Microsoft

SQL Server Index Structure

Module 5

Learning Units covered in this Module

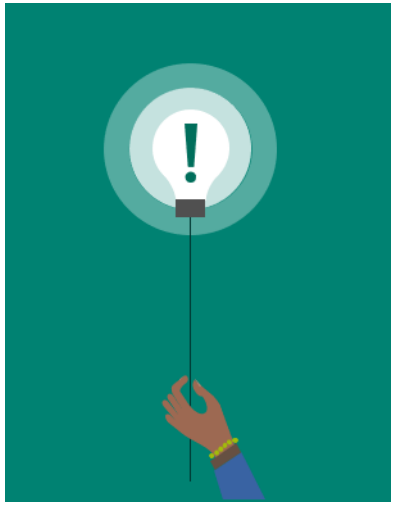
- Lesson 1: Index Internals
- Lesson 2: Index Strategy
- Lesson 3: Partitioned Tables and Indexes
- Lesson 5: Index Monitoring and Fragmentation
- Lesson 4: Columnstore Indexes
- Lesson 6: In-Memory Tables

Lesson 1: Index Internals

Objectives

After completing this learning, you will be able to:

- Understand differences between heap and clustered objects.
- Describe B-Trees, and why they are beneficial.
- Understand the differences between clustered and non-clustered indexes.
- Describe the different methods by which data structures are accessed.



What is an Index?

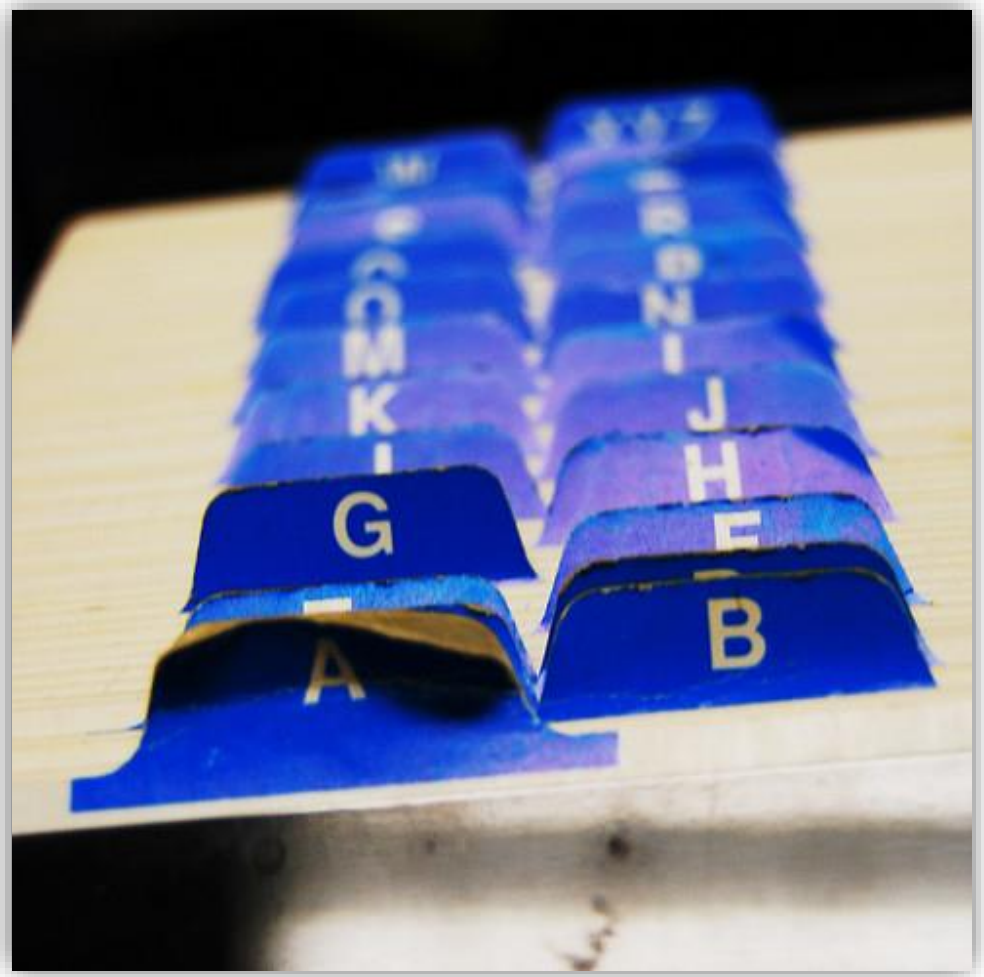
An index is an on-disk structure associated with a table that speeds retrieval of rows.

An index contains keys built from one or more columns in the table.

What is SARGability?

A SARGable item in a search predicate is able to use an index.

Non-SARGable expressions can significantly slow down queries.



How Data is Stored in Data Pages

Data stored in a Heap is not stored in any order and normally does not have a Primary Key.

Heap

AcctID	AcctName	Balance
1	Jack	500.00
2	Diane	750.00
29	Kelli	1250.00
27	Jessica	1005.00
18	Maddison	745.00
22	Bella	445.00

Clustered Index data is stored in sorted order by the Clustering key. In many cases, this is the same value as the Primary Key.

Clustered Index

AcctID	AcctName	Balance
1	Jack	500.00
2	Diane	750.00
12	Danny	630.00
14	Mayleigh	204.00
15	Molly	790.00
18	Maddison	745.00

