

Optimisation des requêtes et schéma d'accès aux données

Rappel SQL

Table

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
5	Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Rappel SQL

- Instruction exemple :

```
SELECT * FROM Customers;
```

- SQL n'est pas sensible à la casse
- On sépare les instructions par des points virgule

Des commandes importants

- **SELECT**
- **UPDATE**
- **DELETE**
- **INSERT INTO**
- **CREATE DATABASE**
- **ALTER DATABASE**
- **CREATE TABLE**
- **ALTER TABLE**
- **DROP TABLE**
- **CREATE INDEX**
- **DROP INDEX**

SELECT

SELECT * FROM *table_name*;

Ou

**SELECT column_name,column_name
FROM table_name;**

DISTINCT

```
SELECT DISTINCT column_name,column_name  
FROM table_name;
```

Exemple :

```
SELECT DISTINCT City FROM Customers;
```

WHERE

```
SELECT column_name,column_name  
FROM table_name  
WHERE column_name operator value;
```

Exemple :

```
SELECT * FROM Customers  
WHERE Country='Mexico';
```


AND & OR

```
SELECT * FROM Customers  
WHERE Country='Germany'  
AND City='Berlin';
```

```
SELECT * FROM Customers  
WHERE City='Berlin'  
OR City='München';
```

```
SELECT * FROM Customers  
WHERE Country='Germany'  
AND (City='Berlin' OR City='München');
```

ORDER BY

```
SELECT column_name, column_name  
FROM table_name  
ORDER BY column_name ASC|DESC, column_name ASC|DESC;
```

```
SELECT * FROM Customers  
ORDER BY Country;
```

```
SELECT * FROM Customers  
ORDER BY Country DESC;
```

ORDER BY

```
SELECT * FROM Customers  
ORDER BY Country, CustomerName;
```

```
SELECT * FROM Customers  
ORDER BY Country ASC, CustomerName DESC;
```

INSERT INTO

```
INSERT INTO table_name  
VALUES (value1,value2,value3,...);
```

```
INSERT INTO table_name (column1,column2,column3,...)  
VALUES (value1,value2,value3,...);
```

```
INSERT INTO Customers (CustomerName, City, Country)  
VALUES ('Cardinal', 'Stavanger', 'Norway');
```

UPDATE

*UPDATE table_name
SET column1=value1,column2=value2,...
WHERE some_column=some_value;*

UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg'
WHERE CustomerName='Alfreds Futterkiste';

Attention au where

UPDATE Customers
SET ContactName='Alfred Schmidt', City='Hamburg';

DELETE

```
DELETE FROM table_name  
WHERE some_column=some_value;
```

```
DELETE FROM Customers  
WHERE CustomerName='Alfreds Futterkiste' AND ContactName='Maria  
Anders';
```

Si on met pas where on supprime toutes les données

SQL Injection

L'injection SQL peut détruire votre base de données.

```
txtUserId = getRequestString("UserId");  
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

SELECT * FROM Users WHERE UserId = 105 or 1=1

SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers

Pour se protéger : Requêtes paramétrées

Exemple en PHP

```
$stmt = $dbh->prepare("INSERT INTO Customers  
(CustomerName,Address,City)  
VALUES (:nam, :add, :cit)");  
$stmt->bindParam(':nam', $txtNam);  
$stmt->bindParam(':add', $txtAdd);  
$stmt->bindParam(':cit', $txtCit);  
$stmt->execute();
```


SQL SELECT TOP

```
SELECT TOP number|percent column_name(s)  
FROM table_name;
```

```
SELECT column_name(s)  
FROM table_name  
LIMIT number;
```

LIKE

```
SELECT column_name(s)  
FROM table_name  
WHERE column_name LIKE pattern;
```

```
SELECT * FROM Customers  
WHERE City LIKE 's%';// Tous les clients dont le cité commencent par s
```

```
SELECT * FROM Customers  
WHERE City LIKE '%s';// Tous les clients dont le cité se terminent par s
```

```
SELECT * FROM Customers  
WHERE Country LIKE '%land%';// Tous les clients dont le pays contient land
```

Wildcard

`%`: remplace zéro ou plusieurs caractères

`_`: remplace une seule caractère

`[charlist]`: ensemble ou intervalles de caractères à trouver

`[^charlist]`: ensemble ou intervalles de caractères à trouver
or

`[!charlist]`

Exemple wildcards

```
SELECT * FROM Customers  
WHERE City LIKE 'ber%';  
// retourne les clients don't la cité commence par ber
```

```
SELECT * FROM Customers  
WHERE City LIKE '%es%';  
// retourne les clients don't la cité contient es
```

```
SELECT * FROM Customers  
WHERE City LIKE '_erlin';  
// retourne les clients don't la cite commence n'importe quelle caractère suivi par erlin
```

SQL functions

SQL Aggregate Functions

SQL Scalar functions

SQL Aggregate Functions

SQL aggregate functions return a **single value, calculated from values in a column.**

Useful aggregate functions:

- AVG() - Returns the average value
- COUNT() - Returns the number of rows
- FIRST() - Returns the first value
- LAST() - Returns the last value
- MAX() - Returns the largest value
- MIN() - Returns the smallest value
- SUM() - Returns the sum

SQL Scalar functions

- UCASE() - Converts a field to upper case
- LCASE() - Converts a field to lower case
- MID() - Extract characters from a text field
- LEN() - Returns the length of a text field
- ROUND() - Rounds a numeric field to the number of decimals specified
- NOW() - Returns the current system date and time
- FORMAT() - Formats how a field is to be displayed

SQL Function : AVG()

Syntax :

SELECT AVG(column_name) FROM table_name

Example :

select AVG(ActualCost) from Production.TransactionHistory

SQL Function : COUNT ()

Syntax :

```
SELECT COUNT(column_name) FROM table_name
```

To display distinct values :

```
SELECT COUNT(DISTINCT column_name) FROM table_name;
```

Example :

```
select COUNT(*) from Production.TransactionHistory
```

```
select COUNT(distinct ActualCost) from Production.TransactionHistory
```

SQL Function : FIRST ()

Syntax :

```
SELECT FIRST(column_name) FROM table_name;
```

The FIRST() function is only supported in MS Access.

```
SELECT TOP 1 column_name FROM table_name  
ORDER BY column_name ASC;//SQL Server
```

```
SELECT CustomerName FROM Customers  
WHERE ROWNUM <=1  
ORDER BY CustomerID ASC;//Oracle
```

```
SELECT column_name FROM table_name  
ORDER BY column_name ASC  
LIMIT 1;//Mysql
```

SQL Function : FIRST ()

Example:

```
select top 1 * from Production.TransactionHistory  
order by TransactionID asc
```

SQL Function : LAST ()

Syntax :

SELECT FIRST(column_name) FROM table_name;

The FIRST() function is only supported in MS Access.

SELECT TOP 1 *column_name* FROM *table_name*
ORDER BY *column_name* DESC; // **SQL Server**

SELECT CustomerName FROM Customers
WHERE ROWNUM <=1
ORDER BY CustomerID DESC; // **Oracle**

SELECT *column_name* FROM *table_name*
ORDER BY *column_name* DESC
LIMIT 1; // **Mysql**

SQL Function : LAST ()

Example:

```
select top 1 * from Production.TransactionHistory  
order by TransactionID desc
```

SQL Function : MAX()

Syntax :

SELECT MAX(column_name) FROM table_name

Example :

select MAX(ActualCost) from Production.TransactionHistory

SQL Function : MIN()

Syntax :

SELECT MIN (column_name) FROM table_name

Example :

select MIN (ActualCost) from Production.TransactionHistory

SQL Function : SUM()

Syntax :

SELECT SUM (column_name) FROM table_name

Example :

select SUM (ActualCost) from Production.TransactionHistory

SQL GROUP BY Statement

Syntax :

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name;
```

Example :

```
select SUM(ActualCost), t.ModifiedDate, t.Quantity
from Production.TransactionHistory t
group by
t.ModifiedDate, t.Quantity
```

SQL HAVING Clause

Syntax :

```
SELECT column_name, aggregate_function(column_name)
FROM table_name
WHERE column_name operator value
GROUP BY column_name
HAVING aggregate_function(column_name) operator value;
```

Example :

```
select SUM(ActualCost), t.ModifiedDate, t.Quantity
from Production.TransactionHistory t
group by
t.ModifiedDate, t.Quantity
having
SUM(ActualCost) > 100
```

SQL UCASE() Function

Syntax :

SELECT UPPER (column_name) FROM table_name

Example :

select UPPER (p.FirstName) from Person.Person p

Datetime vs datetime2

```
DECLARE
```

```
@thedatetime2 datetime2(7),
```

```
@thedatetime datetime;
```

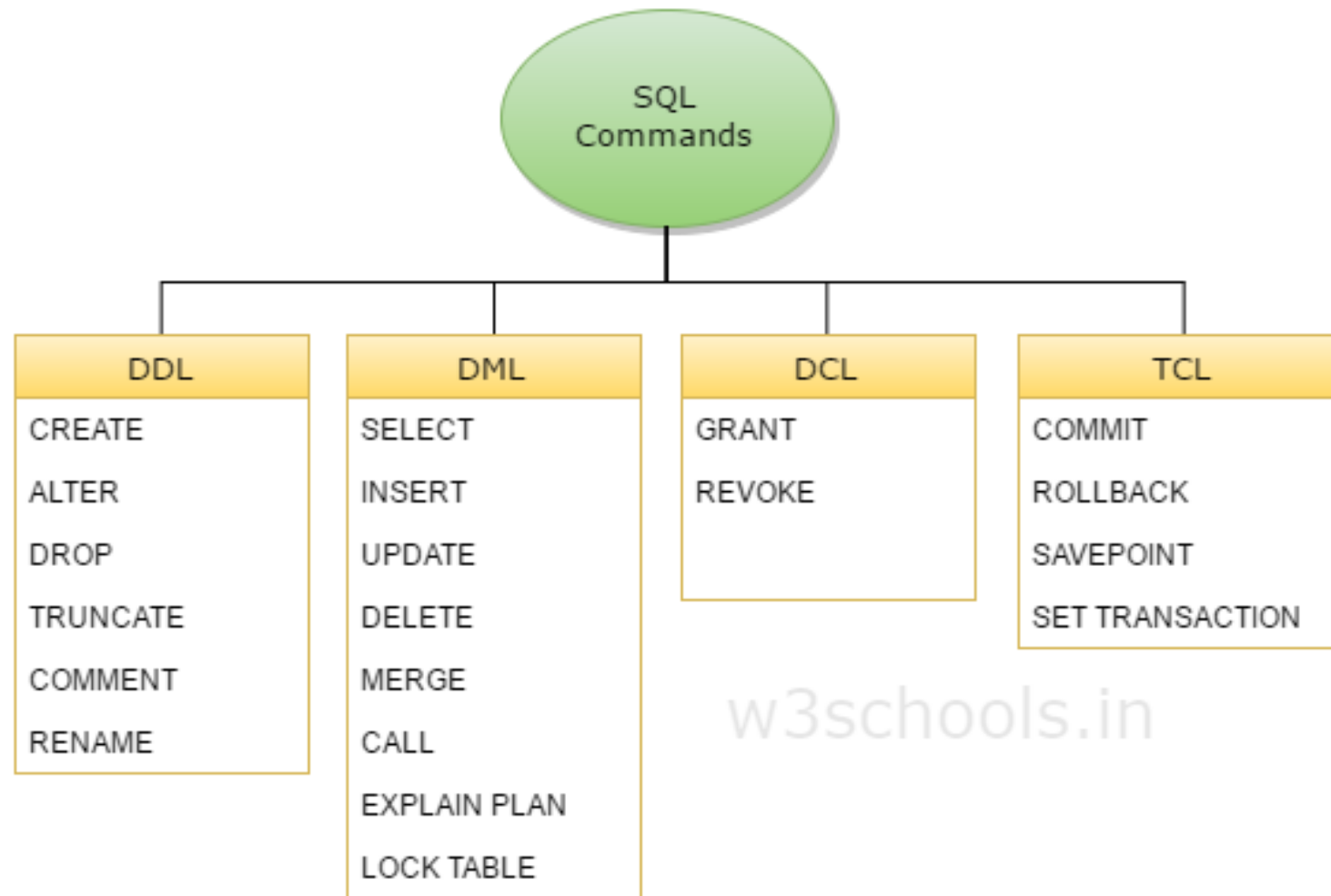
```
SET @thedatetime2 = '2025-05-21  
10:15:30.5555555';
```

```
SET @thedatetime = @thedatetime2;
```

```
SELECT
```

```
@thedatetime2 AS 'datetime2',
```

```
@thedatetime AS 'datetime';
```



w3schools.in

Catégorie
des
commandes
SQL

Procédure stocké et fonctions

Les fonctions retournent une valeur et ne peuvent pas modifier les données qu'elles reçoivent en tant que paramètres (les arguments).

