Correspondence'. Scan count 1, logical reads 24, physical
reads 2, read-ahead reads 22, lob logical reads 0, lob physical reads
0, lob read-ahead reads 0.
Table 'Adminstration'. Scan count 1, logical reads 20, physical reads
2, read-ahead reads 18, lob logical reads 0, lob physical reads 0,
lob read-ahead reads 0.

 SQL Server Execution Times:
   CPU time = 16 ms,  elapsed time = 10 ms.
```



نلاحظ تأثير خيار Fast حيث أنه جعل عملية جمع الجدول تتم وفق ال Nested Loops عوضاً عن Hash Match.

7. الطلب السابع الدمج:

1. Nested Loop Join:

الاستعلام هو:

```
SELECT FirstName + ' ' + LastName, A.Name  FROM DB.Adminstration
AS A join DB.Employee AS E ON (A.ID = E.Adminstration_ID ) WHERE
(A.Department = 61 and Salary > 5000 and E.Gender = 'M');
```

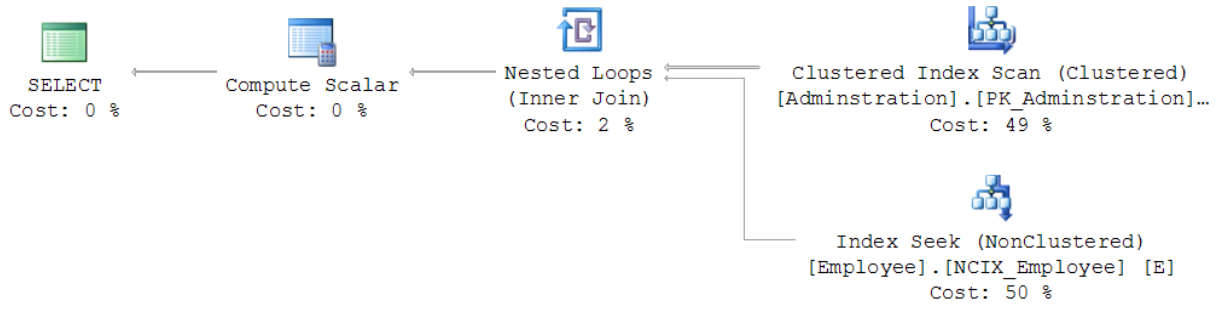
وبعد انشاء الفهارس المناسبة وتنفيذ الاستعلام ينتج ما يلي:

Physical Logical Reads

```
Table 'Correspondence'. Scan count 1, logical reads 2, physical reads
2, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.
Table 'Adminstration'. Scan count 1, logical reads 2, physical reads
2, read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.

SQL Server Execution Time:
    CPU time = 0 ms,  elapsed time = 2 ms.
```

Exuction Plan



الفهرس المضاف هو:

```
CREATE NONCLUSTERED INDEX NCIX_Employee ON
DB.Employee(Adminstration_ID) include
(FirstName,LastName,Salary,Gender);
```

التوضيح:

استخدم Nested Loops وذلك لأن الجدول الأول عدد أسطره قليلة نتيجة وجود Where في الاستعلام حددت جزء صغير منه والجدول الثاني يملك فهرس مناسب يحوي جميع الحقول المستخدمة في الاستعلام.

الفهارس التي تم دمجها:

NCIX_Employee - PK_Adminstration

فلولا وجود هذا الفهرس فأن عملية الدمج سوف تتم بواسطة Hash Join.

2. Hash Join:

الاستعلام هو:

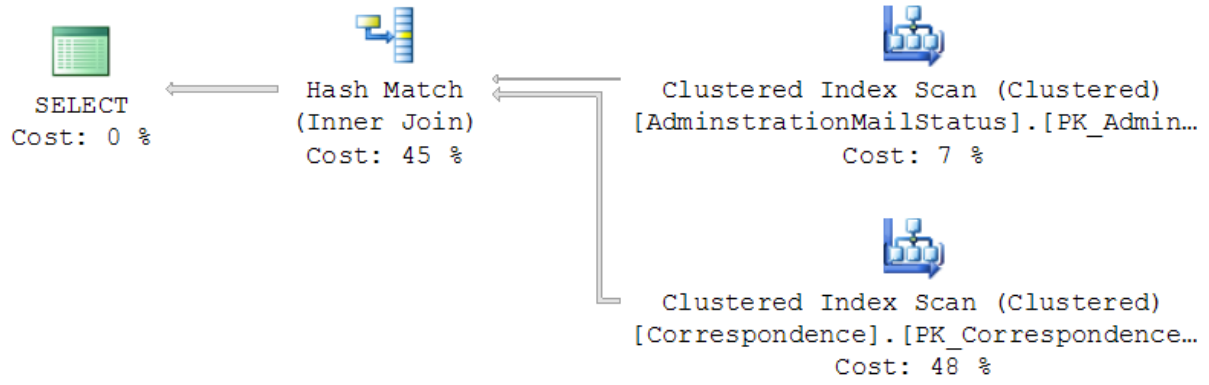
```
SELECT C.Subject,C.Message,AMS.Status FROM
DB.AdminstrationMailStatus AS  AMS join DB.Correspondence AS C
ON (AMS.ID = C.Status_ID) WHERE (Year(C.WriteDate) = '2007' );
```

Physical Logical Reads