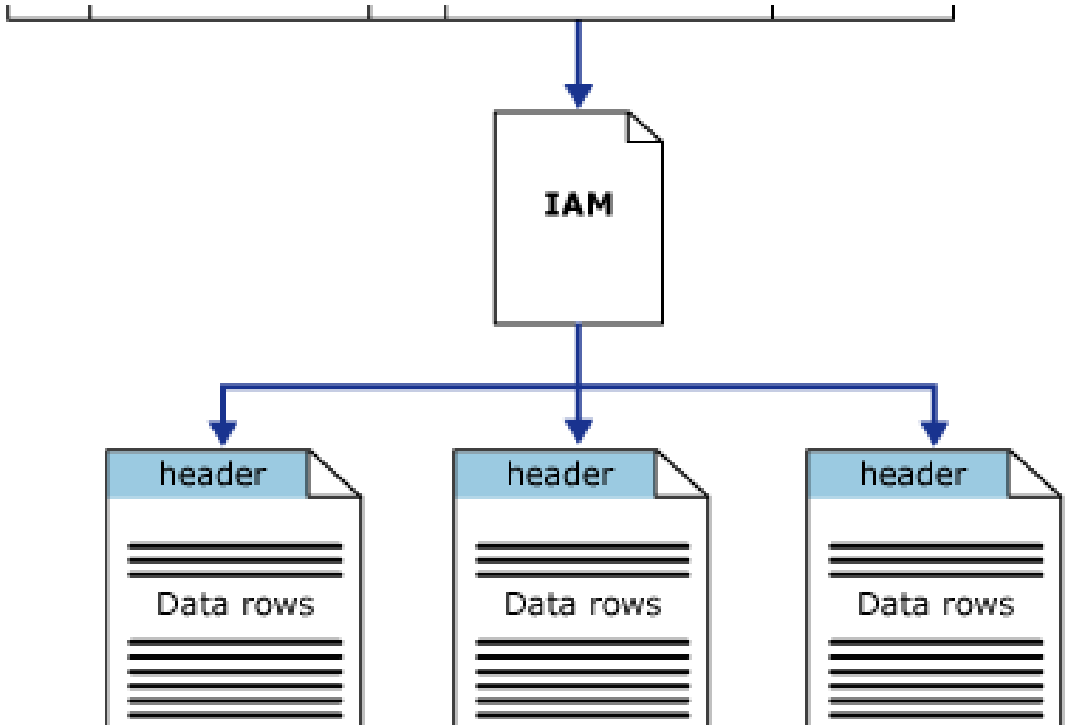
Heap

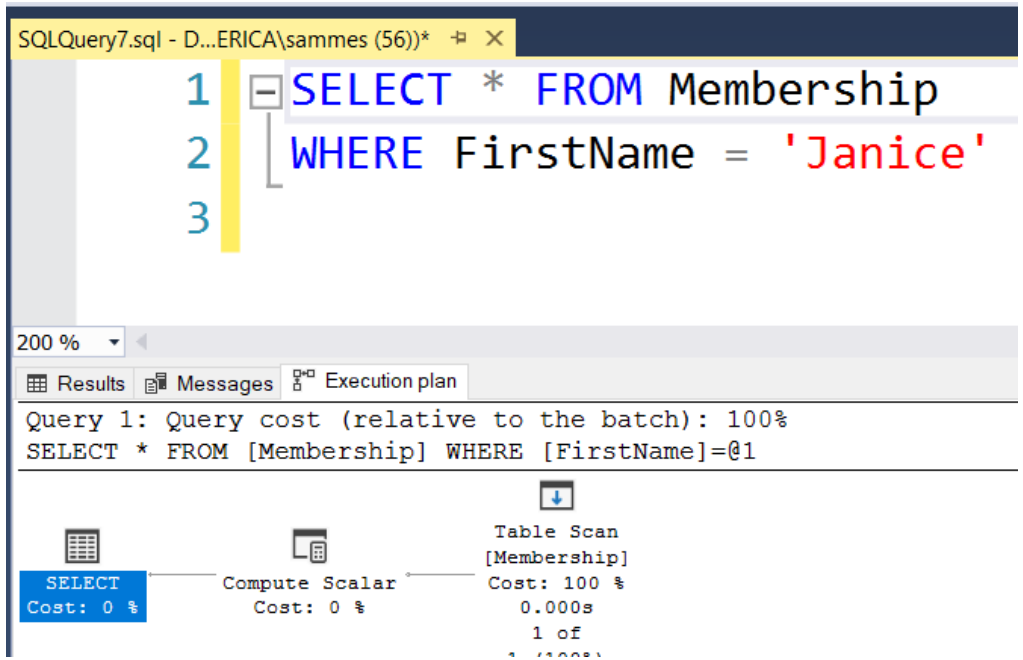
A heap is a table without a clustered index

- Unordered masses of data
- Can be good for quickly importing large sets of data
- Not a good idea for reporting-based data structures
- Use the ALTER TABLE…REBUILD command to “rebuild” a heap table
- Do not use a heap

Heap Structures

Heaps have one row in sys.partitions, with index_id = 0 for each partition used by the heap