Les fonctions ne sont pas autorisées à modifier quoi que ce soit, doivent avoir au moins un paramètre et doivent renvoyer une valeur. Les proc stockés n'ont pas besoin d'avoir un paramètre, peuvent changer les objets de base de données et n'ont pas besoin de retourner une valeur.

Clustered index and non clustered index



UN INDEX CLUSTERISÉ SIGNIFIE QUE VOUS INDIQUEZ À LA BASE DE DONNÉES DE STOCKER SUR LE DISQUE DES VALEURS PROCHES LES UNES DES AUTRES (SUR LE DISQUE). CELA PRÉSENTE L'AVANTAGE D'ANALYSER / DE RÉCUPÉRER RAPIDEMENT LES ENREGISTREMENTS QUI SE SITUENT DANS UNE PLAGE DE VALEURS D'INDEX EN CLUSTER.



UN SEUL INDEX CLUSTERED PAR TABLE



LE CI DOIT TOUJOURS ÊTRE UTILISÉ POUR LA PK

Clustered index and non clustered index

Avec un index non clusterisé, il existe une seconde liste qui contient des pointeurs sur les lignes physiques. Vous pouvez avoir plusieurs index non clusterisés, bien que chaque nouvel index augmente le temps nécessaire pour écrire de nouveaux enregistrements.

Il est généralement plus rapide de lire à partir d'un index clusterisé si vous souhaitez récupérer toutes les colonnes. Vous n'êtes pas obligé d'aller d'abord à l'index, puis à la table.

L'écriture dans une table avec un index clusterisé peut être plus lente s'il est nécessaire de réorganiser les données.

Physical joins



Nested Loop
join



Merge join



And Hash join

Physical joins

On va exécuter cette requête en SQL server :

```
SELECT e.[BusinessEntityID]
,p.[Title]
,p.[FirstName]
,p.[MiddleName]
,p.[LastName]
,p.[Suffix]
,e.[JobTitle]
FROM [HumanResources].[Employee] e
INNER JOIN [Person].[Person] p
ON p.[BusinessEntityID] = e.[BusinessEntityID]
```

Physical joins

C'est maintenant un travail de serveur SQL de créer un plan approprié pour la requête, de l'exécuter et de renvoyer l'ensemble des résultats à l'appelant. SQL Server dispose de plusieurs composants pour effectuer cette série de tâches, notamment l'analyse de requête, la création d'une arborescence de requête, la création d'un plan binaire, l'exécution de la requête et le retour du jeu de résultats.

L'exécution réelle de la requête est gérée par le composant du moteur de stockage de SQL Server.

Physical joins

Pour exécuter la requête ci-dessus, SQL peut sélectionner n'importe laquelle

des actions Nested Loop join, merge join et hash joins, en fonction de critères tels que :

- Taille de table
- Ordre d'index utilisé dans join
- Index disponibles
- Type d'index - Cluster ou non-cluster
- etc

Loop join

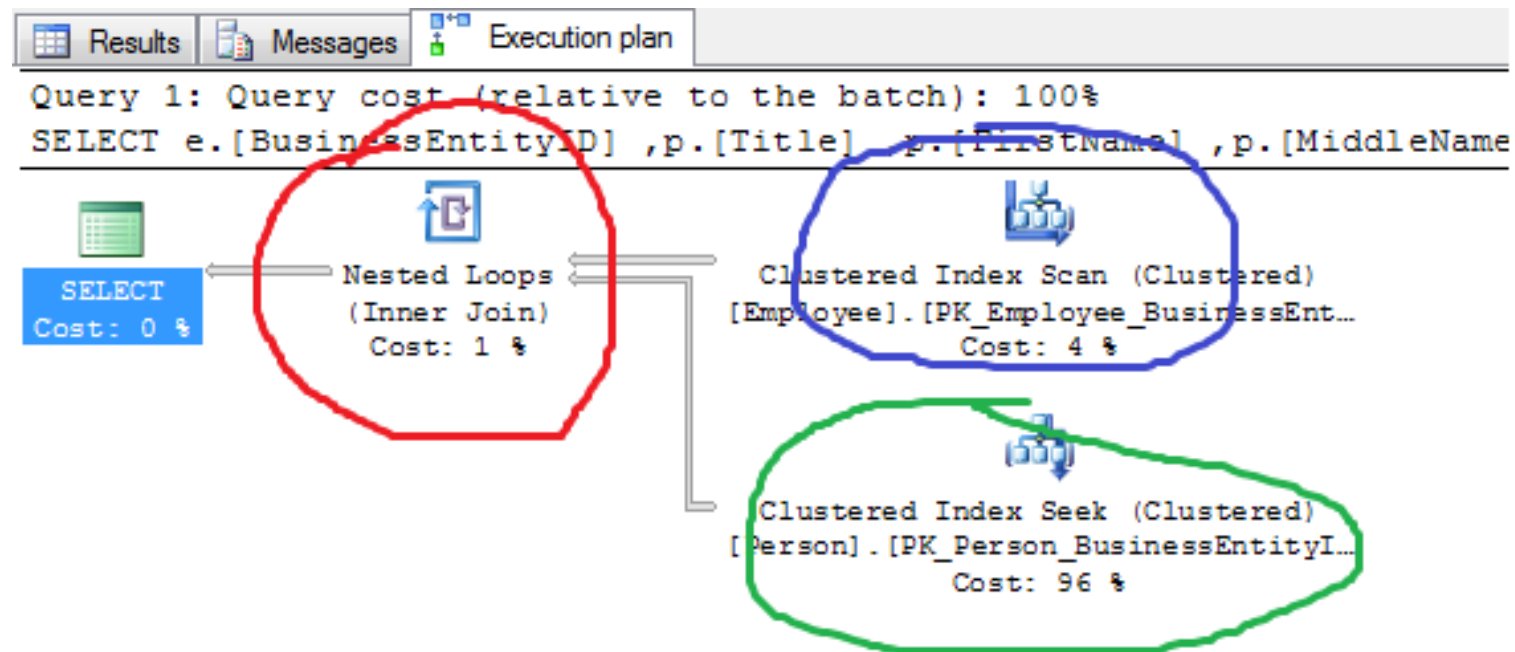
SQL Server choisit Loop join quand un jeu d'entrée est petit et que l'autre est assez volumineux et indexé sur sa colonne de jointure, nested loop join est l'opération de jointure la plus rapide car elle nécessite le moins d'Entrées / Sortie et le moins de comparaisons.

On l'appelle aussi itération imbriquée.

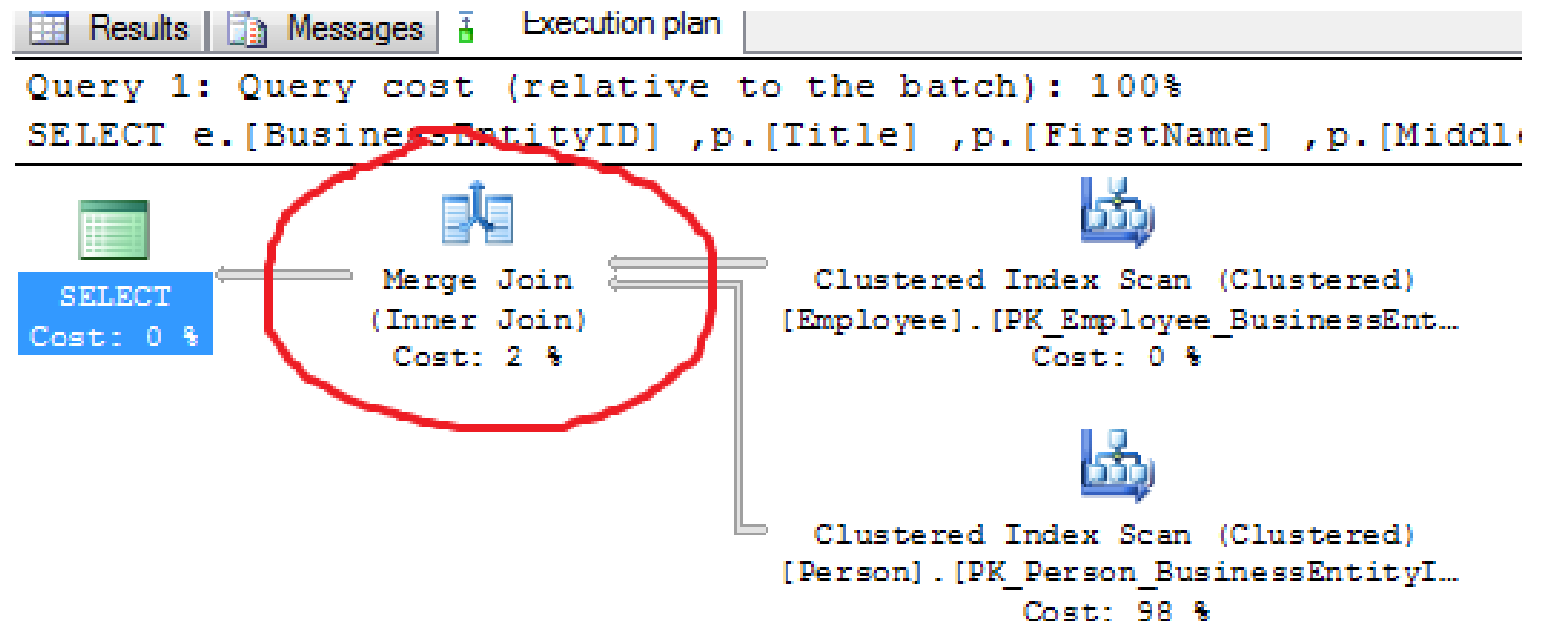
Il est particulièrement efficace si l'entrée externe est petite et si l'entrée interne est préindexée et grande.

Pour de nombreuses petites transactions avec de petits ensembles de données, le type de jointure nested loop est supérieur à merge et hash joins.

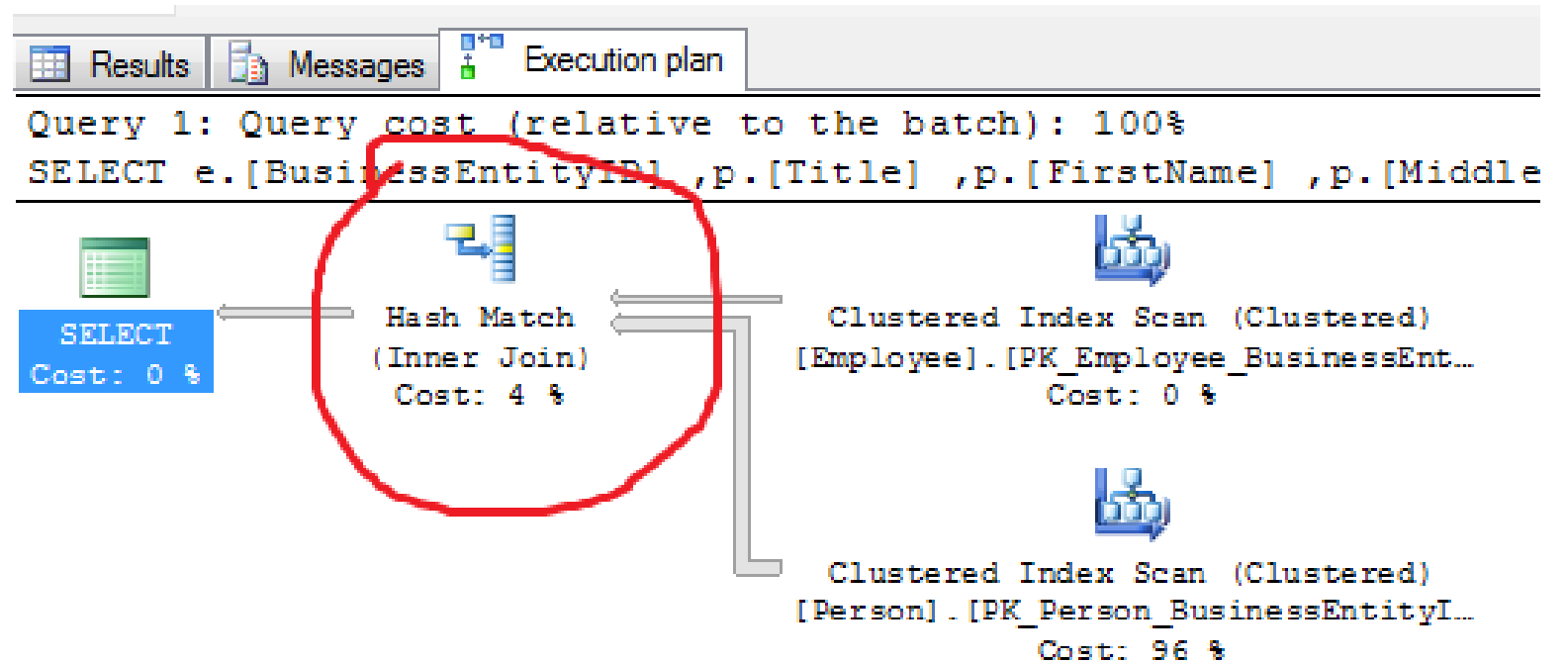
Exemple de
plan
d'exécution
de Loop join



Exemple de plan d'exécution de Merge join



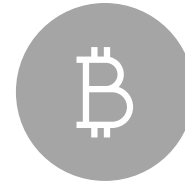
Exemple de plan d'exécution de Hash join



B-Tree and Bitmap index



B-TREE ET BITMAP
SONT DEUX TYPES
D'INDEX UTILISÉS DANS
ORACLE.