```
Table 'Worktable'. Scan count 0, logical reads 0, physical reads 0,
read-ahead reads 0, lob logical reads 0, lob physical reads 0, lob
read-ahead reads 0.
Table 'Correspondence'. Scan count 1, logical reads 24, physical
reads 5, read-ahead reads 22, lob logical reads 0, lob physical reads
0, lob read-ahead reads 0.
Table 'AdminstrationMailStatus'. Scan count 1, logical reads 2,
physical reads 1, read-ahead reads 0, lob logical reads 0, lob
physical reads 0, lob read-ahead reads 0.

SQL Server Execution Times:
  CPU time = 0 ms,  elapsed time = 92 ms.
```

Exuction Plan



التوضيح:

استخدم Hash Match لأن عدد أسطر الجدول الأول قليل نسبياً فبُنى جدول التقطيع عليه، أما الجدول الثاني فلا يحوي اي فهرس وعدد أسطره كبير لا يوجد أي قيود عليه.

المفاتيح التي تم دمجها:

PK_AdminstrationMailStatus - PK_Correspondence

في حال حددنا حالة معينة من الرسائل فان ال Optimizer سوف يستخدم Nested Loop لنفس السبب.

3. Merge Join:

الاستعلام يجلب جميع عناوين ونصوص المراسلات الواردة في كل الأقسام والإدارات ورقم الادراة أو القسم التي أرسلت إليها:

```
SELECT  C.Subject AS 'Title', C.Message AS
'Message',IM.Reciver_Adminstration_ID AS 'Aminstration ID'
FROM DB.Correspondence AS C
     join DB.InboxMail  AS IM ON (C.ID = IM.Correspondence_ID)
     join DB.AdminstrationMailStatus as AMS ON (IM.Status_ID =
AMS.ID);
```

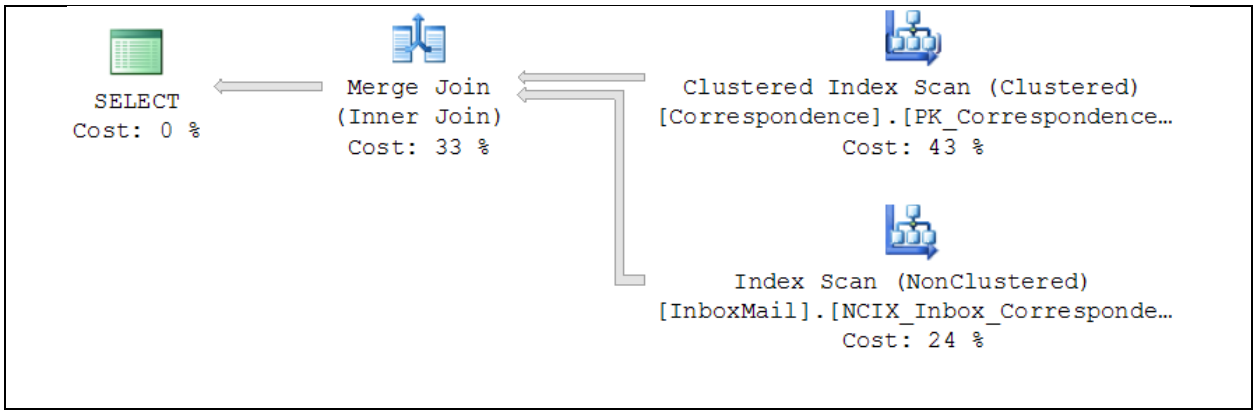
بعد تنفيذ الاستعلام السابق ينتج ما يلي:

Physical Logical Reads

```
Table 'InboxMail'. Scan count 1, logical reads 10, physical reads 2, read-
ahead reads 8, lob logical reads 0, lob physical reads 0, lob read-ahead
reads 0.
Table 'Correspondence'. Scan count 1, logical reads 24, physical reads 2,
read-ahead reads 22, lob logical reads 0, lob physical reads 0, lob read-
ahead reads 0.