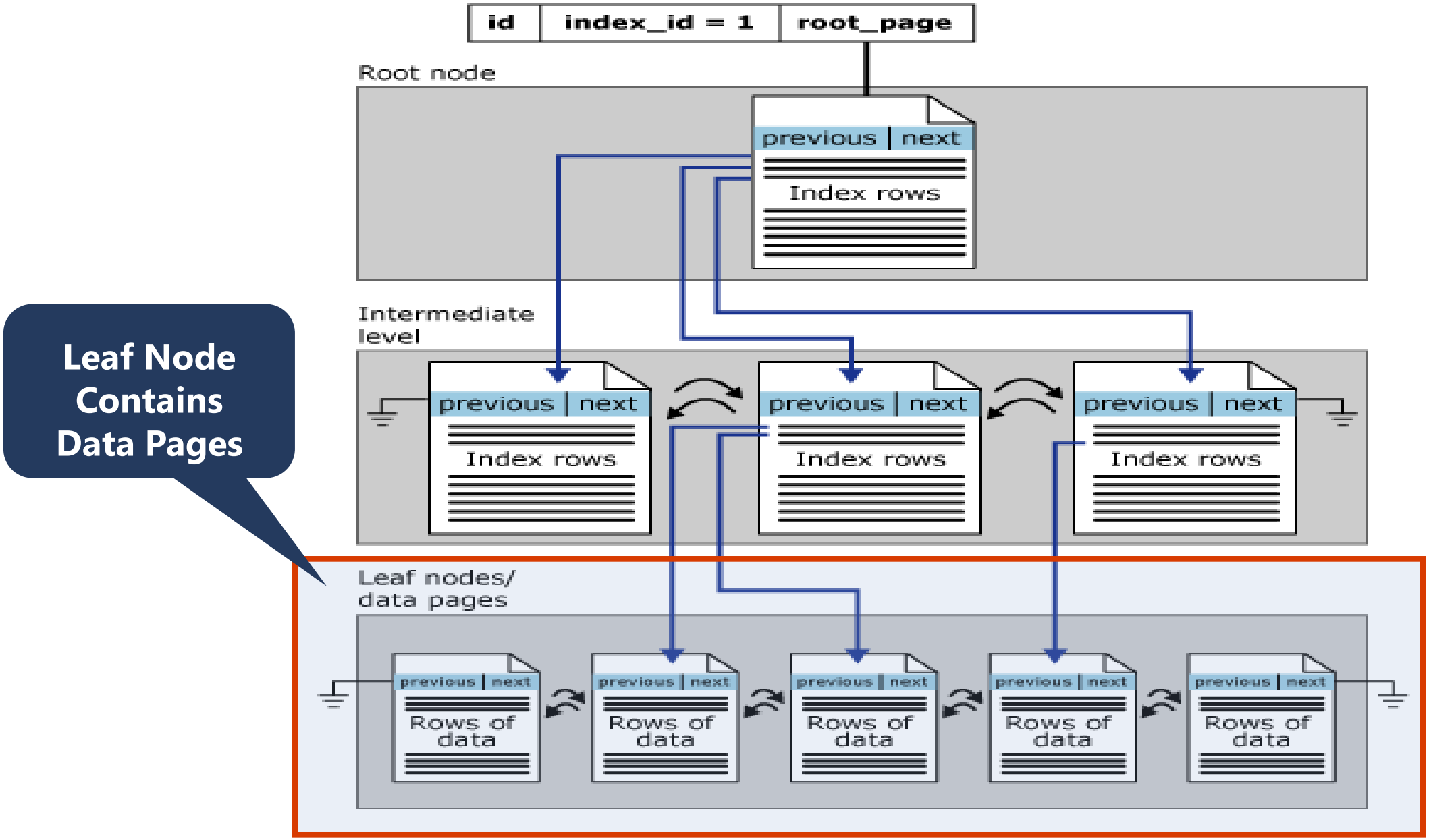
Clustered Indexes

An ordered data structure that is implemented as a Balanced (B) Tree.

When a table is clustered, the leaf level of the index contains ALL data in the table. This means that the clustered index IS the table. This is also why there is only one per table.

The leaf level of the clustered index contains data pages.

Clustered Index Structure



Demonstration

Examining index metadata and Building a B-Tree

- This demonstration reviews index, partition, and allocation unit metadata.
- Show how a B-Tree begins to form as rows are added to a table.

Non-Clustered Indexes

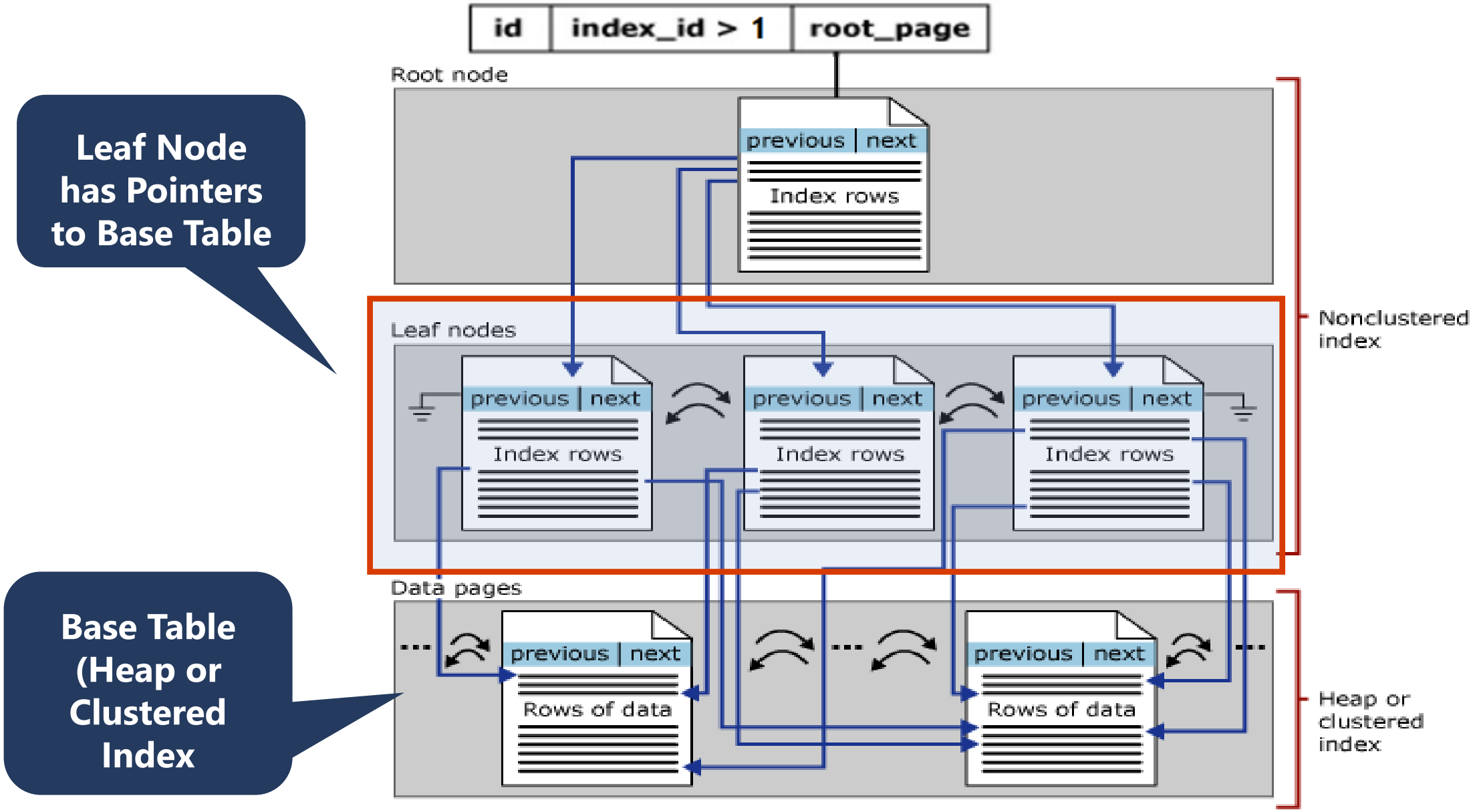
Same B-tree data structure as a clustered index.

It is a separate structure built on top of a Heap or Clustered Index.

Only contains a subset of the columns in the base table

The leaf level contains only the columns defined in the index as well as the clustered key/heap ID that points to the base table structure.

Non-Clustered Index Structure



Clustered vs Nonclustered Indexes

An index is an on-disk structure associated with a table or view that speeds retrieval of rows.