BITMAP EST UNE
MÉTHODE
D'INDEXATION
OFFRANT DES
AVANTAGES EN TERMES
DE PERFORMANCES ET
D'ÉCONOMIE DE
STOCKAGE.



L'INDEX B-TREE EST UN
INDEX CRÉÉ SUR DES
COLONNES CONTENANT
DES VALEURS TRÈS
UNIQUES.



B-TREE FONCTIONNE
MIEUX AVEC DE
NOMBREUSES VALEURS
INDEXÉES DISTINCTES



BITMAP FONCTIONNE
MIEUX AVEC DE
NOMBREUSES VALEURS
INDEXÉES DISTINCTES

Créer des index

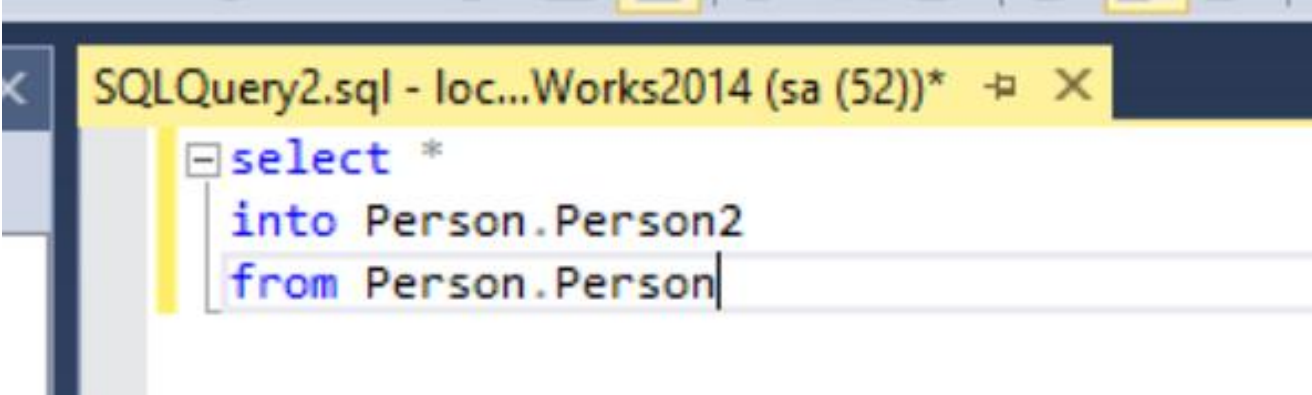
On crée une table Person2

La table créée ne contient pas de PK

et ne contient pas d'index

Select into copie la structure de la table sans les PK et les index

La table Person2 ne possède pas d'index

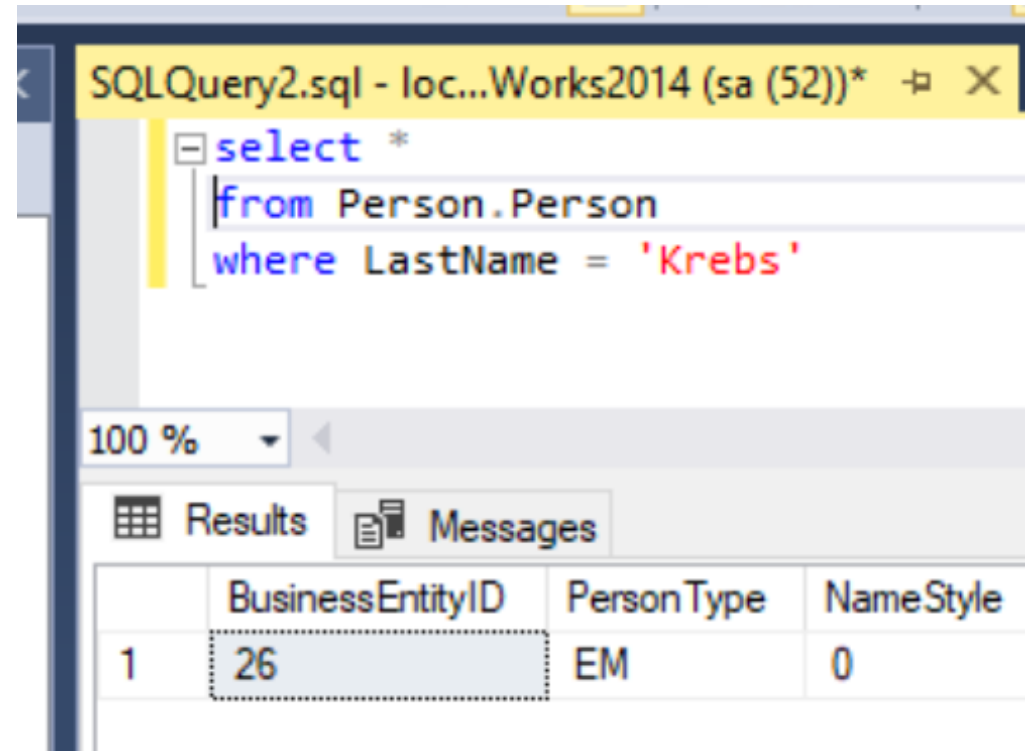
A screenshot of a SQL query window. The title bar reads 'SQLQuery2.sql - loc...Works2014 (sa (52))*'. The query text is:

```
select *  
into Person.Person2  
from Person.Person
```

```
SQLQuery2.sql - loc...Works2014 (sa (52))*  
select *  
into Person.Person2  
from Person.Person
```

index

On va exécuter la requête suivante :



The screenshot shows a SQL Server Enterprise Manager window titled "SQLQuery2.sql - loc...Works2014 (sa (52))*". The query editor contains the following SQL code:

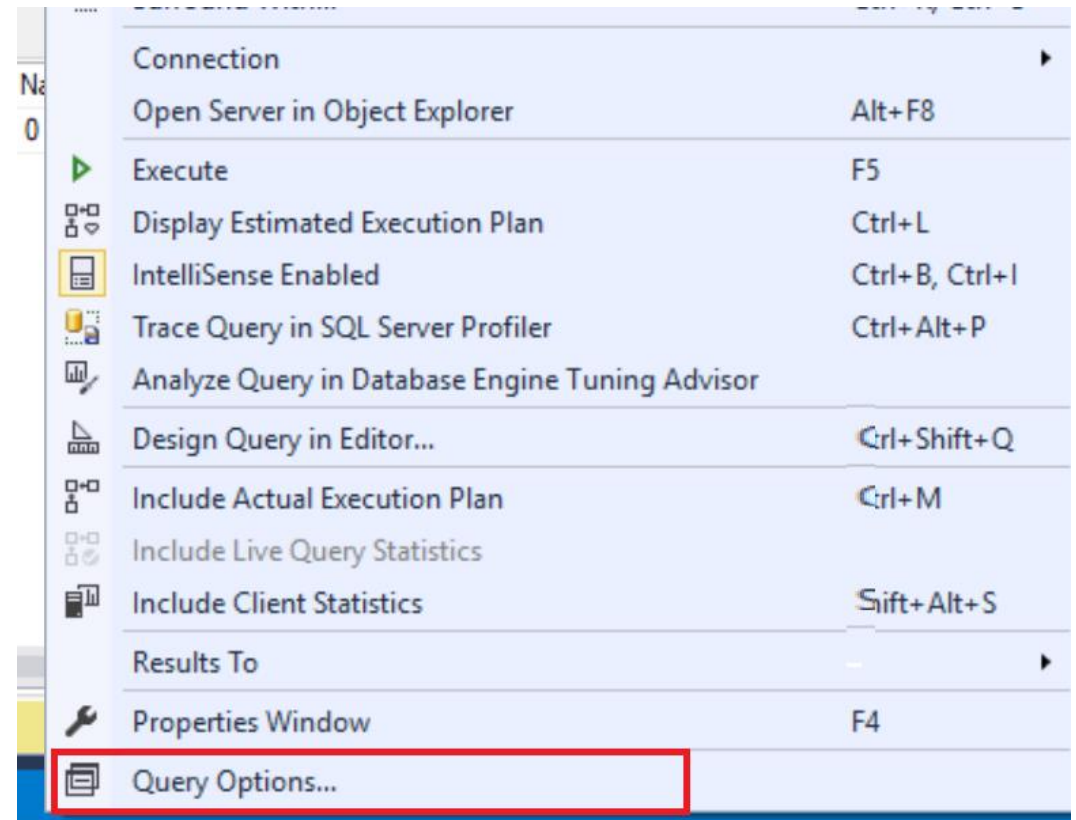
```
select *  
from Person.Person  
where LastName = 'Krebs'
```

Below the query editor, the "Results" tab is selected, displaying a table with the following data:

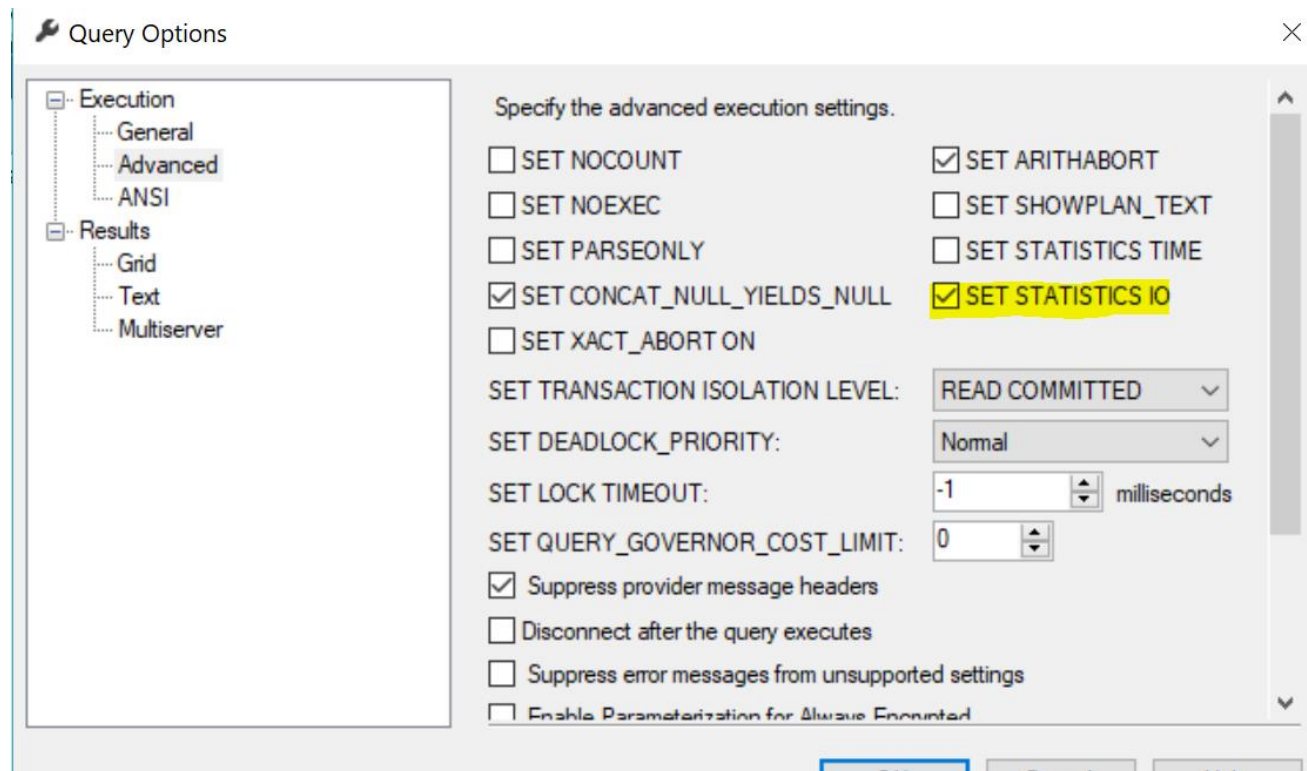
	BusinessEntityID	Person Type	NameStyle
1	26	EM	0

