INDEX SCAN – INDEX SEEK – KEY LOOKUP

Which one? It depends on the table size and the number of rows that need a lookup.
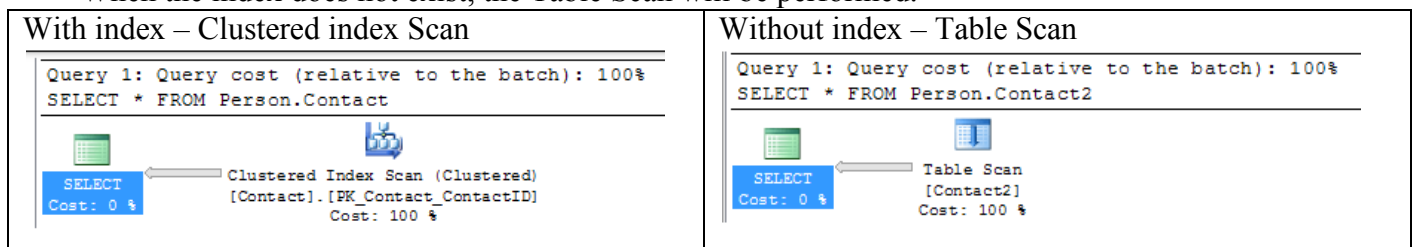Index scan should be eliminated by Index seek, which is better and also introduce a Key Lookup which is something should also be eliminated to improve performance.

Try to avoid key lookup (due to the cost). Key Lookup appears in the queries in which the fields/columns involved (in select, join, where, order by, …) are not part of a clustered/non-clustered index. This can be handled with the covering indexes or by using the INCLUDE clause in create index statement.

Index Scan = Table Scan
- Retrieves all the rows from the table
- when SQL Server has to scan the data or index pages to find the appropriate records.
- the opposite of a seek (index seek)
- take longer to complete
- by finding and fixing the Index Scans/Table Scans, the performance will be improved, especially for larger tables
- Index Scan for Clustered or NonClustered indexes
- Since a scan touches every row in the table, whether or not it qualifies, the cost is proportional to the total number of rows in the table. Thus, a scan is an efficient strategy if the table is small or if most of the rows qualify for the predicate.
- Index Scan is nothing but scanning on the data pages from the first page to the last page. If there is an index on a table, and if the query is touching a larger amount of data, which means the query is retrieving more than 50 percent or 90 percent of the data, and then the optimizer would just scan all the data pages to retrieve the data rows. If there is no index, then you might see a Table Scan (Index Scan) in the execution plan.
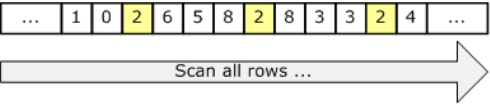-
When the index does not exist, the Table Scan will be performed.
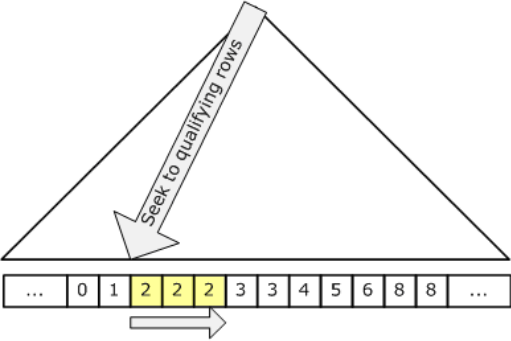
With index – Clustered index Scan	Without index – Table Scan
Query 1: Query cost (relative to the batch): 100% SELECT * FROM Person.Contact Clustered Index Scan (Clustered) SELECT [Contact].[PK_Contact_ContactID] Cost: 0 % Cost: 100 %	Query 1: Query cost (relative to the batch): 100% SELECT * FROM Person.Contact2 Table Scan SELECT [Contact2] Cost: 0 % Cost: 100 %

Index Seek (Clustered)
- Retrieves selective rows from the table
- A seek uses the index to pinpoint the records that are needed to satisfy the query
- It is better than an Index Scan
- Since a seek only touches rows that qualify and pages that contain these qualifying rows, the cost is proportional to the number of qualifying rows and pages rather than to the total number of rows in the table.