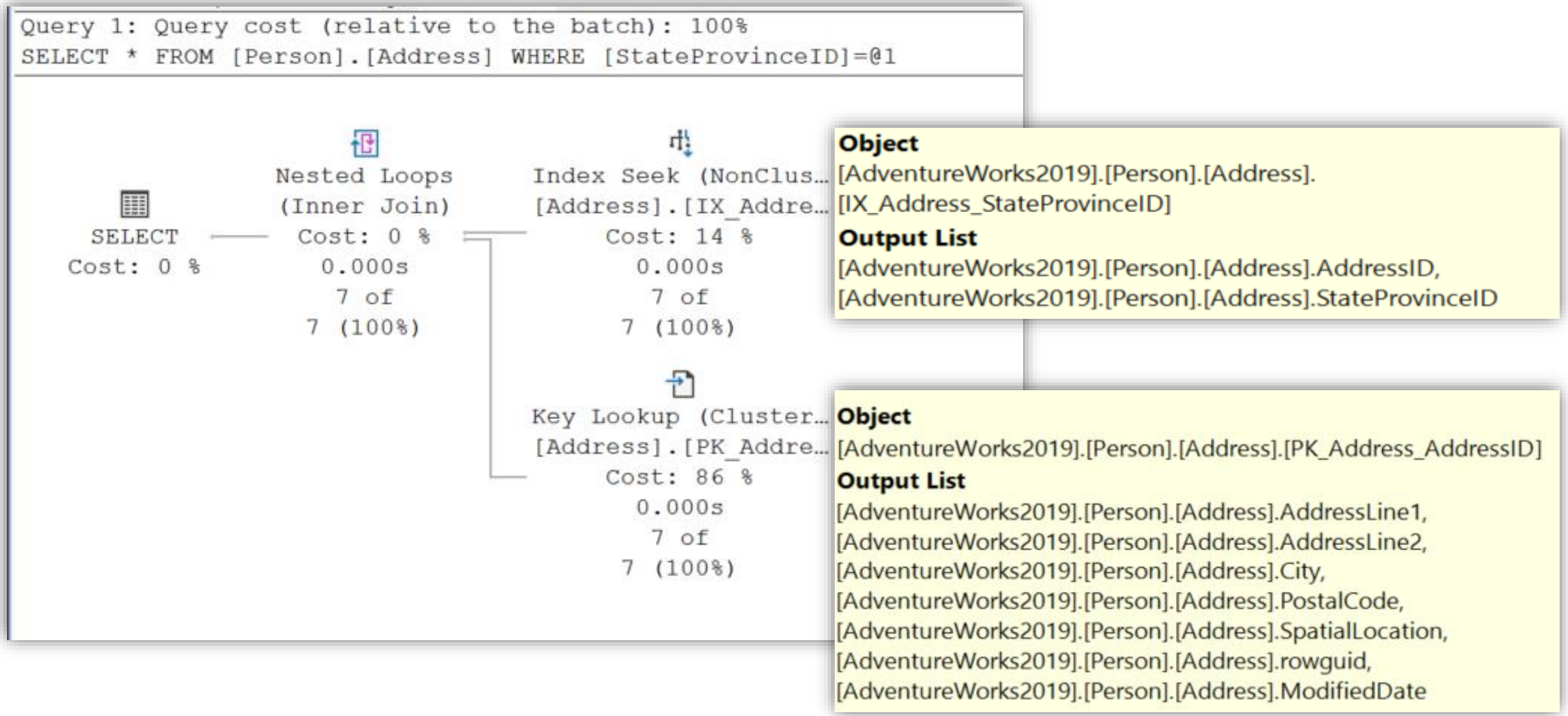
Clustered Indexes

Bookmark Lookup or RID lookup

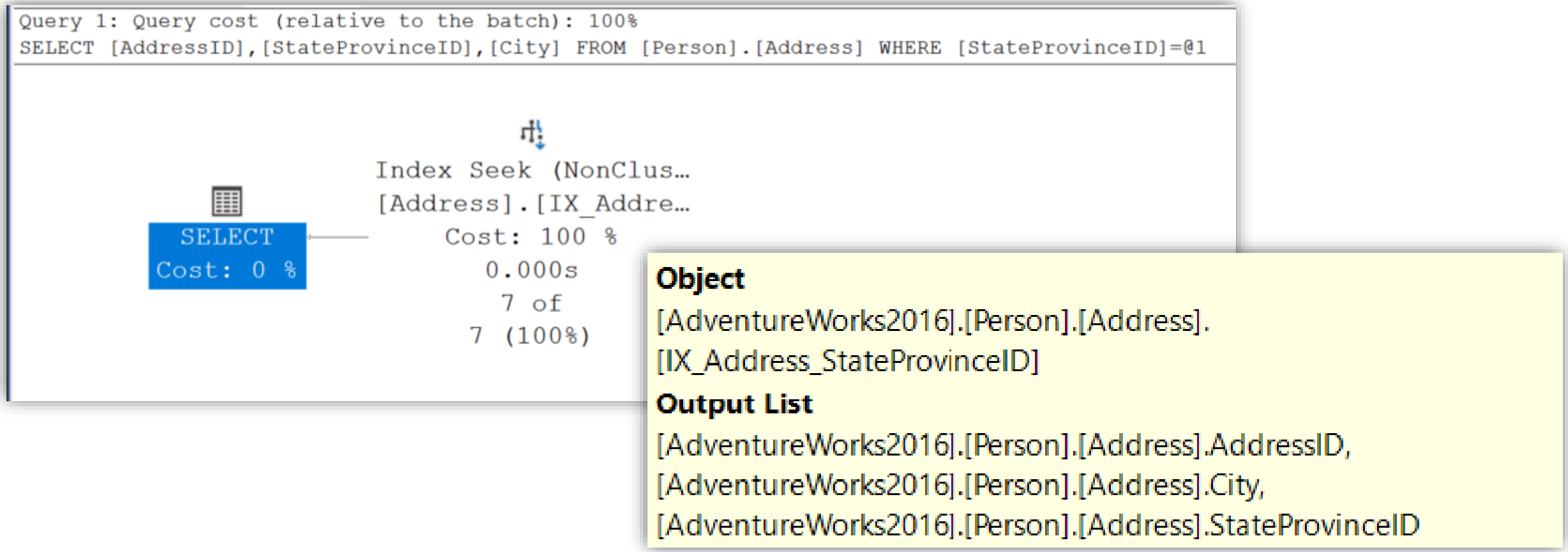
Occurs when SQL uses a nonclustered index to satisfy all or some of a query's predicates, but it doesn't contain all the information needed to cover the query.

Lookup effectively join the nonclustered index back to the clustered index or heap.