Afficher le plan d'exécution

Clique droit sur la fenêtre de requête et sélectionner query options

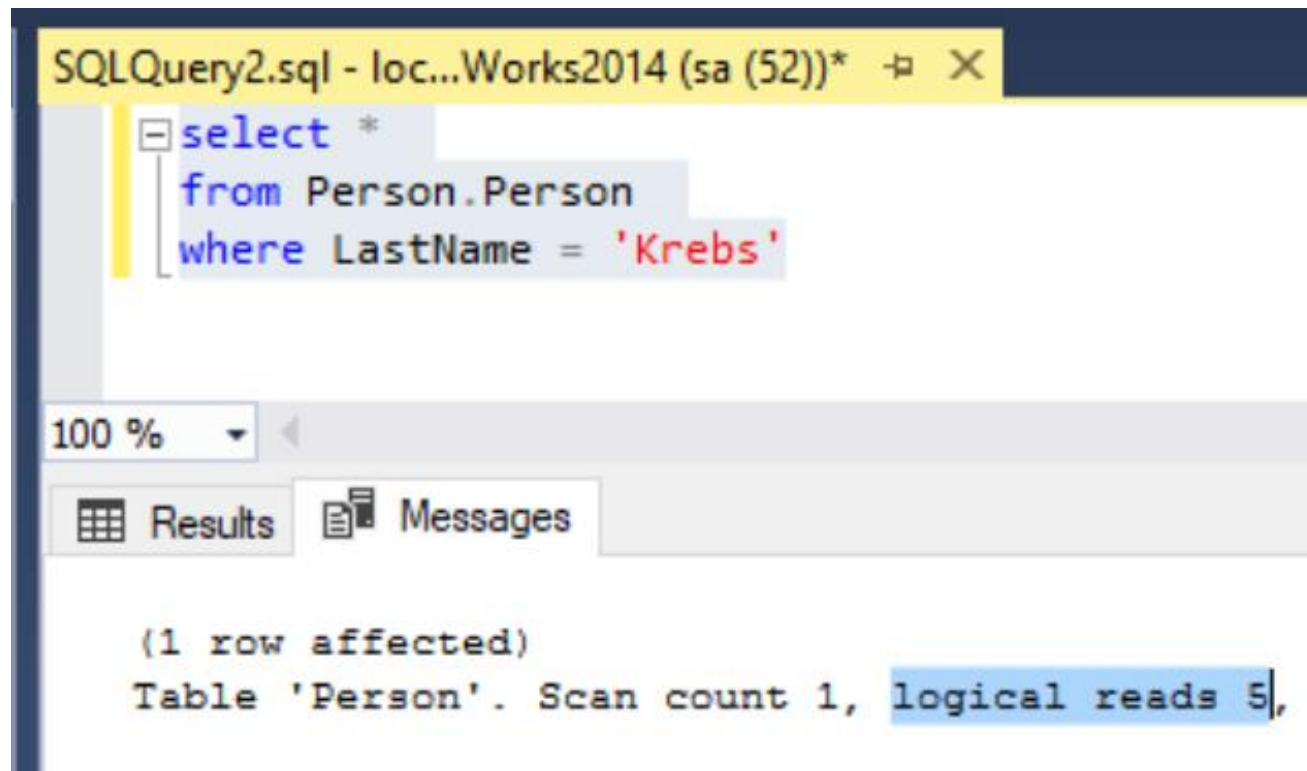


Afficher le plan d'exécution



Choisir set statistics IO

Nombre de lectures logiques avec index



The screenshot shows a SQL Server Enterprise Manager window titled "SQLQuery2.sql - loc...Works2014 (sa (52))*". The query editor contains the following SQL code:

```
select *  
from Person.Person  
where LastName = 'Krebs'
```

Below the query editor, the "Results" tab is selected, displaying the execution statistics:

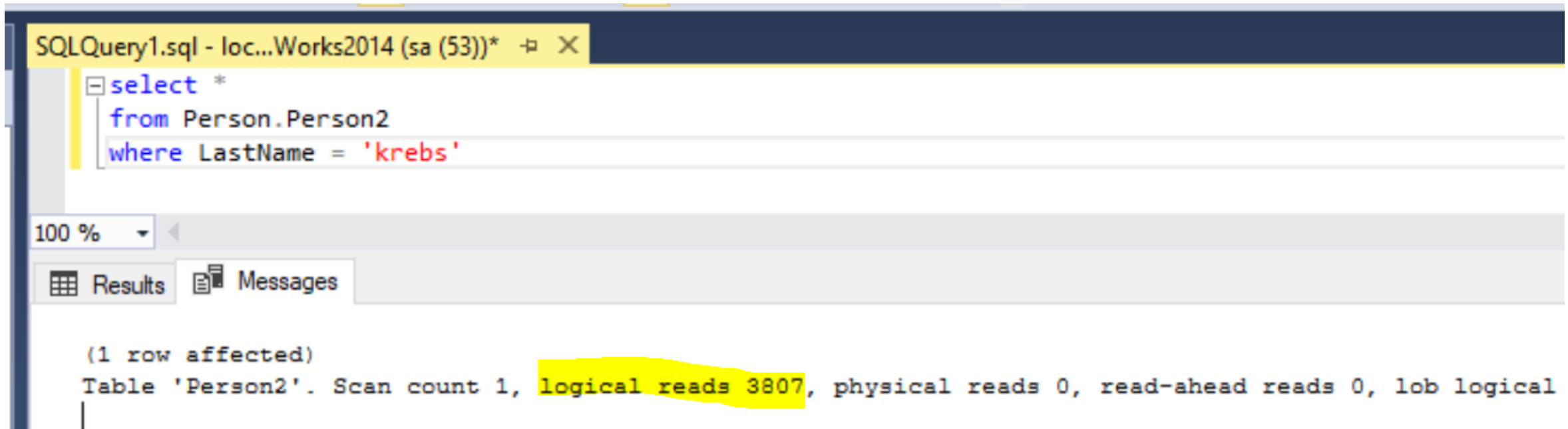
```
(1 row affected)  
Table 'Person'. Scan count 1, logical reads 5,
```

The "logical reads 5" value is highlighted in blue.

Exécuter cette requête : on a 5 lecture logique

Nombre de lectures logiques sans index

Maintenant on exécute cette requête sur la table qui ne contient pas d'index et on voit le nombre de lecture logique



The screenshot shows a SQL Server Enterprise Manager window with a query executed. The query is:

```
select *  
from Person.Person2  
where LastName = 'krebs'
```

The execution statistics at the bottom of the window are:

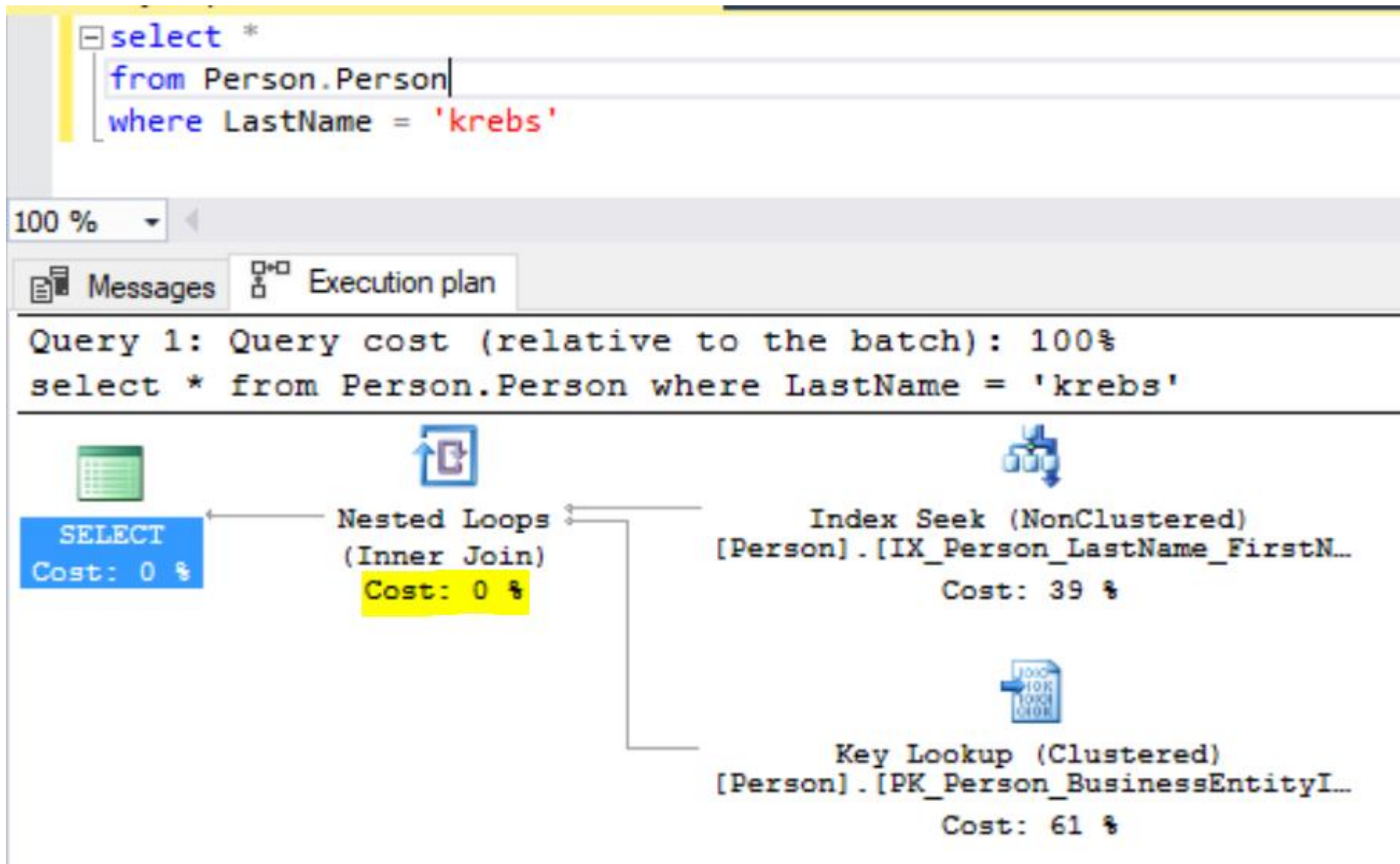
```
(1 row affected)  
Table 'Person2'. Scan count 1, logical reads 3807, physical reads 0, read-ahead reads 0, lob logical
```

The value "logical reads 3807" is highlighted in yellow in the original image.

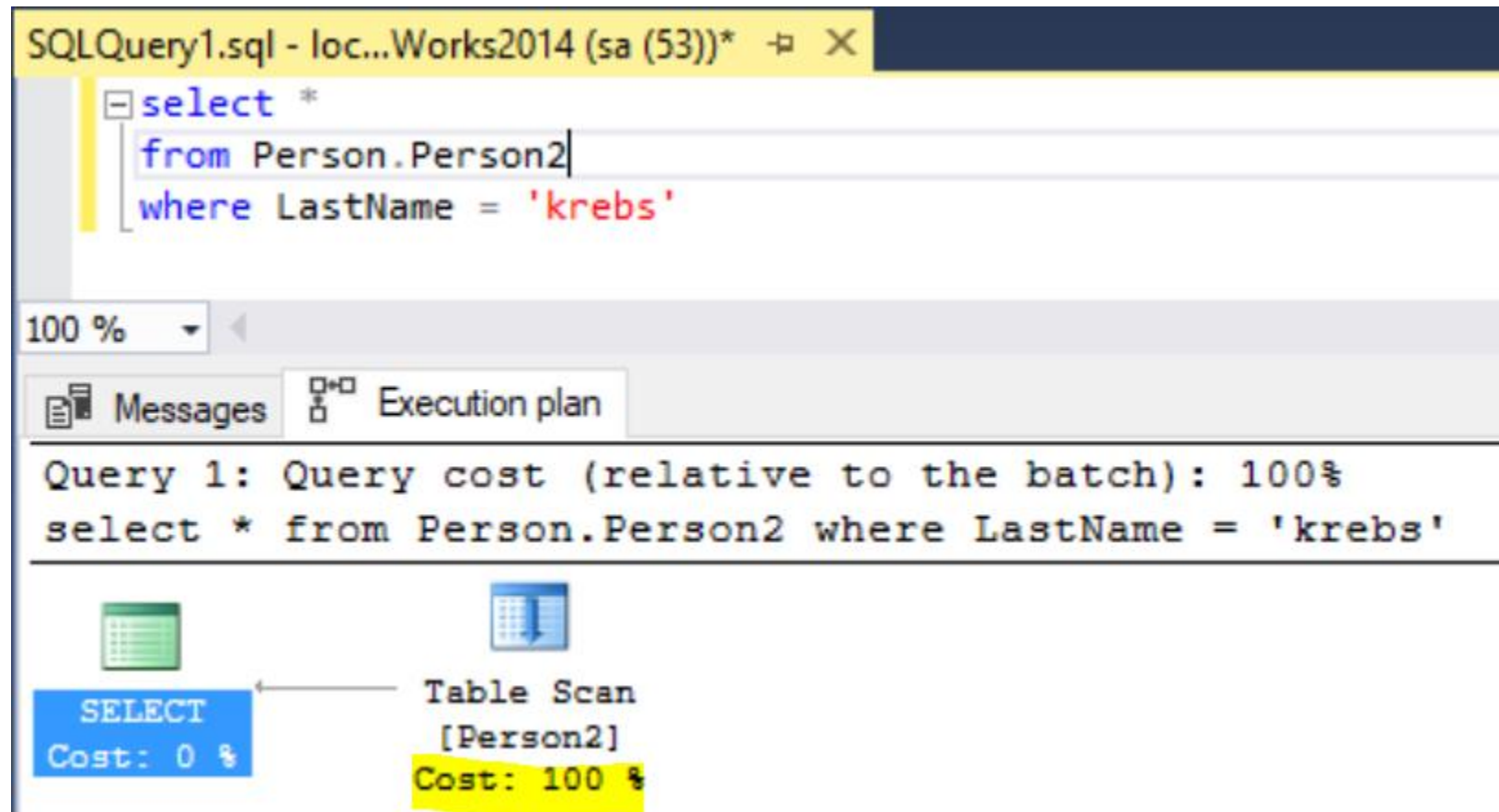
Comparaison des plans d'exécution

On va comparer le plan d'exécution pour les deux tables **Person** qui possède un index et Person2 qui ne possède pas d'index