 SQL Server Execution Times:
   CPU time = 16 ms,  elapsed time = 161 ms.
```

Exuction Plan



التوضيح:

عملية الدمج بين جدول المراسلة وجدول الوارد تمت وفق Merge لأن كلا الجدولين مرتبين مسبقاً احدهما وفق فهرس المفتاح الرئيسي والثاني وفق الفهرس المساعد الذي بنيناه والذي هو:

```
CREATE NONCLUSTERED INDEX NCIX_Inbox_Correspondence ON
DB.InboxMail(Correspondence_ID) include
(Status_ID,Reciver_Adminstration_ID);
```

المفاتيح التي تم دمجها هي:

PK_Correspondence - NCIX_Inbox_Correspondence

9. الطلب التاسع:

نكتب الاجرائية التالية والتي تقوم بتنفيذ الاستعلامات الأربعة السابقة 100 مرة:

```
CREATE PROCEDURE DB.TestIndex(@N INT) AS
BEGIN

DECLARE @i INT;
SET @i = 0;

WHILE(@i < @N)
BEGIN

      --First Query from Question 5+6
      SELECT a.Name AS 'Sender Name',a.Address AS 'Sender
Address',c.WriteDate AS 'Send Date',c.Message AS 'Correspondence
Message' FROM DB.Adminstration AS a join DB.Correspondence AS c
      on (a.ID = c.Sender_ID)
      WHERE (ADDRESS = 'Address of Adminstration #4' and WriteDate >
'2005-01-01') OPTION (FAST 3);

      --Second Query from 7+8 Question (Nested Loop Join)
```

```
      SELECT FirstName + ' ' + LastName, A.Name  FROM
DB.Adminstration AS A join DB.Employee AS E ON (A.ID =
E.Adminstration_ID ) WHERE
      (A.Department = 61 and Salary > 5000 and E.Gender = 'M');



      --Thired Query from 7+8 Question (Merge Join)
      SELECT  C.Subject AS 'Title', C.Message AS
'Message',IM.Reciver_Adminstration_ID AS 'Aminstration ID'
      from DB.Correspondence AS C
      join DB.InboxMail  AS IM ON (C.ID = IM.Correspondence_ID)
      join DB.AdminstrationMailStatus AS AMS ON (IM.Status_ID =
AMS.ID);


      --Forth Query from 7+8 Question (Hash Join)
      SELECT C.Subject,C.Message,AMS.Status FROM
DB.AdminstrationMailStatus AS  AMS join DB.Correspondence AS C
      on (AMS.ID = C.Status_ID) WHERE (Year(C.WriteDate) = '2007' );

      SET @i = @i + 1;
END
End
Go

Execute DB.TestIndex 1000;
```

وبعد تنفيذ الاجرائية السابقة سوف تُخزن معلومات استخدام الفهارس في الجدول SYS.DM_DB_INDEX_USAGE_STATS لعرضها جميعاً ننفذ الاستعلام التالي:

```
SELECT    OBJECT_NAME(S.[OBJECT_ID]) AS [OBJECT NAME],
          I.[NAME] AS [INDEX NAME],
          I.[TYPE_DESC] AS [INDEX TYPE],
          USER_SEEKS,
          USER_SCANS,
          USER_LOOKUPS,
          USER_UPDATES,
          (USER_SEEKS + USER_SCANS + USER_LOOKUPS) as INDEX_Reads