Key Lookup


Query 1: Query cost (relative to the batch): 100%
SELECT * FROM [Person].[Address] WHERE [StateProvinceID]=@1
SELECT
Cost: 0 %
Nested Loops
(Inner Join)
Cost: 0 %
0.000s
7 of
7 (100%)
Index Seek (NonClus…
[Address].[IX_Addre…
Cost: 14 %
0.000s
7 of
7 (100%)
Key Lookup (Cluster…
[Address].[PK_Addre…
Cost: 86 %
0.000s
7 of
7 (100%)
Object
[AdventureWorks2019].[Person].[Address].
[IX_Address_StateProvinceID]
Output List
[AdventureWorks2019].[Person].[Address].AddressID,
[AdventureWorks2019].[Person].[Address].StateProvinceID
Object
[AdventureWorks2019].[Person].[Address].[PK_Address_AddressID]
Output List
[AdventureWorks2019].[Person].[Address].AddressLine1,
[AdventureWorks2019].[Person].[Address].AddressLine2,
[AdventureWorks2019].[Person].[Address].City,
[AdventureWorks2019].[Person].[Address].PostalCode,
[AdventureWorks2019].[Person].[Address].SpatialLocation,
[AdventureWorks2019].[Person].[Address].rowguid,
[AdventureWorks2019].[Person].[Address].ModifiedDate

Non-Clustered Index with Included Column



Included columns for non-clustered indexes

Included as additional non-key columns of data in the leaf level.

Allows for covering more queries.

As query optimizer can locate column values in index (leaf level) hence performance gain is achieved.

Not restricted to a Maximum of 32 key columns and index key size of 1,700 bytes.

(n)varchar(max) can be used, but not (n)text or image data types.

Index search is not permitted on non-key columns.

Demonstration

Access Methods

Demonstrate seeks, scans, and lookups

Questions?

Knowledge Check

What are the three access methods reviewed in this lesson?

What are the three different types of allocations?

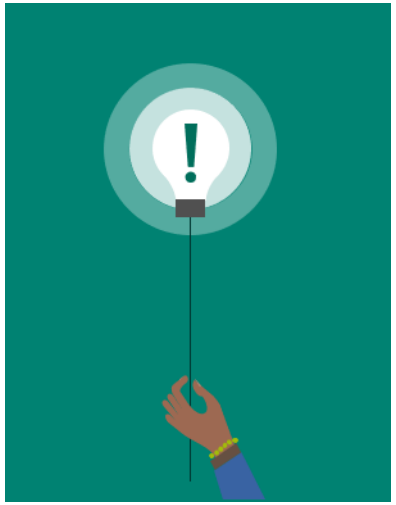
What is the primary goal of indexing?

Lesson 2: Index Strategy

Objectives

After completing this learning, you will be able to:

- Describe best practices associated with clustered index design.
- Consider factors involved in designing a nonclustered indexing strategy.
- Explain how an indexing strategy can reduce logical reads.
- Describe use of Filter indexes and Indexed views.
- Explain the role of Fill factor.
- Use a technique to reduce the fragmentation.
- Apply Index Compression.



Characteristics of a Good Clustering Key

Narrow	Unique	Static	Increasing
• Use a data type with a small number of bytes to conserve space in tables and indexes	• To avoid SQL adding a 4-byte uniquifier	• Allows data to stay constant without constant changes which could lead to page splits	• Allows better write performance and reduces fragmentation issues

Best Practices for Clustered indexes

The primary goal of any indexing strategy is to make queries faster

Best Practice	Seeks, not scans	Avoid key lookups	Get most rows on the index page	Get most rows on the data page	Avoid page splits
Use frequently searched value	X	X			
Use narrow keys			X	X	
Use unique values			X	X	
Use static values					X
Use ascending or descending key					X

Clustered Index vs Primary key considerations

Frequently, clustering on the primary key supports the clustered index best practices

Non-Clustered Index Strategy

Index columns used in WHERE clauses and JOINs.

Keep indexes to a minimum, to minimize impact on DML and log writes.

Two basic approaches: Single column indexes or Multicolumn indexes

Easy to create too many nonclustered indexes.

Specific indexes are appropriate for high-impact queries.

Ideally the focus should be writing queries to use existing indexes, rather than on adding more indexes.