Plan d'exécution de la table Person (indexé)



Plan d'exécution de la table Person2 (non indexé)

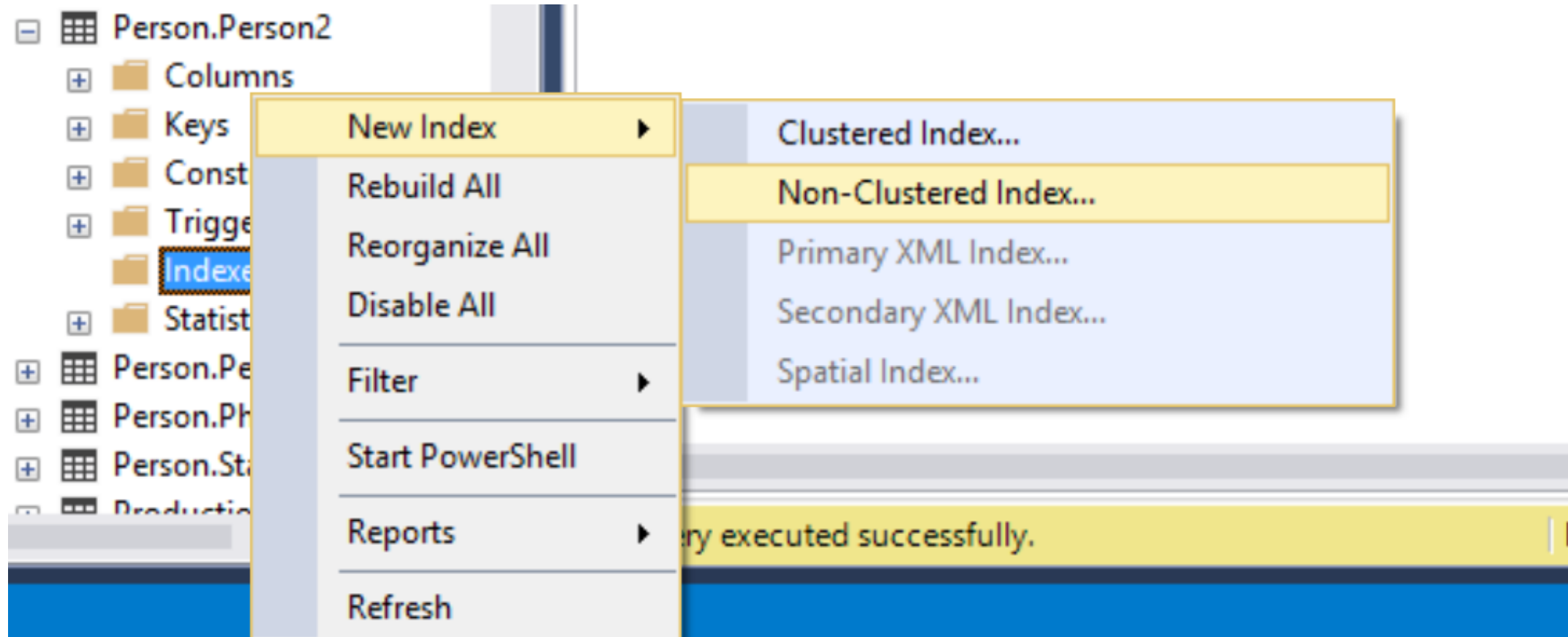


La recherche dichotomique

Si on a pas d'index on est obligé de parcourir toute la table ligne par ligne. Ce qui est très coûteux surtout si le nombre de résultat est très important. Dans un système de production la taille de la table augmente et le temps pris par la requête augmente ce qui diminue la performance.

Si on a une index on utilise **la recherche dichotomique**

La solution: création d'un index



La solution: création d'un index

Select a page

- General
- Options
- Storage
- Filter
- Extended Properties

Connection

localhost [sa]

[View connection properties](#)

Progress

Ready

Script | Help

Table name:
Person2

Index name:
ind_person2_lastname

Index type:
Nonclustered

☐ Unique

Index key columns | Included columns

Name	Sort Order	Data Type	Size	Identity	Allow NULLs
------	------------	-----------	------	----------	-------------

Add...
Remove
Move Up
Move Down

La solution: création d'un index

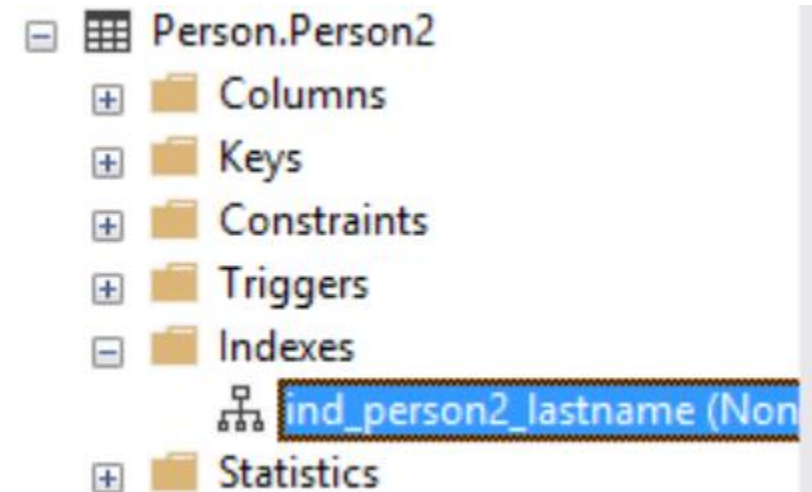
Select Columns from 'Person.Person2' — □ ×

Ready

Select table columns to be added to the index:

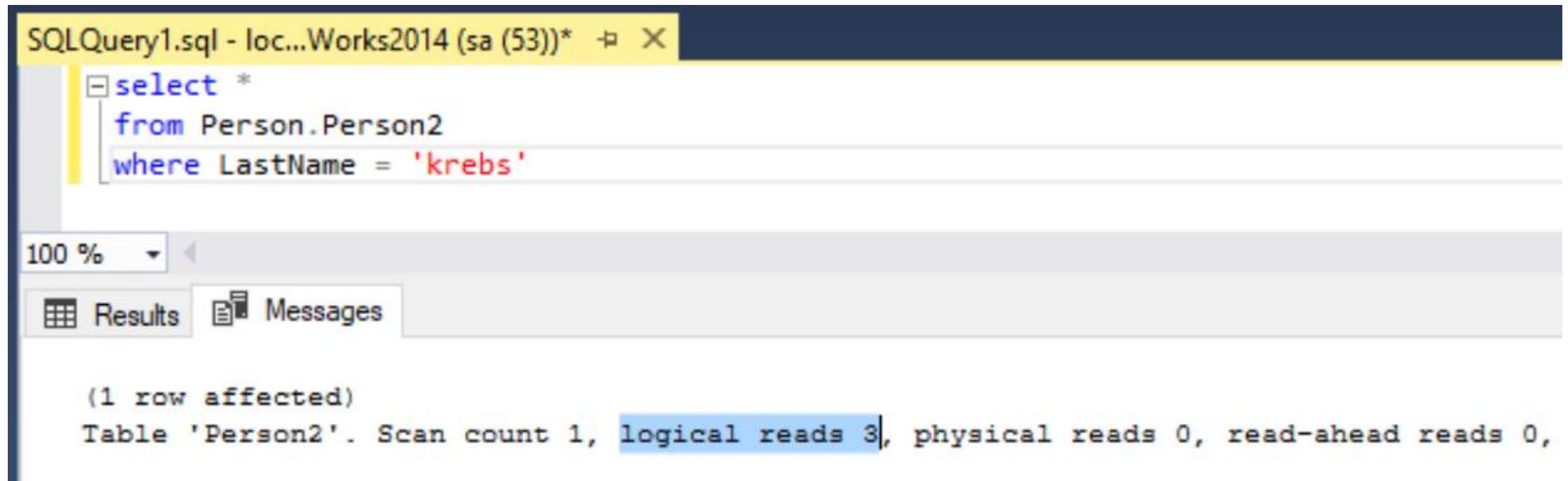
<input type="checkbox"/>	Name	Data Type	Size	Identity	Allow NULLs
<input type="checkbox"/>	BusinessEntityID	int	4	No	No
<input type="checkbox"/>	PersonType	nchar(2)	4	No	No
<input type="checkbox"/>	NameStyle	dbo.NameStyle(bit)	1	No	No
<input type="checkbox"/>	Title	nvarchar(8)	16	No	Yes
<input type="checkbox"/>	FirstName	dbo.Name(nvarchar(50))	100	No	No
<input type="checkbox"/>	MiddleName	dbo.Name(nvarchar(50))	100	No	Yes
<input checked="" type="checkbox"/>	LastName	dbo.Name(nvarchar(50))	100	No	No
<input type="checkbox"/>	Suffix	nvarchar(10)	20	No	Yes
<input type="checkbox"/>	EmailPromotion	int	4	No	No
<input type="checkbox"/>	rowguid	uniqueidentifier	16	No	No
<input type="checkbox"/>	ModifiedDate	datetime	8	No	No

OK Cancel Help



Revoir Person 2 après ajout d'index

Le nombre de lecture après la création de l'index



The screenshot shows a SQL Server query window titled 'SQLQuery1.sql - loc...Works2014 (sa (53))*'. The query is:

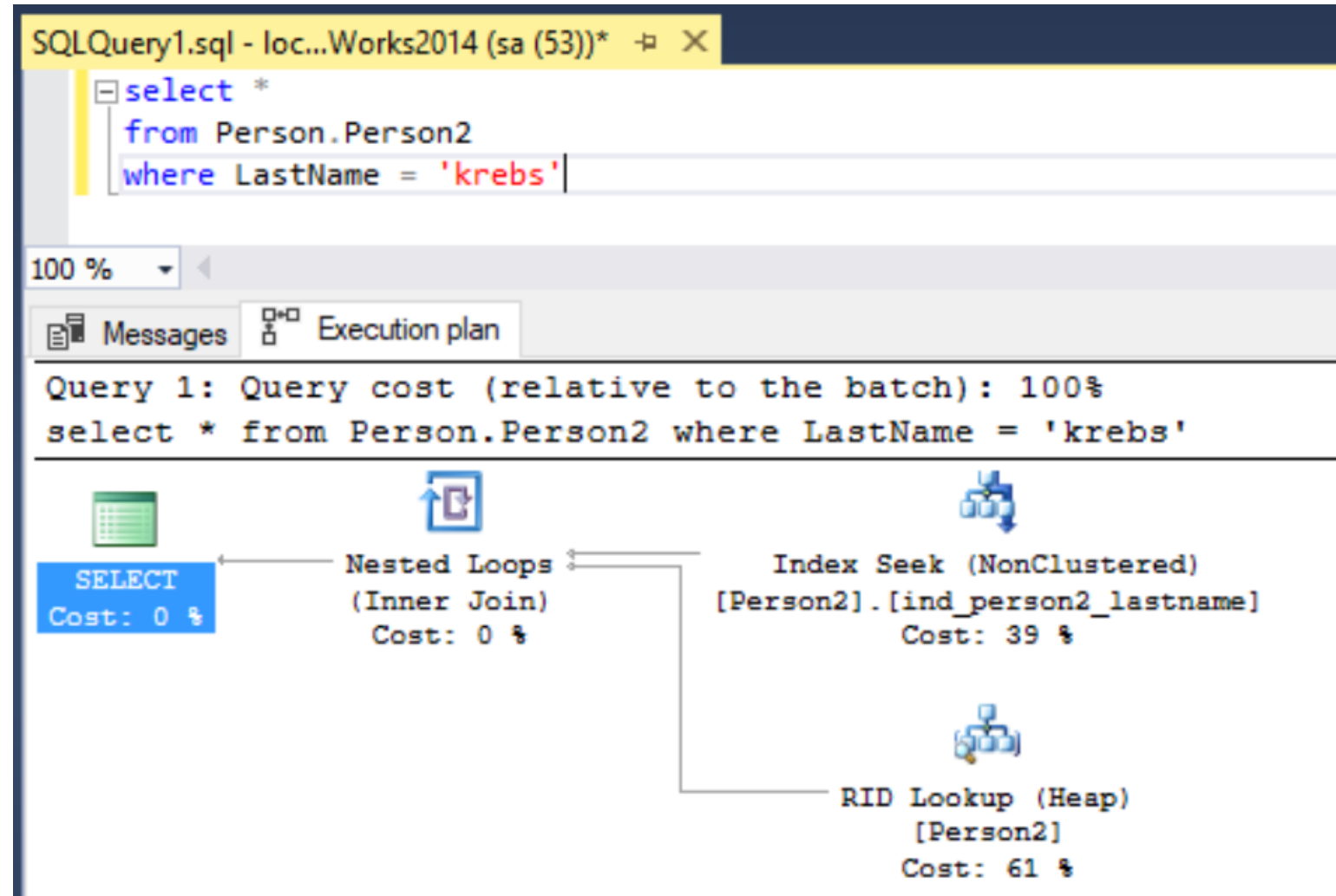
```
select *  
from Person.Person2  
where LastName = 'krebs'
```

Below the query, the 'Results' tab is selected, showing the execution statistics:

```
(1 row affected)  
Table 'Person2'. Scan count 1, logical reads 3, physical reads 0, read-ahead reads 0,
```

The value 'logical reads 3' is highlighted in blue.