- Index seeks are generally preferred for the highly selective queries. What that means is that the query is just requesting a fewer number of rows or just retrieving the other 10 (some documents says 15 percent) of the rows of the table.
- In general query optimizer tries to use an Index Seek which means that the optimizer has found a useful index to retrieve recordset. But if it is not able to do so either because there is no index or no useful indexes on the table, then SQL Server has to scan all the records that satisfy the query condition (Index Scan).
-

Index Scan	Index Seek
The text showplan for a query using a scan: |–Table Scan(OBJECT:([ORDERS]), WHERE:([ORDERKEY]=(2))) The following figure illustrates the scan: ... 1 0 2 6 5 8 2 8 3 3 2 4 ... Scan all rows ...	The text showplan for the same query using a seek: |–Index Seek(OBJECT:([ORDERS].[OKEY_IDX]), SEEK:([ORDERKEY]=(2)) ORDERED FORWARD) *Seek to qualifying rows* ... 0 1 2 2 2 3 3 4 5 6 8 8 ...

Key Lookup (Clustered)
- similar to a clustered index seek ('seek with lookup'), but Key Lookup may specify an additional PRE-FETCH argument instructing the execution engine to prefetch more keys in the clustered
- normal in the Nested Loops (join's)
- are expensive (are used when a small percentage of the table fits the Where clause)
- A key lookup occurs when data is found in a non-clustered index, but additional data is needed from the clustered index to satisfy the query and therefore a lookup occurs. If the table does not have a clustered index then a RID Lookup occurs instead. The reason you would want to eliminate Key/RID Lookups is because they require an additional operation to find the data and may also require additional I/O. I/O is one of the biggest performance hits on a server and any way you can eliminate or reduce I/O is a performance gain.
- Key Lookup appears in the queries in which the fields/columns involved (in select, join, where, order by, …) are not part of a clustered/non-clustered index. This can be handled with the covering indexes or by using the INCLUDE clause in create index statement.
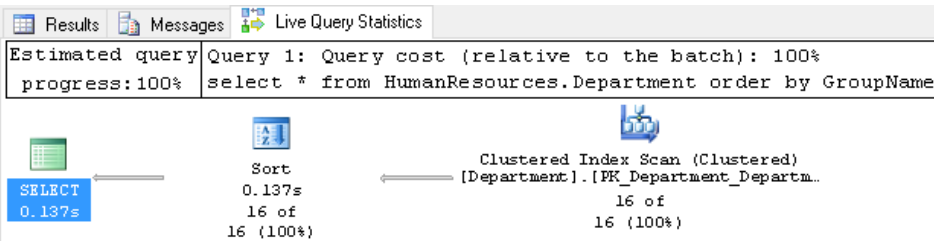
Examples:

```
use AdventureWorks2012
go
```

2

```sql
-- without index
select * from HumanResources.Department order by GroupName
```

Clustered Index Scan (Clustered)
Scanning a clustered index, entirely or only a range.
Estimated operator progress: 100%

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	16
Estimated I/O Cost	0.003125
Estimated Operator Cost	0.0032996 (22%)
Estimated Subtree Cost	0.0032996
Estimated CPU Cost	0.0001746
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	16
Estimated Row Size	123 B
Ordered	False
Node ID	1

Object
[AdventureWorks2012].[HumanResources].[Department].
[PK_Department_DepartmentID]
Output List
[AdventureWorks2012].[HumanResources].
[Department].DepartmentID, [AdventureWorks2012].
[HumanResources].[Department].Name,
[AdventureWorks2012].[HumanResources].
[Department].GroupName, [AdventureWorks2012].
[HumanResources].[Department].ModifiedDate

Results | Messages | Live Query Statistics

Estimated query progress: 100%

Query 1: Query cost (relative to the batch): 100%
select * from HumanResources.Department order by GroupName

SELECT
0.137s

Sort
0.137s
16 of
16 (100%)