Single Vs Multi-column indexes

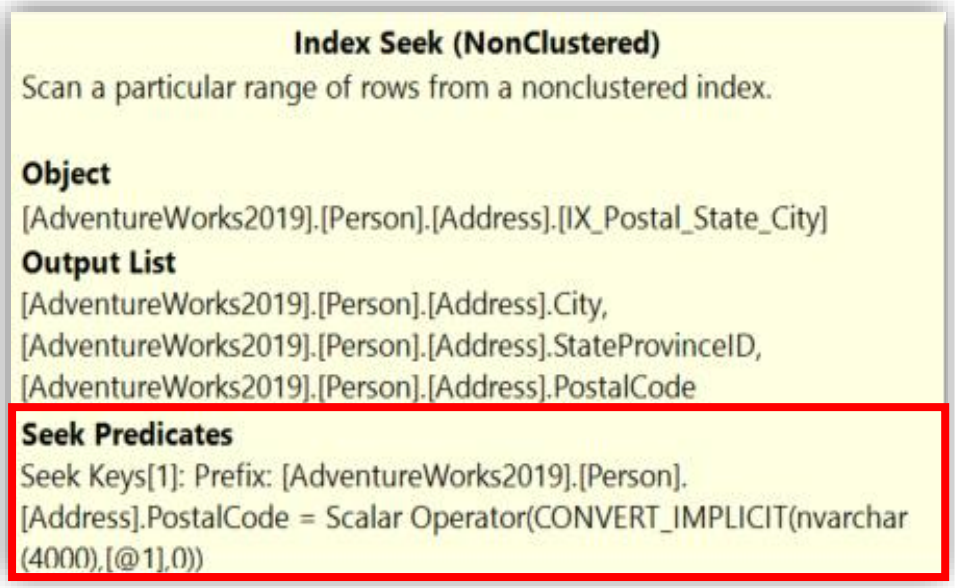
Single-column Indexing

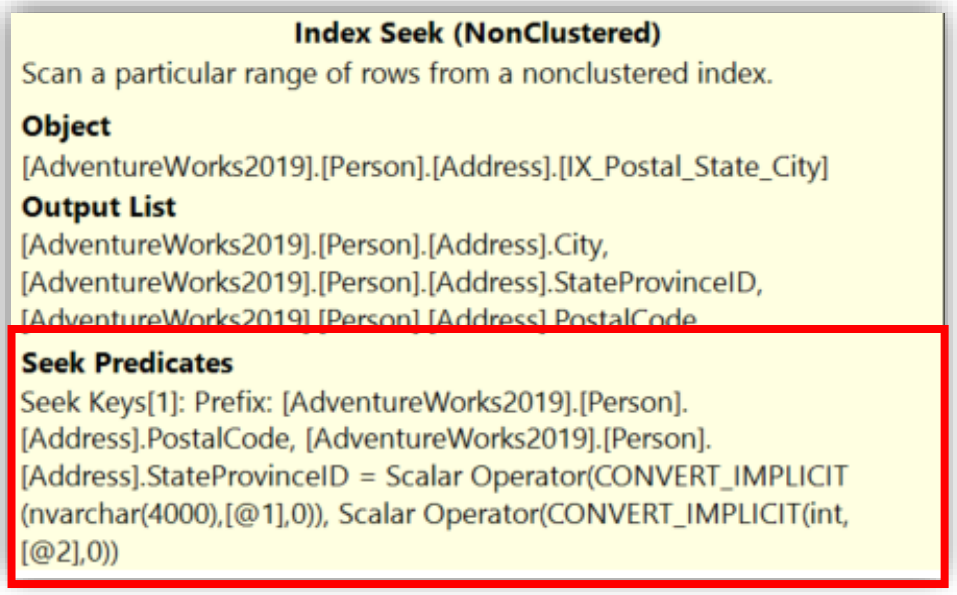
Multi-Column Indexing Access

Multi-Column Indexing Access (Seek Predicates)

```
--Single value performs Index Seek.
SELECT City, StateProvinceID,
PostalCode
FROM Person.Address
WHERE PostalCode = '98011'
```

```
--Index Seek on both columns
--Search condition in same order as Index.
SELECT City, StateProvinceID, PostalCode
FROM Person.Address
WHERE PostalCode = '98011' AND
StateProvinceID = 79
```





Multi-Column Indexing Access (Scan Predicates)

```
--Index Seek on first column
--After seek, will scan second column
SELECT City, StateProvinceID, PostalCode
FROM Person.Address
WHERE PostalCode = '98011' AND City =
'Bothell'
```

```
--Search condition not in same order as
Index.
--Performs Index Scan.
SELECT  City, StateProvinceID, PostalCode
FROM Person.Address
WHERE StateProvinceID = 79
```

Index Seek (NonClustered)

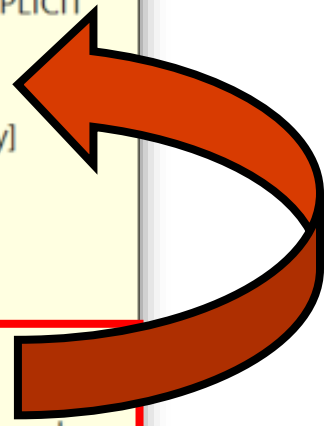
Scan a particular range of rows from a nonclustered index.

Predicate
[AdventureWorks2019].[Person].[Address].[City]=CONVERT_IMPLICIT
(nvarchar(4000),[@2],0)

Object
[AdventureWorks2019].[Person].[Address].[IX_Postal_State_City]

Output List
[AdventureWorks2019].[Person].[Address].City,
[AdventureWorks2019].[Person].[Address].StateProvinceID,
[AdventureWorks2019].[Person].[Address].PostalCode

Seek Predicates
Seek Keys[1]: Prefix: [AdventureWorks2019].[Person].
[Address].PostalCode = Scalar Operator(CONVERT_IMPLICIT(nvarchar
(4000),[@1],0))



Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Predicate
[AdventureWorks2019].[Person].[Address].[StateProvinceID]=(79)

Object
[AdventureWorks2019].[Person].[Address].[IX_Postal_State_City]

Output List
[AdventureWorks2019].[Person].[Address].City,
[AdventureWorks2019].[Person].[Address].StateProvinceID,
[AdventureWorks2019].[Person].[Address].PostalCode

Demonstration

Analyzing Index Usage

Show how index usage patterns can help identify index and query improvements.

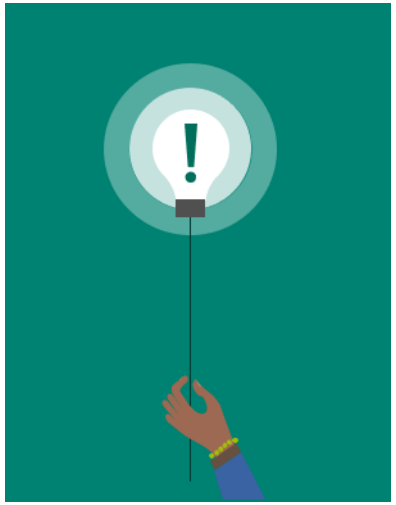
Questions?

Lesson 3: Partitioned Tables and Indexes

Objectives

After completing this learning, you will be able to:

- Understand Partition Concepts.
- Explain benefits of partitioning.
- Distinguish between aligned vs non-aligned indexes.
- Manage partitions.
- Apply Partition performance guidelines.



What is Partitioning?

"Horizontal Partition"

Table or Index Partition

Partitioning benefits

Data Access

Horizontal Partitioning

Table without Partition

ProductKey	OrderDateKey	CustomerKey	SalesOrderNumber	UnitPrice	ExtendedAmount	ProductStandardCost
312	20050922	28191	SO44236	3578.27	3578.27	2171.2942
313	20050923	28163	SO44242	3578.27	3578.27	2171.2942
314	20050926	28142	SO44254	3578.27	3578.27	2171.2942
312	20050927	28171	SO44259	3578.27	3578.27	2171.2942

Table with Partition

Partition Key

Partition 1

ProductKey	OrderDateKey	CustomerKey	SalesOrderNumber	UnitPrice	ExtendedAmount	ProductStandardCost
312	20050922	28191	SO44236	3578.27	3578.27	2171.2942
313	20050923	28163	SO44242	3578.27	3578.27	2171.2942

Partition 2

ProductKey	OrderDateKey	CustomerKey	SalesOrderNumber	UnitPrice	ExtendedAmount	ProductStandardCost
314	20050926	28142	SO44254	3578.27	3578.27	2171.2942
312	20050927	28171	SO44259	3578.27	3578.27	2171.2942

Table Partitioning Concepts

Partition Function

Aligned vs Non-aligned indexes

Aligned Index

Indexes on Partitioned Table

Logical Alignment

- Indexes are logically aligned when they use partitioning column.

Storage Alignment

Partitioned table and index sample

```
-- Creating a partition function with four partitions
CREATE PARTITION FUNCTION myRangePF1 (int)
    AS RANGE LEFT FOR VALUES (1, 100, 1000) ;
GO
-- Creates a partition scheme that applies myRangePF1 to the four filegroups
CREATE PARTITION SCHEME myRangePS1
    AS PARTITION myRangePF1
    TO (test1fg, test2fg, test3fg, test4fg) ;
GO
-- Creates a partitioned table that uses myRangePS1 to partition col1
CREATE TABLE PartitionTable (col1 int NOT NULL, col2 char(10))
    ON myRangePS1 (col1) ;
GO
CREATE CLUSTERED INDEX ClusIndxCol1 ON PartitionTable(col1)
ON myRangePS1(col1)
GO
```

Table Partitioning Operations

Data Movement in table or Partition is done using ALTER …SWITCH/MERGE/SPLIT

SWITCH

Demonstration

Partitioning

Explore Partition Schemes and Functions.