Voir le plan d'exécution après la création de l'index



Aborder la recherche dichotomique sur un index

- Algorithme de recherche : la même démarche qu'on fait naturellement lorsqu'on consulte un dictionnaire.
- Si on cherche un mot « parler » précis on va pas parcourir le dictionnaire page par page : c'est un scan , c'est un parcourt de la table
- C'est pas bien et ce n'est pas efficace
- Le dictionnaire est ordonné par un ordre alphabétique
- Il faut d'abord trouver le p
- En quelques recherche on trouve le mot

Explorer la structure d'index

```
select * from Person.Person2 where LastName = 'krebs'
```

Results Messages

(1 row affected)
Table 'Person2'. Scan count 1, logical reads 3, physical reads 3, read-ahead reads 0,

```
select * from sys.indexes i where i.object_id = OBJECT_ID('Person.Person2')
```

Results Messages

object_id	name	index_id	type	type_desc	is_unique	data_space_id	ignore_dup_key
615673241	NULL	0	0	HEAP	0	1	0
615673241	ind_person2_lastname	3	2	NONCLUSTERED	0	1	0

Explorer la structure d'index: structure interne de l'index

La structure : les pages qui le composent

FID: file identifiant, peut être composé de plusieurs PID

PID: page identifiant

Explorer la structure d'index

SQLQuery1.sql - loc...Works2014 (sa (53))*

DBCC IND ('AdventureWorks2014', 'Person.Person2', 3)

100 %

Results Messages

	PageFID	PagePID	IAMFID	IAMPID	ObjectID	IndexID	PartitionNumber	PartitionID	iam_chain_type	PageType
1	1	24153	NULL	NULL	615673241	3	1	72057594059030528	In-row data	10
2	1	28376	1	24153	615673241	3	1	72057594059030528	In-row data	2
3	1	28377	1	24153	615673241	3	1	72057594059030528	In-row data	2
4	1	28378	1	24153	615673241	3	1	72057594059030528	In-row data	2
5	1	28379	1	24153	615673241	3	1	72057594059030528	In-row data	2
6	1	28380	1	24153	615673241	3	1	72057594059030528	In-row data	2
7	1	28381	1	24153	615673241	3	1	72057594059030528	In-row data	2

Explorer la structure d'index

IndexLevel: le niveau le plus élevé est la racine, le point d'entrée, la page correspondant 28408

SQLQuery1.sql - loc...Works2014 (sa (53))*

DBCC IND ('AdventureWorks2014', 'Person.Person2', 3)

100 %

Results Messages

	PagePID	IAMFID	IAMPID	ObjectID	IndexID	PartitionNumber	PartitionID	iam_chain_type	PageType	IndexLevel
27	28401	1	24153	615673241	3	1	72057594059030528	In-row data	2	0
28	28402	1	24153	615673241	3	1	72057594059030528	In-row data	2	0
29	28403	1	24153	615673241	3	1	72057594059030528	In-row data	2	0
30	28404	1	24153	615673241	3	1	72057594059030528	In-row data	2	0
31	28405	1	24153	615673241	3	1	72057594059030528	In-row data	2	0
32	28406	1	24153	615673241	3	1	72057594059030528	In-row data	2	0
33	28407	1	24153	615673241	3	1	72057594059030528	In-row data	2	0
34	28408	1	24153	615673241	3	1	72057594059030528	In-row data	2	1

Explorer la structure d'index

DBCC PAGE ('AdventureWorks2014', 1, 28408, 3);

La base, le FID, PID, un parameter supplémentaire pour déterminer le type d'affichage, 3 pour l'affichage tabulaire.

Cette commande donne le contenu de la page.

SQL server va chercher “krebs” dans la page 28408.

Explorer la structure d'index

```
DBCC PAGE ('AdventureWorks2014', 1, 28408, 3);
```

100 %



Results



Messages

	FileId	PageId	Row	Level	ChildFileId	ChildPageId	LastName (key)	HEAP RID (key)
1	1	28408	0	1	1	28376	NULL	NULL
2	1	28408	1	1	1	28377	Allen	0x086B000001000400
3	1	28408	2	1	1	28378	Alvarez	0xE664000001000400
4	1	28408	3	1	1	28379	Anderson	0xD76A000001000200
5	1	28408	4	1	1	28380	Baker	0x7D65000001000200
6	1	28408	5	1	1	28381	Bennett	0x006D000001000400
7	1	28408	6	1	1	28382	Bradley	0x0E63000001000100
8	1	28408	7	1	1	28383	Bryant	0x2762000001000200
9	1	28408	8	1	1	28384	Cai	0x685F000001000200

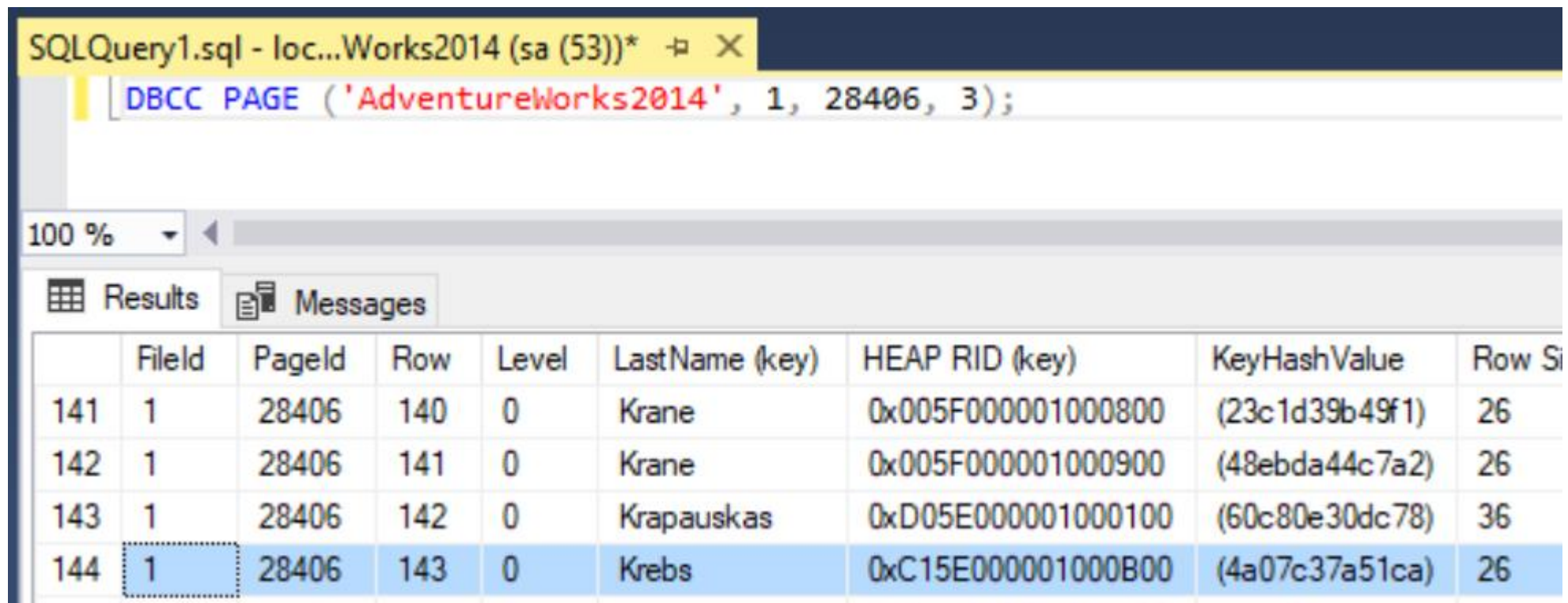
Explorer la structure d'index

« krebs n'est pas ici mais entre « keyser » et « Lal » donc Sql server va chercher dans la page fille: 28406

Results		Messages						
	FileId	PageId	Row	Level	ChildFileId	ChildPageId	LastName (key)	HEAP RID (key)
25	1	28408	24	1	1	28400	Hensien	0x365F000001000600
26	1	28408	25	1	1	28401	Hill	0x696A000001000300
27	1	28408	26	1	1	28402	Huang	0x9F6A000001000200
28	1	28408	27	1	1	28403	Jai	0x3E64000001000100
29	1	28408	28	1	1	28404	Jenkins	0xBA67000001000200
30	1	28408	29	1	1	28405	Jones	0x306B000001000300
31	1	28408	30	1	1	28406	Keyser	0xC65E000001000B00
32	1	28408	31	1	1	28407	Lal	0x2660000001000200

Explorer la structure d'index

HEAP RID (row ID): identifiant de ligne, référence pour aller chercher la ligne dans la table.



SQLQuery1.sql - loc...Works2014 (sa (53))*

```
DBCC PAGE ('AdventureWorks2014', 1, 28406, 3);
```

100 %

Results Messages

	FileId	PageId	Row	Level	LastName (key)	HEAP RID (key)	KeyHashValue	Row Si
141	1	28406	140	0	Krane	0x005F000001000800	(23c1d39b49f1)	26
142	1	28406	141	0	Krane	0x005F000001000900	(48ebda44c7a2)	26
143	1	28406	142	0	Krapauskas	0xD05E000001000100	(60c80e30dc78)	36
144	1	28406	143	0	Krebs	0xC15E000001000B00	(4a07c37a51ca)	26

Explorer la structure d'index

Au total : 3 lectures pour trouver le résultat:

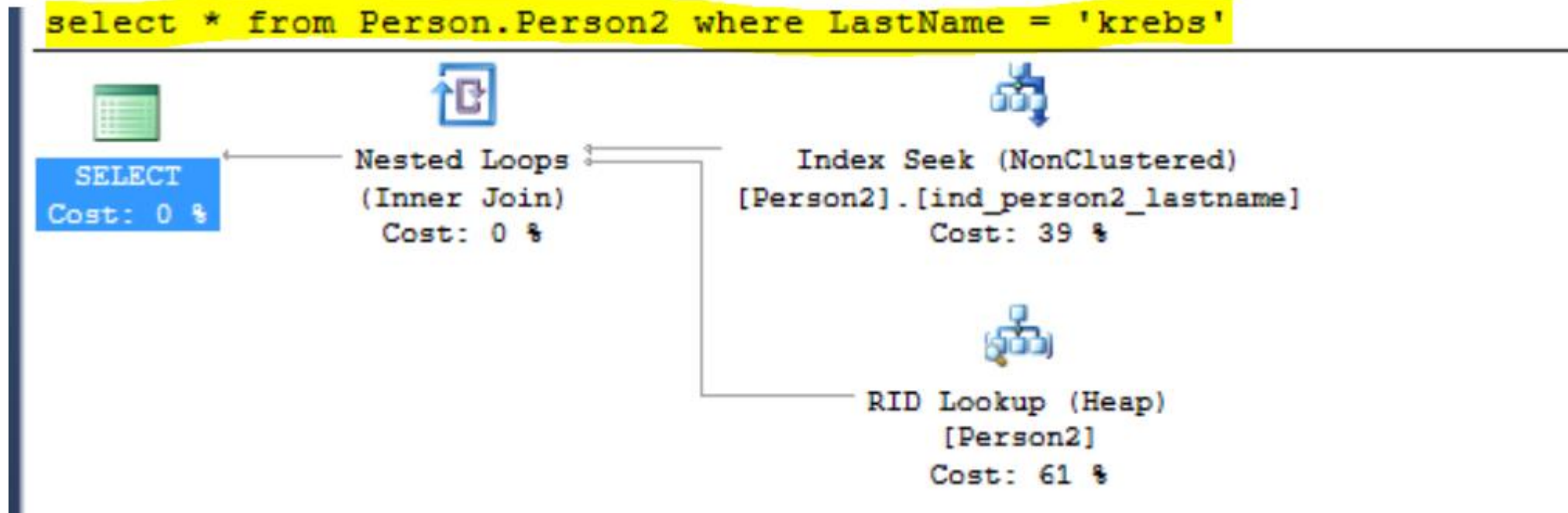
```
DBCC PAGE ('AdventureWorks2014', 1, 28408, 3);
DBCC PAGE ('AdventureWorks2014', 1, 28406, 3);
```