Clustered Index Scan (Clustered)
[Department].[PK_Department_Departm...
16 of
16 (100%)

```sql
-- create index
IF EXISTS (SELECT name FROM sys.indexes WHERE name = N'nidx_gn')
    DROP INDEX nidx_gn ON HumanResources.Department
GO
CREATE NONCLUSTERED INDEX nidx_gn ON HumanResources.Department (GroupName);
GO
```

```sql
-- with index
select * from HumanResources.Department order by GroupName
```

Results | Messages | Live Query Statistics

Estimated query progress: 100%

Query 1: Query cost (relative to the batch): 100%
-- with index select * from HumanResources.Department order by GroupName

SELECT

Nested Loops
(Inner Join)
16 of
16 (100%)

Index Scan (NonClustered)
[Department].[nidx_gn]
16 of
16 (100%)

Key Lookup (Clustered)
[Department].[PK_Department_Departm...
16 of
1 (1600%)

Nested Loops
For each row in the top (outer) input, scan the bottom (inner) input, and output matching rows.
Estimated operator progress: 100%

Physical Operation	Nested Loops
Logical Operation	Inner Join
Estimated Execution Mode	Row
Actual Number of Rows	16
Estimated I/O Cost	0
Estimated Operator Cost	0.0000669 (1%)
Estimated Subtree Cost	0.0090211
Estimated CPU Cost	0.0000669
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	16
Estimated Row Size	123 B
Node ID	0

Output List
[AdventureWorks2012].[HumanResources].
[Department].DepartmentID, [AdventureWorks2012].
[HumanResources].[Department].Name,
[AdventureWorks2012].[HumanResources].
[Department].GroupName, [AdventureWorks2012].
[HumanResources].[Department].ModifiedDate
Outer References
[AdventureWorks2012].[HumanResources].
[Department].DepartmentID

Index Scan (NonClustered)
Scan a nonclustered index, entirely or only a range.
Estimated operator progress: 100%

Physical Operation	Index Scan
Logical Operation	Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	16
Estimated Operator Cost	0.0032996 (37%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001746
Estimated Subtree Cost	0.0032996
Number of Executions	1
Estimated Number of Executions	1
Estimated Number of Rows	16
Estimated Row Size	63 B
Ordered	True
Node ID	1

Object
[AdventureWorks2012].[HumanResources].
[Department].[nidx_gn]
Output List
[AdventureWorks2012].[HumanResources].
[Department].DepartmentID, [AdventureWorks2012].
[HumanResources].[Department].GroupName

Key Lookup (Clustered)
Uses a supplied clustering key to lookup on a table that has a clustered index.
Estimated operator progress: 100%

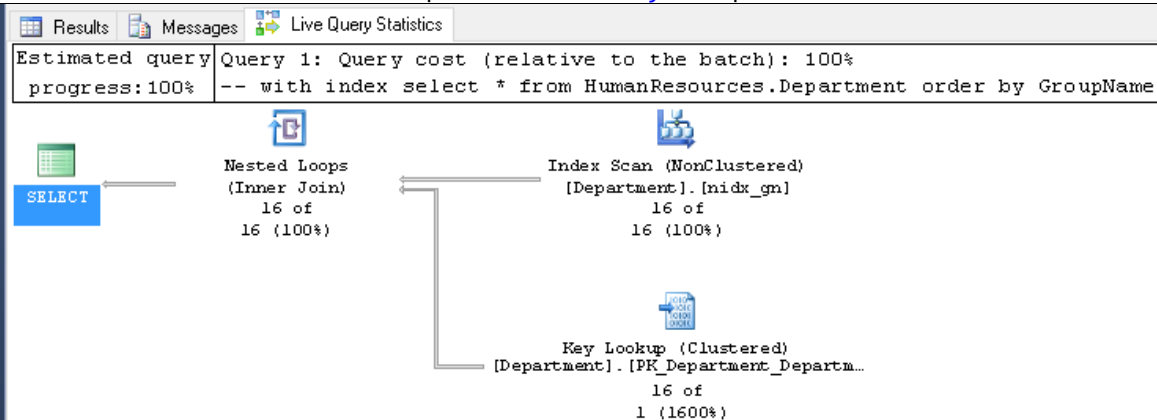|	
Physical Operation	Key Lookup
Logical Operation	Key Lookup
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	16
Estimated Operator Cost	0.0056546 (63%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001581
Estimated Subtree Cost	0.0056546
Number of Executions	16
Estimated Number of Executions	16
Estimated Number of Rows	1
Estimated Row Size	69 B
Ordered	True
Node ID	3