SQL Server Partitioning

- Practicing with a sliding partition window

Questions?

Knowledge Check

What three structures are required to support partitions?

What are some benefits of partitioning?

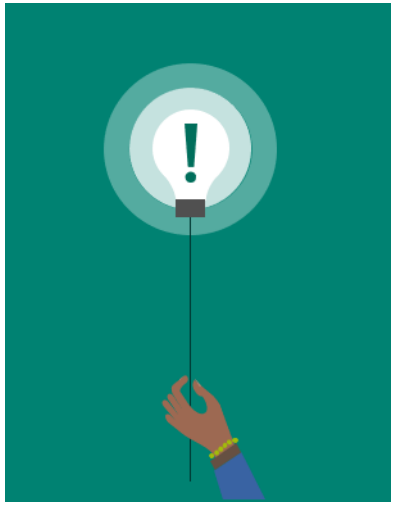
What is used for data movement in partitions or table?

Lesson 4: Index Monitoring and Fragmentation

Objectives

After completing this learning, you will be able to:

- Monitor index usage.
- Describe page splits.
- Understand problems caused by page splits.
- Monitor index fragmentation and free space.
- Review options to remove fragmentation and free space.



Why to monitor Index usage

Data access and distribution changes over time.

By reviewing index usage, we can ensure that the indexes are appropriate for the current workload.

Use sys.dm_db_index_usage_stats and sys.dm_db_index_operational_stats to monitor index usage.

Database standard report also monitors Index usage and Physical Statistics.

Clustered Index usage patterns

Pattern	Interpretation
Seeks, but no scans	• May not need maintenance. Seeks are not impacted by fragmentation
Clustered index with High number of scans	• Review missing index recommendations. • Consider changing Clustering key. • Simplify queries with complex WHERE clauses
Clustered index with Low number of seeks and high number of lookups	• May need to change Clustering Key • Look how often bookmark lookups occur.

Non-Clustered Index usage patterns

Pattern	Interpretation
Seeks, but no scans	• May not need maintenance. • Seeks are not impacted by fragmentation
Non-Clustered index with High number of scans	• Consider compression. • Check for non-SARGable queries. • Review missing index recommendations
Index with No/low usage	• May be able to drop or disable index
Non-Clustered Index Used at specific times only	• May be able to drop index, recreate when needed

sys.dm_db_index_usage_stats

- Tracks seeks, scans, and lookups
- Provides date of last access
- Counters are incremented once per query execution
- User and system access (such as index rebuilds) are tallied separately
- Counters are initialized to at SQL Server (MSSQLSERVER) service is restarts

```
SELECT
        Distinct OBJECT_NAME(Indx.OBJECT_ID) Table_Name  ,Indx.name  IndexName
        ,Indx.type_desc IndexType,Iusg.user_seeks  Seeks,Iusg.user_scans  Scans
        ,Iusg.user_lookups  Lookups ,Iusg.user_updates Updates ,Iusg.last_user_seek LastSeek
        ,Iusg.last_user_scan LastScan,Iusg.last_user_lookup LastLookup ,Iusg.last_user_update LastUpdate
FROM
        SYS.INDEXES Indx
        INNER JOIN SYS.DM_DB_INDEX_USAGE_STATS Iusg
                ON Iusg.index_id = Indx.index_id AND Iusg.OBJECT_ID = Indx.OBJECT_ID
        INNER JOIN SYS.DM_DB_PARTITION_STATS PrtSts
                ON PrtSts.object_id=Indx.object_id
WHERE
        OBJECTPROPERTY(Indx.OBJECT_ID,'IsUserTable') = 1
```

sys.dm_db_index_operational_stats

- Returns information about the lower-level I/O activities
- Takes database_id, the object_id, the index_id and the partition_number as parameters
- Memory-optimized indexes do not appear in this DMV

```
SELECT
        *
FROM
        SYS.DM_DB_INDEX_OPERATIONAL_STATS (NULL,NULL,NULL,NULL ) opsts
        INNER JOIN SYS.INDEXES AS Indx
                ON Indx.[OBJECT_ID] = opsts.[OBJECT_ID]
                AND Indx.INDEX_ID = opsts.INDEX_ID
WHERE
        OBJECTPROPERTY(opsts.[OBJECT_ID],'IsUserTable') = 1
```

sys.dm_db_index_physical_stats

- Returns size and fragmentation information for the data and indexes of the specified table or view in SQL Server.
- Takes database_id, the object_id, the index_id and the partition_number as parameters
- Memory-optimized indexes do not appear in this DMV

```
SELECT OBJECT_SCHEMA_NAME(I.object_id) AS SchemaName, OBJECT_NAME(I.object_id) AS
TableName, I.name, I.Index_ID, IPS.partition_number, IPS.avg_fragmentation_in_percent,
   IPS.page_count, IPS.avg_page_space_used_in_percent
FROM sys.indexes as I
INNER JOIN sys.dm_db_index_physical_stats(DB_ID(), NULL, NULL, NULL, 'Limited') as IPS
ON I.object_id = IPS. object_id
AND I.index_id = IPS.index_id
--WHERE IPS.avg_fragmentation_in_percent >30
--AND IPS.page_count >1000
```