100 %

Results Messages

	Field	PageId	Row	Level	LastName (key)	HEAP RID (key)	KeyHashValue	Row Size
141	1	28406	140	0	Krane	0x005F000001000800	(23c1d39b49f1)	26
142	1	28406	141	0	Krane	0x005F000001000900	(48ebda44c7a2)	26
143	1	28406	142	0	Krapauskas	0xD05E000001000100	(60c80e30dc78)	36
144	1	28406	143	0	Krebs	0xC15E000001000B00	(4a07c37a51ca)	26

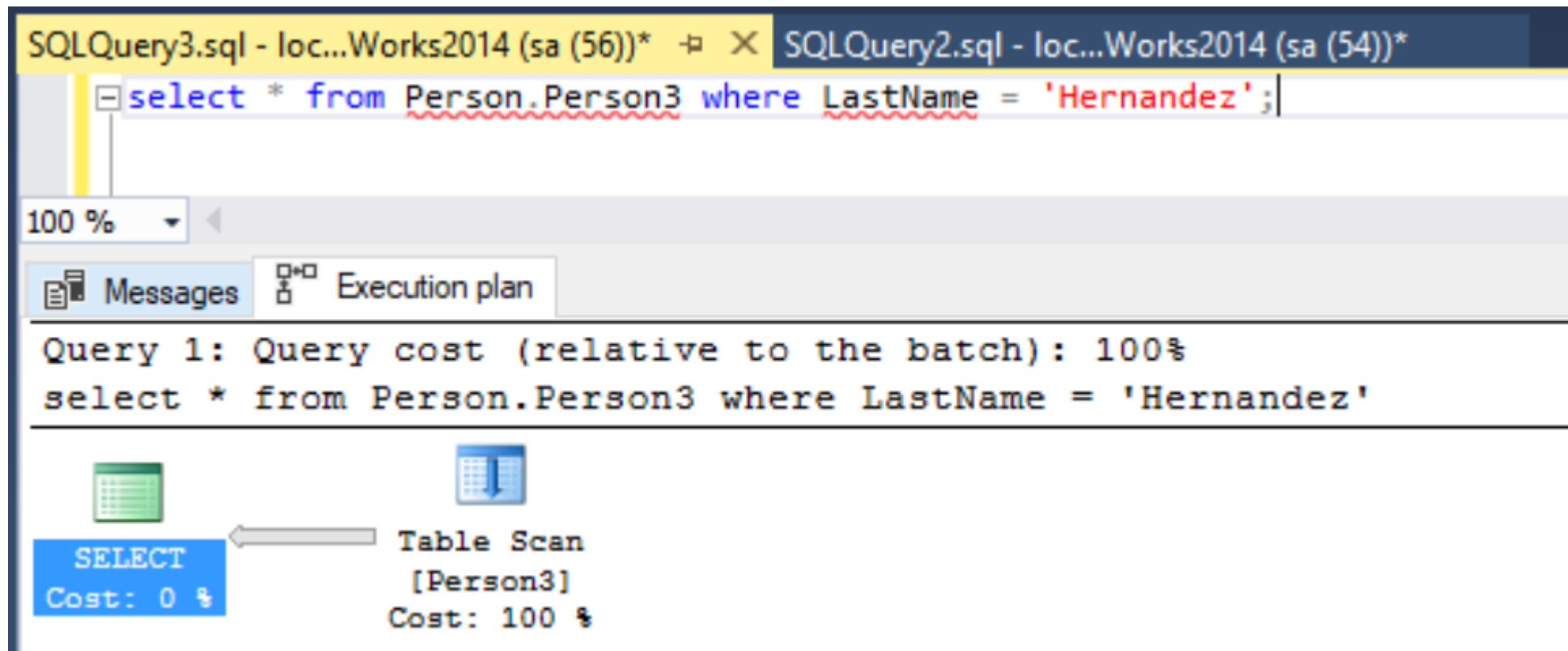
Le choix de l'optimiseur



A chaque fois qu'on trouve le RID on fait une recherche mais cette requête retourne un résultat d'où le choix de nested loops

Le choix de l'optimiseur

Il y a pas de garanti que l'index sera utilisé



The screenshot shows the SQL Server Enterprise Manager interface. At the top, two tabs are visible: 'SQLQuery3.sql - loc...Works2014 (sa (56))*' and 'SQLQuery2.sql - loc...Works2014 (sa (54))*'. The active tab displays the following SQL query:

```
select * from Person.Person3 where LastName = 'Hernandez';
```

Below the query editor, the 'Execution plan' tab is selected. It shows the following text:

```
Query 1: Query cost (relative to the batch): 100%  
select * from Person.Person3 where LastName = 'Hernandez'
```

The execution plan diagram is displayed below the text. It consists of two nodes:

- A 'SELECT' node (represented by a green icon) with a cost of 0%.
- A 'Table Scan [Person3]' node (represented by a blue icon) with a cost of 100%.

An arrow points from the 'Table Scan' node to the 'SELECT' node, indicating the data flow.

Découvrir le choix par l'optimiseur

Selon le nombre de lignes qui seront affectées par cette requête l'optimiseur peut choisir n'est pas utilisé l'index

Si l'optimiseur estime que parcourir la table est moins coûteux il va utiliser cette stratégie sinon il utilise l'index

Donc l'index est utilisé si l'optimiseur estime (en fonction de nombre de ligne) que l'index est sélective.

Découvrir le choix par l'optimiseur

Exécuter les requêtes suivantes et voir le plan d'exécution:

- `select * from Person.Person3;`
- `select p.LastName from Person.Person3 p;`
- `select p.FirstName from Person.Person3 p;`

Index couvrant

Il répond à tous les besoins de la requêtes:

Dans cette requête l'index n'est pas couvrant :

```
select * from Person.Person3 p where LastName = 'hernandez';
```

Mais dans cette requête il est couvrant :

```
select p.LastName from Person.Person3 p where LastName = 'hernandez';
```

La notion d'index couvrant est très important pour la performance:

- La recherche se fait sur l'index et non sur la table
- Pas de verouillage de la table

Exploiter la table heap

//On va créer une table sans aucune contrainte ni index ni clé

```
create table Person.test(  
id int not null identity(1,1),  
texte char(20) not null default ('commentaire')  
);
```

```
go
```

```
insert into Person.test default values;
```

```
go 10 -- execute la requête 10 fois
```

```
select * from Person.test
```

```
go
```


Exploiter la table heap

```
delete from Person.test where id = 5;
```

```
go
```

```
insert into Person.test default values;
```

```
go
```

//Cette nouvelle valeur va s'insérer où ?

Exploiter la table heap

//Le moteur de stockag va le mettre dans l'emplacement libre si elle est suffisant

```
select * from Person.test  
go
```

Il n'y a pas d'ordre de ligne, on peut avoir un ordre d'affichage en utilisant order by

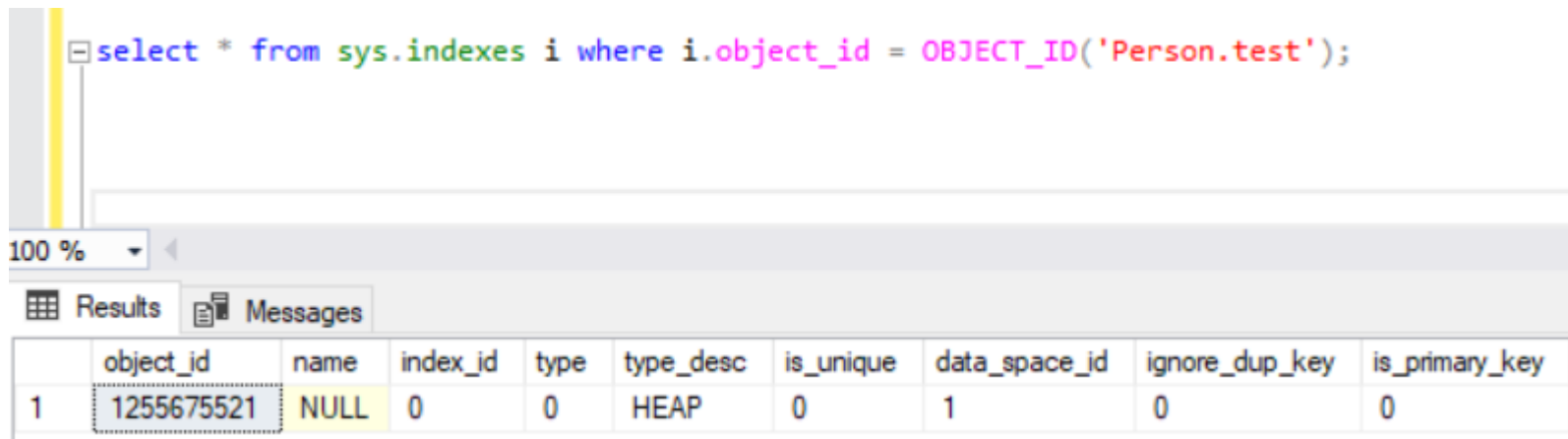
Cette table n'a pas d'index clustered

Exploiter la table heap

SQL Server considère toute table comme une forme d'index, c'est un index de type particulier, **c'est un index de type HEAP**

HEAP = tas, ensemble (en français), un ensemble de ligne sans ordre.

Une table est un HEAP = un ensemble non ordonné de lignes



The screenshot shows a SQL query window with the following query:

```
select * from sys.indexes i where i.object_id = OBJECT_ID('Person.test');
```

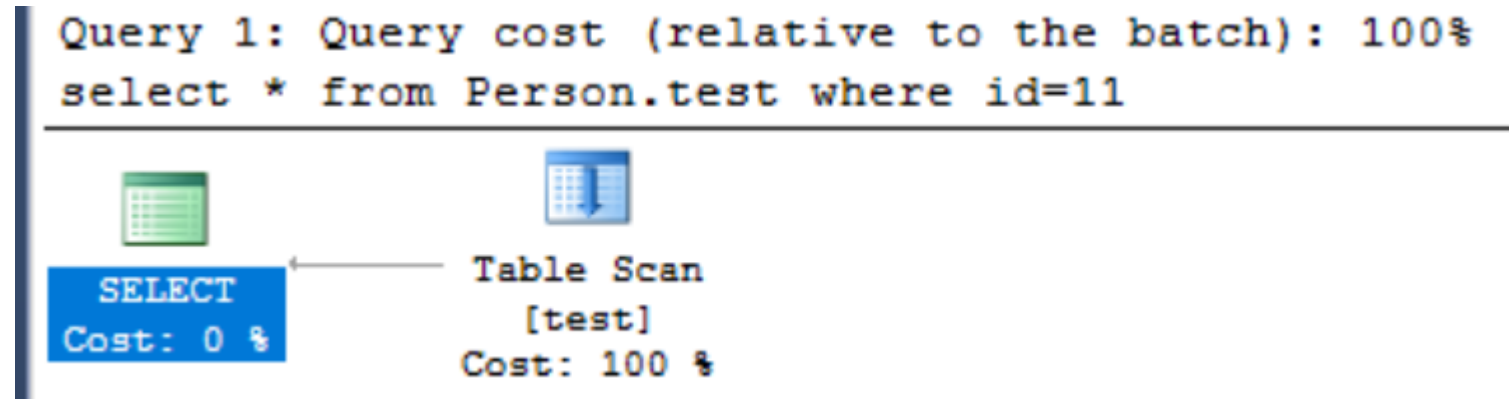
Below the query window, the 'Results' tab is active, displaying a table with 10 columns and 1 row. The columns are: object_id, name, index_id, type, type_desc, is_unique, data_space_id, ignore_dup_key, is_primary_key. The row contains the values: 1, 1255675521, NULL, 0, 0, HEAP, 0, 1, 0, 0.

	object_id	name	index_id	type	type_desc	is_unique	data_space_id	ignore_dup_key	is_primary_key
1	1255675521	NULL	0	0	HEAP	0	1	0	0