Object
[AdventureWorks2012].[HumanResources].
[Department].[PK_Department_DepartmentID]
Output List
[AdventureWorks2012].[HumanResources].
[Department].Name, [AdventureWorks2012].
[HumanResources].[Department].ModifiedDate
Seek Predicates
Seek Keys[1]: Prefix: [AdventureWorks2012].
[HumanResources].[Department].DepartmentID =
Scalar Operator([AdventureWorks2012].
[HumanResources].[Department].[DepartmentID])

Key Lookup

```
-- Key Lookup example
USE AdventureWorks2012
GO
CREATE NONCLUSTERED INDEX IX_LastName ON Person.Person(LastName)
--Now we can use Ctrl+M to turn on the actual execution plan and run the select.
```

Messages | Live Query Statistics

Estimated query | Query 1: Query cost (relative to the batch): 100%
progress:100% | insert [Person].[Person] select * from [Person].[Person] option (maxdop 1)

T-SQL
INSERT ← Index Insert (NonClustered)
[Person].[IX_LastName]
0 of
19972 (0%)
← Sort
19972 of
19972 (100%)
← Index Scan (NonClustered)
[Person].[IX_Person_LastName_FirstN...
19972 of
19972 (100%)

```
--Now we can use Ctrl+M to turn on the actual execution plan and run the select.
SELECT * FROM Person.Person WHERE LastName = 'Russell'
```

```
Results    Messages    Live Query Statistics    Execution plan
Estimated query    Query 1: Query cost (relative to the batch): 100%
progress:100%    SELECT * FROM Person.Person WHERE LastName = 'Russell'
```

Nested Loops
(Inner Join)
0.737s
126 of
126 (100%)

Index Seek (NonClustered)
[Person].[IX_Person_LastName_FirstN...
0.737s
126 of
126 (100%)

SELECT
0.737s

Key Lookup (Clustered)
[Person].[PK_Person_BusinessEntityI...
0.293s
126 of
1 (12600%)

Index Seek (NonClustered)
Scan a particular range of rows from a nonclustered index.
Estimated operator progress: 100%

Physical Operation	Index Seek
Logical Operation	Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	126
Estimated Operator Cost	0.0034206 (1%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0002956
Estimated Subtree Cost	0.0034206
Estimated Number of Executions	1
Number of Executions	1
Estimated Number of Rows	126
Estimated Row Size	42 B
Ordered	True
Node ID	2

Object
[AdventureWorks2012].[Person].[Person].
[IX_Person_LastName_FirstName_MiddleName]
Output List
[AdventureWorks2012].[Person].
[Person].BusinessEntityID, [AdventureWorks2012].
[Person].[Person].FirstName, [AdventureWorks2012].
[Person].[Person].MiddleName,
[AdventureWorks2012].[Person].[Person].LastName
Seek Predicates
Seek Keys[1]: Prefix: [AdventureWorks2012].[Person].
[Person].LastName = Scalar Operator(N'Russell')

Key Lookup (Clustered)
Uses a supplied clustering key to lookup on a table that has a clustered index.
Estimated operator progress: 100%

Physical Operation	Key Lookup
Logical Operation	Key Lookup
Estimated Execution Mode	Row
Storage	RowStore
Actual Number of Rows	126
Estimated Operator Cost	0.404257 (99%)
Estimated I/O Cost	0.003125
Estimated CPU Cost	0.0001581
Estimated Subtree Cost	0.404257
Number of Executions	126
Estimated Number of Executions	126
Estimated Number of Rows	1
Estimated Row Size	8117 B
Ordered	True
Node ID	4