FROM      SYS.DM_DB_INDEX_USAGE_STATS AS S
          INNER JOIN SYS.INDEXES AS I
            ON I.[OBJECT_ID] = S.[OBJECT_ID]
               AND I.INDEX_ID = S.INDEX_ID
WHERE     OBJECTPROPERTY(S.[OBJECT_ID],'IsUserTable') = 1
```

- **user_seeks** – the number of times the index has been used in a seek operation (to find a specific row).
- **user_scans** – number of times the index has been used by scanning the leaf pages of the index for data.

- **`user_lookups`** – for clustered indexes only, this is the number of times the index has been used in a "bookmark lookup" to fetch the full row; this is because non-clustered indexes use the clustered indexes key as the pointer to the base row.
- `user_seeks + user_scans + user_lookups = user_reads.`

وعند تنفيذ الاستعلام السابق ينتج مايلي:

	OBJECT NAME	INDEX NAME	INDEX TYPE	USER_SEEKS	USER_SCANS	USER_LOOKUPS	USER_UPDATES	INDEX_Reads
1	EmployeeMail	PK_EmployeeMail	CLUSTERED	0	0	0	6000	0
2	InboxMail	PK_InboxMail	CLUSTERED	6000	1	0	3000	6001
3	InboxMail	NCIX_Inbox_Correspondence	NONCLUSTERED	0	1001	0	0	1001
4	AdminstrationMailStatus	PK_AdminstrationMailStatus	CLUSTERED	6002	0	0	5	6002
5	Adminstration	PK_Adminstration	CLUSTERED	9000	1003	0	1100	10003
6	Adminstration	NCIX_Address_Adminstration	NONCLUSTERED	1000	0	0	0	1000
7	EmployeeMailStatus	PK_EmployeeMailStatus	CLUSTERED	6000	0	0	2	6000
8	Correspondence	PK_Correspondence	CLUSTERED	3000	2006	0	2000	5006
9	Correspondence	NCIX_Date_Corrspondence	NONCLUSTERED	1000	0	0	0	1000
10	Employee	NCIX_Employee	NONCLUSTERED	1000	0	0	0	1000
11	Employee	PK_Employee	CLUSTERED	6000	1	0	4000	6001

هنا تظهر جميع الفهارس التي تم انشاءها على الجداول وتم استخدامها سواء ك Seek أو Scan أو تم التعديل على شجرة الفهرس وهذه المعطيات فقط لعمليات إدخال المعطيات والاجرائية السابقة (لا يوجد عمليات تعديل أو حذف).

نلاحظ عدم ورود فهرس الجدول Attachment ضمن الفهارس السابقة لانه الفهرس الوحيد الذي لم يتم استخادمه مطلقاً.

الفهرس الأول PK_EmployeeMail لم يتم القراءة منه ابداً ولا مرة واحدة لذلك حقل القراءة = 0 وسبب ذلك هو أن الاستعلامات التي اجريناها لم تتعامل مع هذا الجدول بينما نلاحظ تعديل شجرة الفهرس 6000 مرة وذلك بعدد أسطر هذا الجدول ومن ناحية عملية فاذا وجود هذه الحالة فان هذا الفهرس لا فائدة منه ويجب حذفه.

أما الفهرس التاسع NCIX_Date_Corrspondence فتم استخدامه بعملية Seek ألف مرة ولا يوجد تعديل على شجرة الفهرس وبالتالي هو فهرس جيد.

10. الطلب العاشر:

سوف نختار جدول الموظف ونكتب اجرائية تقوم ب 4000 عملية حذف و 4000 عملية إضافة و 4000 عملية تعديل.

الإجرائية هي:

```
create procedure DB.TestFragmentation(@department_count int,@N
int) as
Begin
declare @i int;
set @i = 0;
while(@i < @N)
Begin
     --Delete Row
     delete from DB.Employee where (ID = @i);

     --Insert Row
     insert into DB.Employee (
               FirstName,
               LastName,
               Birthday,
               Salary,
               Adminstration_ID,
               Gender,
               Phonenumber)
     values (
               DB.random_string(7),
               DB.random_string(8),
               GETDATE(),
               ABS(CHECKSUM(NewId())) % 10000 ,
               (ABS(CHECKSUM(NewId()))%@department_count + 1),
               'M',
               ABS(CHECKSUM(NewId()))
     );

     --Update Row
     update DB.Employee
     set
          Gender         = 'F',
          Salary         = ABS(CHECKSUM(NewId())) % 10000,
          Phonenumber    = ABS(CHECKSUM(NewId()))
     where (ID = @N+@i);

     set @i = @i + 1;
End
```

```
End
Go
```

مع العلم أن الجدول يوجد عليه 3 فهارس استخدمناها مسبقاً في الطلبات السابقة وهي:

- فهرس معنقد على المفتاح الرئيسي.
- فهرس غير معنقد على الاسم الأول والأخير والراتب.
- فهرس غير معنقد على رقم الإدارة التي يتبع لها الموظف يتضمن الاسم الأول والأخير والراتب والجنس.

لتحديد مستوى التجزئة الحاصلة في الجدول يوجد طريقتين:

- باستخدام التعليمة DBCC SHOWCONFIC.
- باستخدام الجدول dm_db_index_physical_stats.

الطريقة الأولى قديمة وأصبحت غير مستخدمة ولا تعطي تفاصيل كل فهرس سوف نستخدم الطريقة الثانية والاستعلام هو:

```
SELECT
      SI.name                          AS 'Index Name',
      SI.type_desc                     AS 'Index Type',
      PS.index_level                   AS 'Index Level (leaf 0)',
      PS.avg_fragment_size_in_pages AS 'AVG Fragmentation %',
      PS.fragment_count                AS 'Fragment Count',
      PS.page_count                    AS 'Total Pages',
      avg_fragment_size_in_pages       AS 'Pages/Frag'
FROM
      sys.dm_db_index_physical_stats(
                db_id(db_name()),
                OBJECT_ID(N'Correspondences_DB.DB.Employee'),
                NULL,NULL , 'DETAILED') AS PS
      LEFT JOIN
      sys.indexes  AS SI ON(PS.index_id = SI.index_id)
WHERE (SI.object_id = PS.object_id);
```

وقبل تنفيذ الإجرائية يكون خرج الاستعلام السابق كما يلي:

	Index Name	Index Type	Index Level (leaf 0)	AVG Fragmentation %	Fragment Count	Total Pages	Pages/Frag
1	PK_Employee	CLUSTERED	0	23.0769230769231	8	26	3.25
2	PK_Employee	CLUSTERED	1	0	1	1	1
3	NCIX_Full_Name_Salary	NONCLUSTERED	0	83.3333333333333	18	18	1
4	NCIX_Full_Name_Salary	NONCLUSTERED	1	0	1	1	1
5	NCIX_Adminstration_ID	NONCLUSTERED	0	88	25	25	1
6	NCIX_Adminstration_ID	NONCLUSTERED	1	0	1	1	1

الاستعلام جلب الأعمدة التي تهمنا من الجدول، كما نلاحظ أن لكل فهرس سطرين فقط أي أن شجرة الفهرس لها مستويين الأول في مستوى الأوراق.

أن فهرس المفتاح الرئيسي المعنقد يوجد فيه تجزئة بمقدار 3.25% وتعتبر نسبة مقبولة (نسبة الصفحات الغير مرتبة) يتوزع هذا الفهرس على 8 أجزاء في وعدد الصفحات الكلي هو 26.

وبعد تنفيذ الإجرائية يكون خرج الإستعلام:

	Index Name	Index Type	Index Level (leaf 0)	AVG Fragmentation %	Fragment Count	Total Pages	Pages/Frag
1	PK_Employee	CLUSTERED	0	38.0952380952381	32	63	1.96875
2	PK_Employee	CLUSTERED	1	0	1	1	1
3	NCIX_Full_Name_Salary	NONCLUSTERED	0	89.7435897435898	39	39	1
4	NCIX_Full_Name_Salary	NONCLUSTERED	1	0	1	1	1
5	NCIX_Adminstration_ID	NONCLUSTERED	0	94.2307692307692	52	52	1
6	NCIX_Adminstration_ID	NONCLUSTERED	1	0	1	1	1

جميع الفهارس قد تضررت من عمليات الاضافة والحذف والتعديل ولاصلاحها أيضاً يوجد خيارين ولكل خيار عدة طرق للتطبيق:

- اعادة بناء الفهرس Rebuild Index عندما تكون نسبة التجزئة كبيرة.
- اعادة ترتيب الفهرس Reorganizing Index في حال كانت نسبة التجزئة صغيرة.

نسبة التجزئة في الفهارس كبيرة جداً لذلك سوف نختار اعادة بناء الفهرس بالطريقة التالية:

```
ALTER INDEX ALL ON DB.Employee REBUILD
```

وعند تنفيذ الاستعلام السابق نجد:

	Index Name	Index Type	Index Level (leaf 0)	AVG Fragmentation %	Fragment Count	Total Pages	Pages/Frag
1	PK_Employee	CLUSTERED	0	0	7	52	7.42857142857143
2	PK_Employee	CLUSTERED	1	0	1	1	1
3	NCIX_Full_Name_Salary	NONCLUSTERED	0	0	3	30	10
4	NCIX_Full_Name_Salary	NONCLUSTERED	1	0	1	1	1
5	NCIX_Adminstration_ID	NONCLUSTERED	0	2.85714285714286	5	35	7
6	NCIX_Adminstration_ID	NONCLUSTERED	1	0	1	1	1

حيث تم إعادة البنية الفيزيائية للفهارس وانخفضت نسبة التجزئة.

11. الطلب الحادي عشر:

نحصل على حجموم ملفات الداتابيز من خلال الإستعلام التالي:

```
SELECT
      name                                    AS 'Logical Name',
      physical_name                           AS 'Physical Name',
      size                                    AS 'Total Pages',
      FILEPROPERTY(name, 'SpaceUsed')         AS 'Used Pages',
      size/128                                AS 'Total Space MB',
      FILEPROPERTY(name, 'SpaceUsed')/ 128     AS 'Used Space MB',
      size/128-FILEPROPERTY(name,'SpaceUsed')/128 AS 'Free Space'
FROM sys.database_files;
```

ننفذ هذا الاستعلام عندما تكون الداتابيز خالية فتكون النتيجة كما يلي:

	Logical Name	Physical Name	Total Pages	Used Pages	Total Space MB	Used Space MB	Free Space MB
1	Correspondences System DB	D:\Database\Project\DatabaseFiles\CSDB.mdf	512	192	4	1	3
2	HW_log1	D:\Database\Project\DatabaseFiles\HW_log1.ldf	704	108	5	0	5
3	HW_file1	D:\Database\Project\DatabaseFiles\HW_file1.ndf	256	32	2	0	2
4	HW_file2	D:\Database\Project\DatabaseFiles\HW_file2.ndf	256	32	2	0	2
5	HW_file3	D:\Database\Project\DatabaseFiles\HW_file3.ndf	256	8	2	0	2
6	HW_file4	D:\Database\Project\DatabaseFiles\HW_file4.ndf	384	8	3	0	3
7	BigFiles	D:\Database\Project\DatabaseFiles\BigFiles.ndf	65536	8	512	0	512
8	HW_log2	D:\Database\Project\DatabaseFiles\HW_log2.ldf	704	108	5	0	5

وبعد إضافة عدد كبير من البيانات على الجداول تصبح:

	Logical Name	Physical Name	Total Pages	Used Pages	Total Space MB	Used Space MB	Free Space MB
1	Correspondences System DB	D:\Database\Project\DatabaseFiles\CSDB.mdf	512	192	4	1	3
2	HW_log1	D:\Database\Project\DatabaseFiles\HW_log1.ldf	704	220	5	1	4
3	HW_file1	D:\Database\Project\DatabaseFiles\HW_file1.ndf	256	32	2	0	2
4	HW_file2	D:\Database\Project\DatabaseFiles\HW_file2.ndf	256	32	2	0	2
5	HW_file3	D:\Database\Project\DatabaseFiles\HW_file3.ndf	512	472	4	3	1
6	HW_file4	D:\Database\Project\DatabaseFiles\HW_file4.ndf	384	384	3	3	0
7	BigFiles	D:\Database\Project\DatabaseFiles\BigFiles.ndf	65536	8	512	0	512
8	HW_log2	D:\Database\Project\DatabaseFiles\HW_log2.ldf	704	220	5	1	4

نلاحظ أن حجم الملف HW_file3 تتضاعف وأصبح 4 ميغا، لأن جميع الجداول قد تم بناءها على مجموعة الملفات FileGroup والملف HW_file3 ينتمي إليها.

وبعد حذف جميع الداتا المضافة تصبح:

	Logical Name	Physical Name	Total Pages	Used Pages	Total Space MB	Used Space MB	Free Space MB
1	Correspondences System DB	D:\Database\Project\DatabaseFiles\CSDB.mdf	512	192	4	1	3
2	HW_log1	D:\Database\Project\DatabaseFiles\HW_log1.ldf	704	226	5	1	4
3	HW_file1	D:\Database\Project\DatabaseFiles\HW_file1.ndf	256	32	2	0	2
4	HW_file2	D:\Database\Project\DatabaseFiles\HW_file2.ndf	256	32	2	0	2
5	HW_file3	D:\Database\Project\DatabaseFiles\HW_file3.ndf	512	8	4	0	4
6	HW_file4	D:\Database\Project\DatabaseFiles\HW_file4.ndf	384	8	3	0	3
7	BigFiles	D:\Database\Project\DatabaseFiles\BigFiles.ndf	65536	8	512	0	512
8	HW_log2	D:\Database\Project\DatabaseFiles\HW_log2.ldf	704	226	5	1	4

بقي حجم الملف كما هو ولكن عدد الصفحات المستخدمة أصبح 8 أي أن الملف فارغ.

نقوم بتنفيذ تعليمة Shrink على الملف:

```
DBCC SHRINKFILE (HW_file3)
```

فتكون النتيجة هي عودة الملف إلى حجمه الابتدائي...

	Logical Name	Physical Name	Total Pages	Used Pages	Total Space MB	Used Space MB	Free Space MB
1	Correspondences System DB	D:\Database\Project\DatabaseFiles\CSDB.mdf	512	192	4	1	3
2	HW_log1	D:\Database\Project\DatabaseFiles\HW_log1.ldf	704	103	5	0	5
3	HW_file1	D:\Database\Project\DatabaseFiles\HW_file1.ndf	256	32	2	0	2
4	HW_file2	D:\Database\Project\DatabaseFiles\HW_file2.ndf	256	32	2	0	2
5	HW_file3	D:\Database\Project\DatabaseFiles\HW_file3.ndf	256	8	2	0	2
6	HW_file4	D:\Database\Project\DatabaseFiles\HW_file4.ndf	384	8	3	0	3
7	BigFiles	D:\Database\Project\DatabaseFiles\BigFiles.ndf	65536	8	512	0	512
8	HW_log2	D:\Database\Project\DatabaseFiles\HW_log2.ldf	704	103	5	0	5

12. الطلب الثاني عشر:

الاستعلام هو:

جلب عنوان المراسلات الشخصية والمذكرات لكل موظف مع حالة كل مراسلة واردة بالنسبة للموظف:

```
select FirstName + + LastName as 'Full Name',Status as 'Inbox
Status',Subject as 'Inobx Subject' from DB.Employee as E
join DB.EmployeeMail      as EM  on (E.ID = EM.Reciver_Employee_ID)
join DB.InboxMail           as IM  on (EM.Inbox_ID = IM.ID)
join DB.Correspondence      as C   on(C.ID = IM.Correspondence_ID)
join DB.EmployeeMailStatus as EMS on (EMS.ID = EM.Status_ID);
```

الاستعلام يحتاج الى أربع عمليات دمج سوف نختصرها بعملية إختيار واحدة من جدول الوارد للموظف مع التوابع التالية:

تابع نعطيه رقم وارد للإدارة يرد لنا رقم المراسلة:

```
Create Function DB.Get_Correspondense_ID(@Inbox_Mail_ID int)
returns int as
Begin
     return (select Correspondence_ID from DB.InboxMail where
     (InboxMail.ID = @Inbox_Mail_ID) )
End
```

تابع نعطيه رقم الموظف يرد لنا اسم الكامل لهذا الموظف:

```
Create Function DB.Get_Employee_Name(@Employee_ID int) returns
varchar(30) as
```

```
Begin
     return (select FirstName +' '+ LastName from DB.Employee
     where (Employee.ID = @Employee_ID));
End
Go
```

تابع نعطيه رقم حالة المراسلة فيرد لنا الحالة:

```
Create Function DB.Get_Status(@Status_ID int) returns
varchar(30) as
Begin
     return (select AdminstrationMailStatus.Status from
     DB.AdminstrationMailStatus where
     (AdminstrationMailStatus.ID = @Status_ID));
End
```

تابع نعطيه رقم المراسلة فيرد لنا عنوانها:

```
Create Function DB.GetCorrespondenceSubject(@Corrsepondence_ID
int) returns varchar(255) as
Begin
     return (select Correspondence.Subject from
     DB.Correspondence where (Correspondence.ID =
     @Corrsepondence_ID));
End
Go
```

فيصبح الاستعلام باستخدام التوابع على الشكل التالي:

```
select
DB.Get_Employee_Name(EmployeeMail.Reciver_Employee_ID) as 'Full
Name',
DB.Get_Status(EmployeeMail.Status_ID) as 'Status Name',
DB.GetCorrespondenceSubject(DB.Get_Correspondense_ID(EmployeeMai
l.Inbox_ID))   as 'Corresponcence Subject'
from DB.EmployeeMail;
```

بمقارنة زمن التنفيذ وعمليات الكتابة والقراءة لكلا الاستعالمين نلاحظ أن عمليات الدمج الداخلية التي يقوم بها Optimizer أسرع بكثير من التوابع التي عرفناها.

13. الطلب الثالث عشر:

```
CREATE TRIGGER PrintTrigger ON  DB.Attachment AFTER
INSERT,UPDATE,DELETE AS
BEGIN
SET NOCOUNT ON;
declare @affected_rows_count int;
IF (EXISTS(SELECT * FROM INSERTED) AND EXISTS(SELECT * FROM
DELETED))
```

```
      Begin
            select @affected_rows_count = count(*) from inserted;
            print cast(@affected_rows_count as varchar) +
                  ' has been updated in Employee table'
      End
Else
IF EXISTS(SELECT * FROM INSERTED)
      Begin
            select @affected_rows_count = count(*) from inserted;
            print cast(@affected_rows_count as varchar) +
                  ' has been inserted into Employee table'
      End



Else
IF  EXISTS(SELECT * FROM DELETED)
      Begin
            select @affected_rows_count = count(*) from deleted;
            print cast(@affected_rows_count as varchar) +
                  ' has been deleted from Employee table'
      End
END
GO
```

التوضيح:

ننشأ Trigger على العمليات الثلاثة (Insert, Delete, Update) ولتميز الحالات الثالثة سوف نستخدم الجدولين Inserted, Delete ومن خلال التابع EXISTS يمكننا معرفة اذا كان أحد الجدولين يحوي أسطر.

فعندما يكون كلا الجدولين يحويان أسطر فالقادح تم اطلاقه بسبب عملية تعديل على الجدول، أما اذا كان فقط جدول Inserted يحوي أسطر فهذا يعني أن العملية هي عملية اضافة، وفي حال كان فقط جدول Deleted يحوي أسطر فهذا يعني أن العملية هي عملية حذف.

لجلب عدد الأسطر المعدلة أو المضافة أو المحذوفة سوف نستخدم تابع العد Count على الجدولين Deleted, Inserted.

14. الطلب الرابع عشر:

سوف نستخدم قادح من نوع instead of فهو يتيح تعديل القيم قبل القيام بعملية الاضافة أو الحذف أو التعديل.

لتحقيق المطلوب سوف يكون لدينا قادحين واحد للإضافة وواحد للحذف.

قادح الاضافة:

```
Create Trigger TR_EmpolyeeSalary_IN ON DB.Employee INSTEAD OF
INSERT AS
Begin
     insert into DB.Employee
          (FirstName,
          LastName,
          Birthday,
          Salary,
          Gender,
          PhoneNumber,
          Adminstration_ID)

          (select
               FirstName,
               LastName,
               Birthday,
               case
                    when (Salary > 0) then Salary * -1
                    when (Salary < 0) then Salary
               end,
               Gender,
               PhoneNumber,
               Adminstration_ID
           from inserted );
End
```

في حالتنا يوجد نوعين من الاضافة واحدة مع قيمة سالبة وواحدة بقيمة موجبة، في الاولى تضاف من دون تعديل أي وفي الثانية يجب عكس الإشارة.

يجب الانتباه إلى أن عملية الاضافة الواحدة يمكن أن تضيف أكثر من سطر وبالتالي القادح لا يعمل إلا مرة واحدة.

يمكن تحقيق ما سبق بأجراء استعلام على جدول Inserted الذي يحوي جميع الأسطر المضافة الاستعلام هو:

```
(select
     FirstName,
     LastName,
     Birthday,
     case
          when (Salary > 0) then Salary * -1
          when (Salary < 0) then Salary
     end,
     Gender,
     PhoneNumber,
     Adminstration_ID
from inserted );
```

كما نلاحظ ان هذا الاستعلام يجلب لنا جميع الاسطر المضافة وخرج هذا الاستعلام هو دخل لعملية إضافة ولكن تبديل القيم سوف يتم بالاستعلام السابق حيث استخدمنا عبارة Case على حقل الراتب ففي حال كان الراتب موجب ترد لنا قيمة سالبة وفي حال كان سالب ترده نفسه عندها نحصل على الأسطر المضافة مع تعديل القيمة الموجبة وجعلها سالبة لنضيفها إلى الجدول بسلام.

يمكن اجراء العملية بطريقة ثانية وهي تجزئة الاستعلام السابق لاستعلامين الأول يجلب الاسطر المضافة مع قيمة موجبة والثاني يجلب الأسطر المضافة مع قيمة سالبة.

قادح التعديل:

```
Create Trigger TR_EmpoyeeSalary_UP on DB.Employee INSTEAD OF
UPDATE AS
Begin

update E set
E.Salary  =  case
                  when I.Salary > 0 then I.Salary * -1
                  when I.Salary < 0 then I.Salary
                  end,
E.FirstName    = I.FirstName,
E.LastName     = I.LastName,
E.BirthDay     = I.BirthDay,
E.Gender  = I.Gender,
E.PhoneNumber = I.PhoneNumber,
E.Adminstration_ID = I.Adminstration_ID
from DB.Employee as E inner join inserted as I
     on (E.ID = I.ID)
     where (UPDATE(Salary))
```

```
update E set
E.Salary  = I.Salary,
E.FirstName    = I.FirstName,
E.LastName     = I.LastName,
E.BirthDay     = I.BirthDay,
E.Gender  = I.Gender,
E.PhoneNumber = I.PhoneNumber,
E.Adminstration_ID = I.Adminstration_ID
from DB.Employee as E inner join inserted as I
     on (E.ID = I.ID)
     where (not UPDATE(Salary))
End
```

ايضاً نفس الفكرة السابقة يوجد ثلاث أنواع من التعديل: تعديل مع قيم سالبة وتعديل مع قيم موجبة وتعديل بدون تغير القيم الموجبة أو السالبة وفي جميع الحالات يمكن أن يتم على أكثر من سطر بنفس الوقت.

لتعديل الجدول من خلال استعلام على جدول Inserted يمكننا دمج الجدولين حسب المفتاح الأولي ومن ثم اختيار القيم التي نريدها من الجدول.

لتحقيق ما سبق سوف نكتب استعلامين:
الاستعلام الأول:

```
update E set
E.Salary  =  case
                  when I.Salary > 0 then I.Salary * -1
                  when I.Salary < 0 then I.Salary
                  end,
E.FirstName    = I.FirstName,
E.LastName     = I.LastName,
E.BirthDay     = I.BirthDay,
E.Gender  = I.Gender,
E.PhoneNumber = I.PhoneNumber,
E.Adminstration_ID = I.Adminstration_ID
from DB.Employee as E inner join inserted as I
     on (E.ID = I.ID)
     where (UPDATE(Salary))
```

يجلب الأسطر من جدول Inserted والتي عدُلت قيمة الراتب فيها سواء بقيم موجبة أو سالبة استفدنا من تابع Update المنطقي الذي يدل على تغير قيمة عامود، وحسب القيمة الجديدة عدلنا.
والاستعلام الثاني كما هو موضح في جسم القادح جلب الأسطر التي لم يتعدل فيها قيمة الراتب وعدلنا على أساسها قيم الجدول الاصلي.