Ajouter un index clustered

Regarder le plan d'exécution de cette requête :

select * from Person.test where id=11



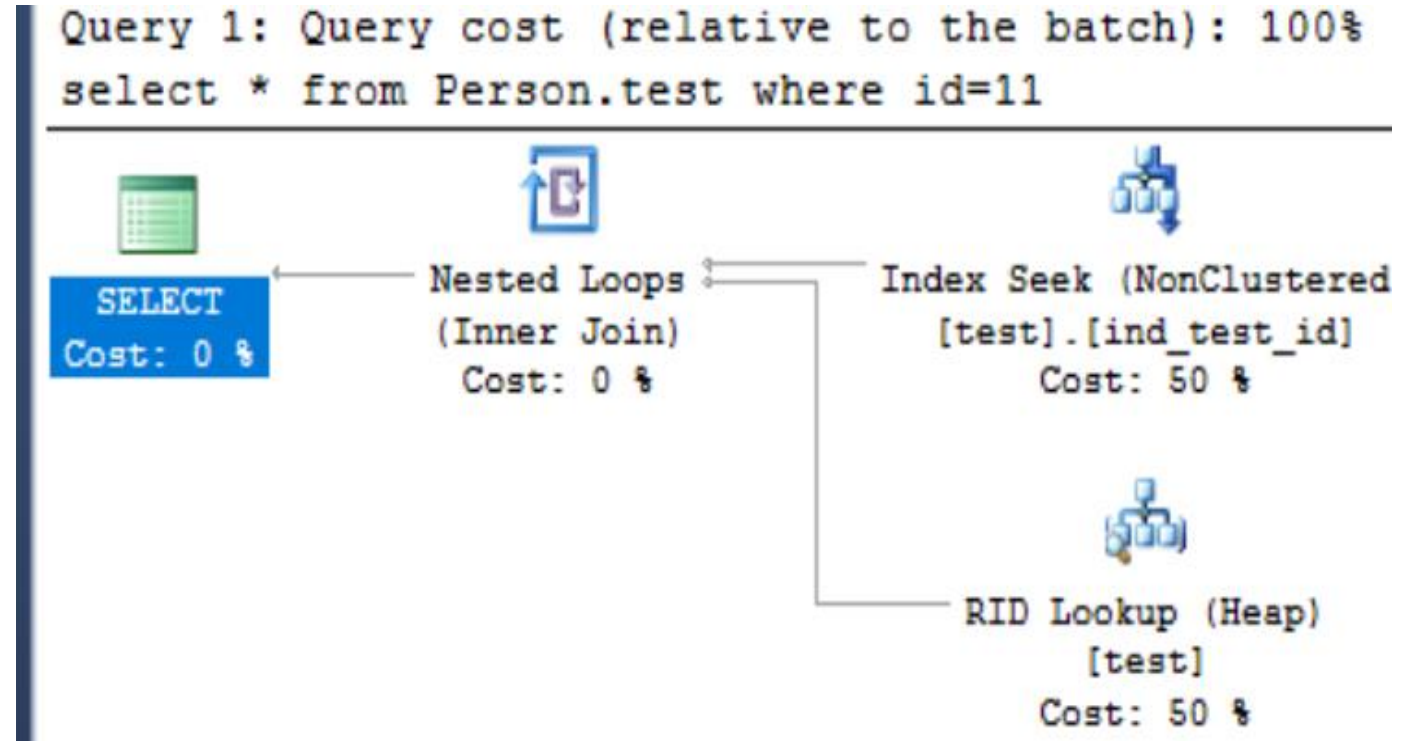
Ajouter un index clustered

Exécuter cette commande:

**create unique index
ind_test_id on Person.test (id)**

Réexécuter la commande :


**select * from Person.test
where id=11**



Ajouter un index clustered

Voir le plan d'exécution de cette requête :

select * from Person.test

Query 1: Query cost (re select * from Person.te		Estimated Execution Mode		row
 <p>SELECT Cost: 0 %</p> <p>Table Scan [test] Cost: 100 %</p>		Storage	RowStore	
		Estimated I/O Cost	0,003125	
		Estimated Operator Cost	0,003293 (100%)	
		Estimated CPU Cost	0,000168	
		Estimated Subtree Cost	0,003293	
		Estimated Number of Executions	1	
		Estimated Number of Rows	10	
		Estimated Row Size	31 B	
		Ordered	False	
		Node ID	0	

Ajouter un index clustered

Exécuter cette requête :

```
drop index ind_test_id on Person.test;  
go
```

Et puis

```
create unique clustered index cl_ind_test_id on Person.test (id)  
go
```

Lorsqu'on crée un index clustered le mot clé « clustered » est obligatoire

Ajouter un index clustered

Exécuter cette requête :

select * from Person.test

Go

Qu'est vous remarquez ?

```
select * from Person.test
go
```

100 %

Results Messages

	id	texte
1	1	commentaire
2	2	commentaire
3	3	commentaire
4	4	commentaire
5	6	commentaire
6	7	commentaire
7	8	commentaire
8	9	commentaire
9	10	commentaire
10	11	commentaire

Clustered index

On ne voit pas la table (HEAP) car la table et l'index clustered sont la même chose

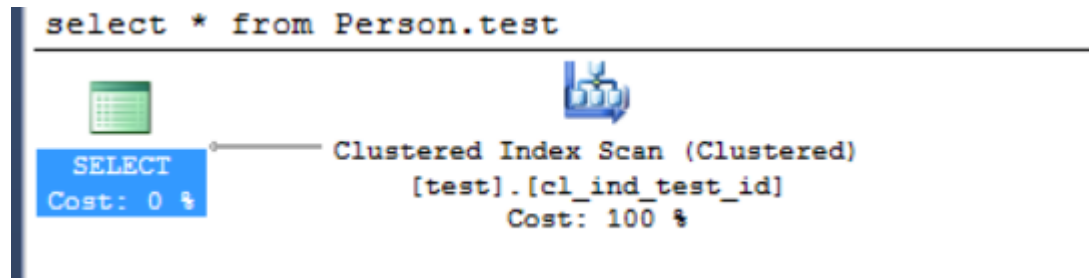
```
select * from sys.indexes i where i.object_id = OBJECT_ID('Person.test');  
go
```

100 %

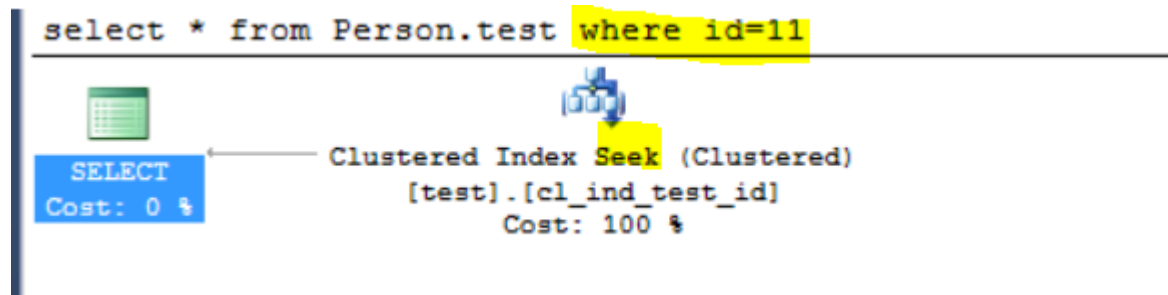
Results Messages

	object_id	name	index_id	type	type_desc	is_unique	data_space_id	ignore_dup_key	is_primary_key	is_unique_constraint
1	1255675521	cl_ind_test_id	1	1	CLUSTERED	1	1	0	0	0

Plan d'exécution d'un index clustered



On fait une recherche pas un scan



On fait pas de boucle car l'index n'est pas couvrant

Différence entre indexes clustered et non clustered

Il est essentiel de comprendre la différence pour bien utiliser l'indexation afin d'optimiser les requêtes

L'index clustered est comme le dictionnaire: la place où se trouve la clé, se trouve la définition

L'index non clustered c'est comme l'index en fin du livre où on trouve le numéro de la page ensuite on cherche le contenu

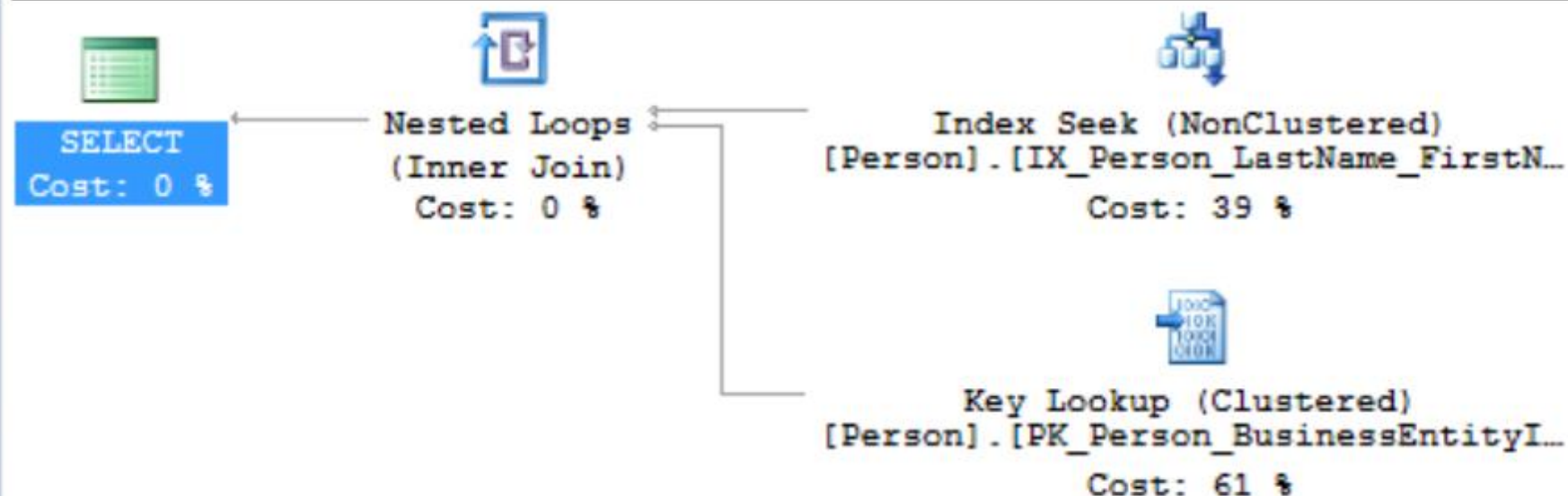
Primary Key et un index clustered

En SQL Server le PK est par défaut un index clustered, on peut changer ce comportement.

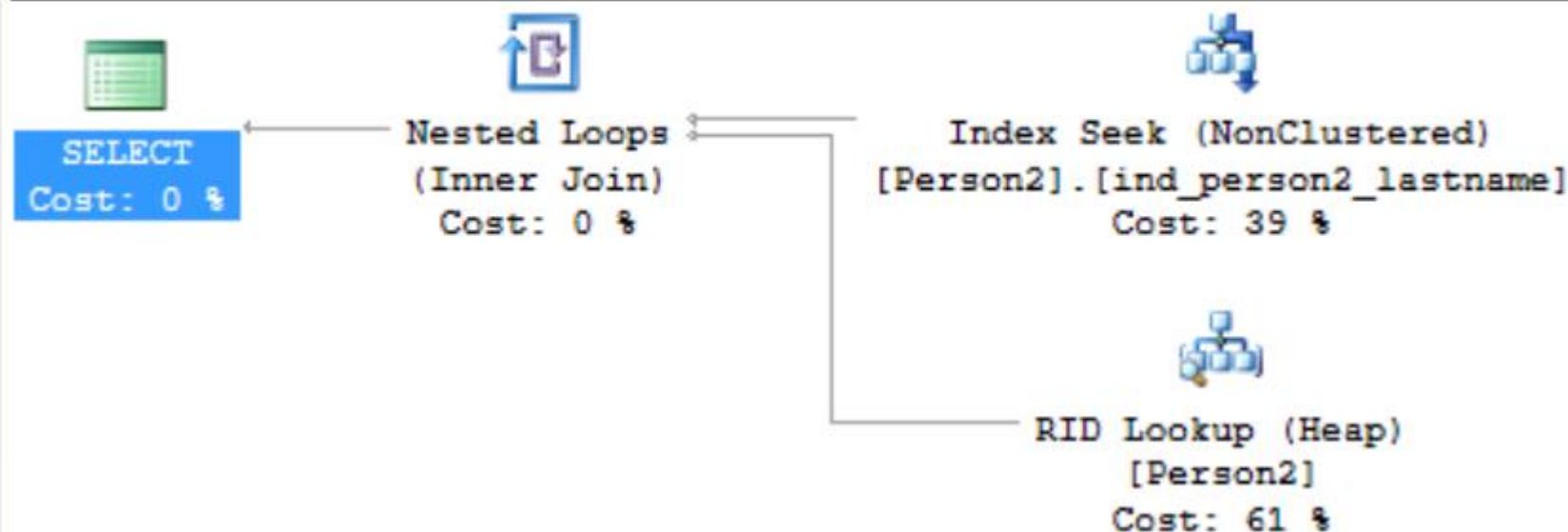
PK est une contrainte du modèle relationnel, une contrainte logique d'unicité.

Un index clustered est une structure physique

```
select * from Person.Person where LastName = 'krebs'
```



```
select * from Person.Person2 where LastName = 'krebs'
```



RID vs Key Lookup