Fragmentation

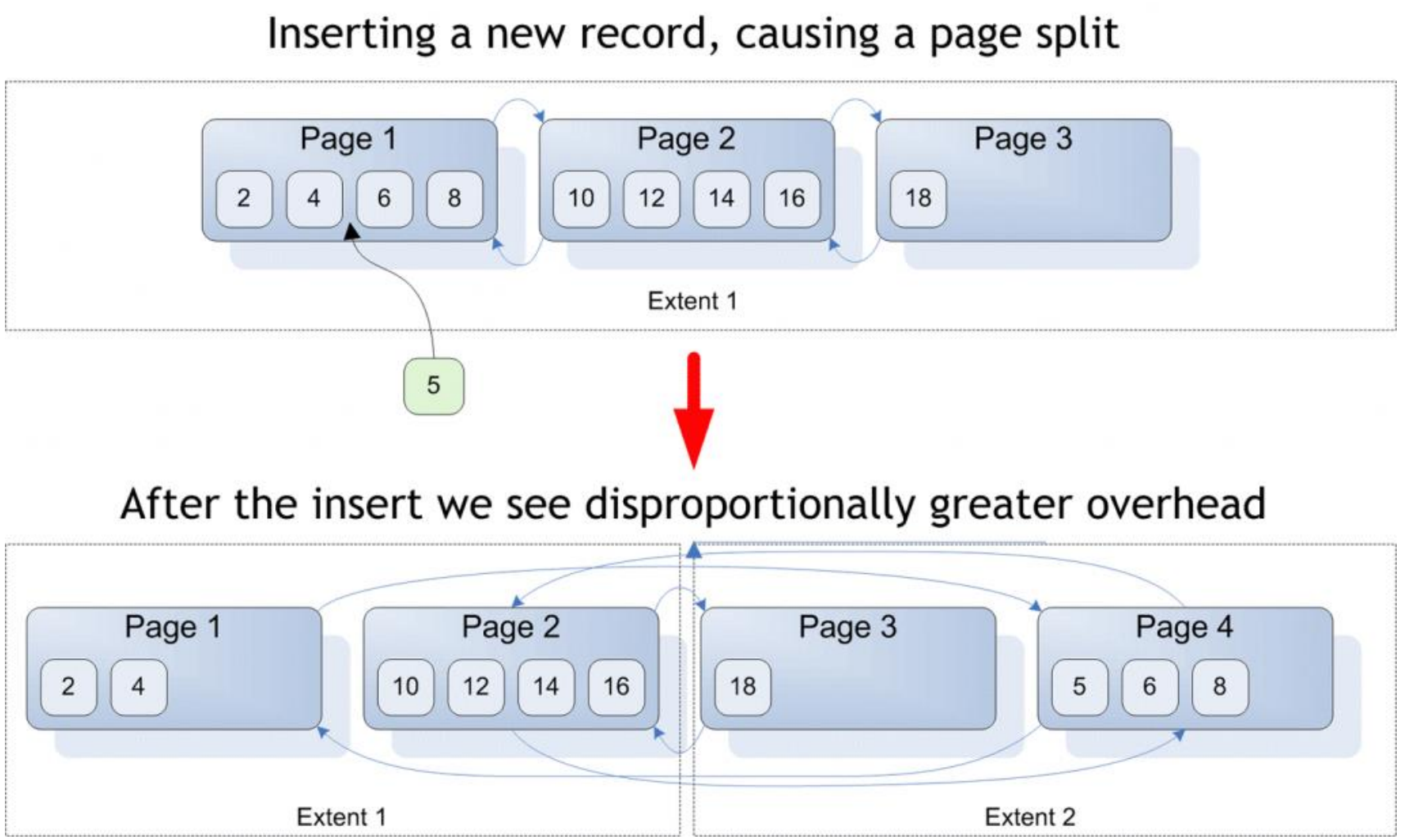
A fragmented table/Index is when some of its data pages point to pages that are not in sequence.

Logical fragmentation

Page splits

- Mechanism to optimize inserts and updates
- Occurs when page is full to hold a new or updated row
- Half the rows are moved to a new page
- The new page is added after the last existing page

Page Splits


Inserting a new record, causing a page split
Page 1
2
4
6
8
Page 2
10
12
14
16
Page 3
18
Extent 1
5
After the insert we see disproportionally greater overhead
Page 1
2
4
Page 2
10
12
14
16
Page 3
18
Page 4
5
6
8
Extent 1
Extent 2

SQL Index Fragmentation and Maintenance options

SSMS -> Index Physical Statistics report

- Will report fragmentation and recommend solution at database level

Custom Solution

Fill Factor

Can address performance issues due to fragmentation.

Use When

Reducing fragmentation

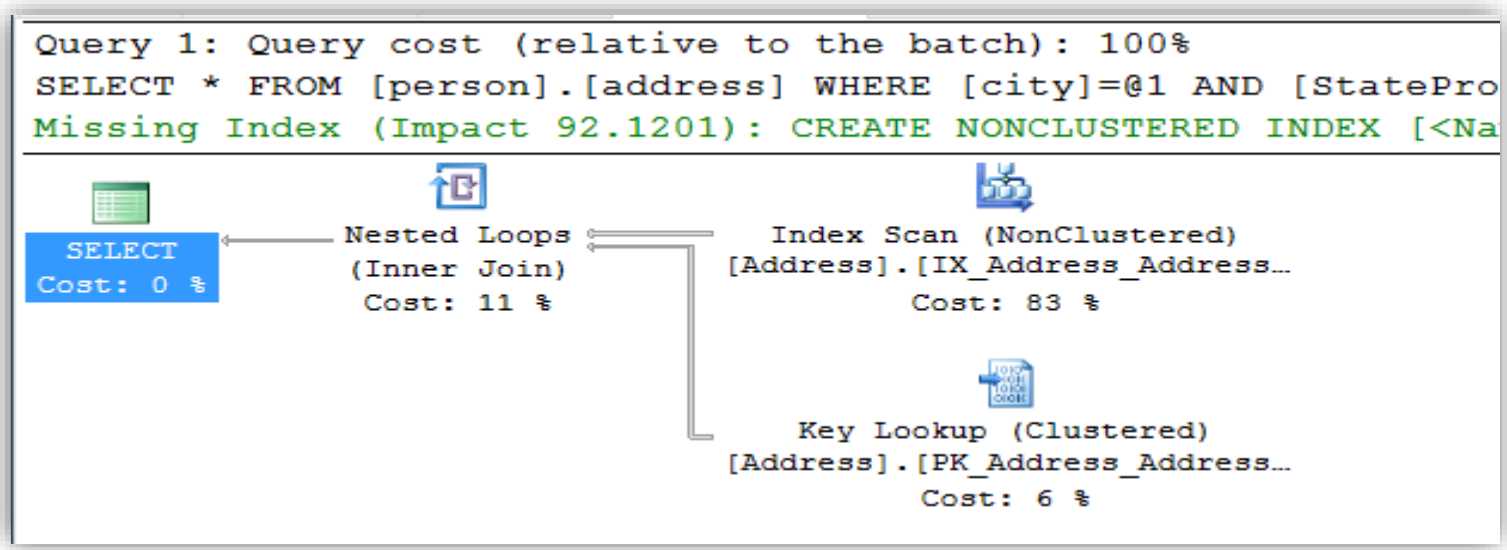
Reducing Fragmentation

Missing Index Recommendations

SQL Server identifies indexes that would help a query's performance

The recommendation is included in the query plan

The cost savings are aggregated in DMVs to help identify the most beneficial indexes



Choosing Indexes

Following sources can help you choosing indexes

Demonstration

Analyzing Index Usage

Show how index usage patterns can help identify index and query improvements.

Index Maintenance

- Identifying and removing physical index fragmentation

Questions?

Knowledge Check

Which DMV/DMF are used to monitor index usage?

Does it make sense to add all SQL Server missing recommendations?

When Managed Lock Priority should be used?