Object
[AdventureWorks2012].[Person].[Person].
[PK_Person_BusinessEntityID]
Output List
[AdventureWorks2012].[Person].[Person].PersonType,
[AdventureWorks2012].[Person].[Person].NameStyle,
[AdventureWorks2012].[Person].[Person].Title,
[AdventureWorks2012].[Person].[Person].Suffix,
[AdventureWorks2012].[Person].
[Person].EmailPromotion, [AdventureWorks2012].
[Person].[Person].AdditionalContactInfo,
[AdventureWorks2012].[Person].
[Person].Demographics, [AdventureWorks2012].
[Person].[Person].rowguid, [AdventureWorks2012].
[Person].[Person].ModifiedDate
Seek Predicates
Seek Keys[1]: Prefix: [AdventureWorks2012].[Person].
[Person].BusinessEntityID = Scalar Operator
([AdventureWorks2012].[Person].[Person].
[BusinessEntityID])

```
-- Index Seek (NonClustered) - 1% ; Key Lookup (Clustered) - 99%
```

In the execution plan we have an Index Seek that is using the new index, but we also have a Key Lookup on the clustered index. The reason for this is that the nonclustered index only contains the LastName column, but since we are doing a SELECT *, the query has to get the other columns from the clustered index and therefore we have a Key Lookup. The other operator that we have is the Nested Loops; this joins the results from the Index Seek and the Key Lookup.

```
SELECT LastName FROM Person.Person WHERE LastName = 'Russell'
-- only Index Seek (NonClustered) :)
-- we no longer have a Key Lookup and we also no longer have the Nested Loops operator
```

```
-- without Key Lookup :)
-- so, the first statement takes 99% of the batch and the second statement takes 1%, so
this is a big improvement.
```

```
-- if the table has a clustered index we can include the clustered index column(s) as
well without doing a Key Lookup.

SELECT BusinessEntityID, LastName FROM Person.Person WHERE LastName = 'Russell'
-- without Key Lookup :)
```



```
-- if you need to include other columns that are not part of the clustered/nonclustered
index, then the Key Lookup comes back again.
SELECT LastName, EmailPromotion FROM Person.Person WHERE LastName = 'Russell'
```



```
/* Key Lookup appears in the queries in which the fields/columns involved
(in select, join, where, order by, …) are not part of a clustered/non-clustered index.
This can be handled with the covering indexes or by using the INCLUDE clause in create
index statement. */
```

```
-- I. covering index - including all of the columns that are needed
CREATE NONCLUSTERED INDEX IX_Email_LastName ON Person.Person (EmailPromotion, LastName)
GO
SELECT LastName, EmailPromotion FROM Person.Person WHERE LastName = 'Russell'
-- no Key Lookup
```

```
Estimated query | Query 2: Query cost (relative to the batch): 1%
progress:100%  | SELECT LastName, EmailPromotion FROM Person.Person WHERE LastName = 'Russell'
```

```
                        Index Scan (NonClustered)
                        [Person].[IX_Email_LastName]
SELECT                            126 of
                                 126 (100%)
```

```
-- II.  include clause
/* Another option is to use the included columns feature for an index.
This allows you to include additional columns so they are stored with the index,
but are not part of the index tree.
So this allows you to take advantage of the features of a covering index and reduces
storage needs
within the index tree. Another benefit is that you can include additional data types
that can not be part of a covering index. */

CREATE NONCLUSTERED INDEX IX_EmailPromotion_LastName ON Person.Person(LastName)
INCLUDE (EmailPromotion)
GO
SELECT LastName, EmailPromotion FROM Person.Person WHERE LastName = 'Russell'
-- no Key Lookup
```

```
Estimated query | Query 2: Query cost (relative to the batch): 0%
progress:100%  | (@1 varchar(8000))SELECT [LastName],[EmailPromotion] FROM [Person].[Person] WHERE
```

```
                        Index Seek (NonClustered)
                  [Person].[IX_EmailPromotion_LastNam...
SELECT                            0.149s
0.149s                            126 of
                                 126 (100%)
```

References:
https://www.mssqltips.com/sqlservertutorial/277/index-scans-and-table-scans/
https://blog.sqlauthority.com/2007/03/30/sql-server-index-seek-vs-index-scan-table-scan/
https://stackoverflow.com/questions/40486539/index-seek-vs-index-scan-in-sql-server
https://www.mssqltips.com/sqlservertutorial/258/eliminating-bookmark-keyrid-lookups/