

The background of the slide features a collage of business-related images. On the left, there is a large, glowing, cylindrical data storage icon. In the center, a hand in a suit is drawing a lightbulb and a bar chart on a transparent surface. To the right, there are various icons including a lightbulb, a bar chart, a pie chart, a puzzle piece, and a globe. The overall color scheme is a warm, orange-brown hue.

Intelligent Data Management with SQL Server

Session: 11

Indexes

Objectives

- Define and explain Indexes
- Explain Storage Structure
- Explain types of Indexes
- Understand Index Management

Introduction 1-2

- Indexes are special data structures associated with tables or views that help speed up the query.

Index Type	Description
Clustered	It sorts and stores the data rows of a table or view in order based on the clustered index key. Clustered index is implemented as a B-tree index structure that supports fast retrieval of the rows, based on their clustered index key values.
Nonclustered	Non-clustered index is defined on a table or view that has data in either a clustered structure or on a heap. Each index row in the non-clustered index contains nonclustered key value and a row locator. Locator points to data row in the clustered index or heap having key value. Rows in index are stored in order of the index key values, but the data rows are not guaranteed to be in any particular order unless a clustered index is created on the table.
Unique	Unique index ensures that index key contains no duplicate values and therefore, each row in the table or view is in some way unique. Uniqueness can be a property of both clustered and nonclustered indexes.
Columnstore	Columnstore index stores and manages data by using column-based data storage and column-based query processing in in-memory. Columnstore indexes work well for data warehousing workloads that primarily perform bulk loads and read-only queries. Use the columnstore index to achieve up to 10x query performance gains over traditional row-oriented storage, and up to 7x data compression over the uncompressed data size.
Filtered	Optimized non-clustered index is suited to cover queries that select from a well-defined subset of data. It uses a filter predicate to index a portion of rows in a table. A well-designed filtered index can improve query performance, reduce index maintenance costs, and reduce index storage costs compared with full-table indexes.
Spatial	It provides the ability to perform certain operations more efficiently on spatial objects in a column of geometry data type.
XML	Due to large size of XML columns, queries that search within these columns can be slow. You can speed up these queries by creating an XML index on each column. An XML index can be a clustered or a nonclustered index.

Introduction 2-2

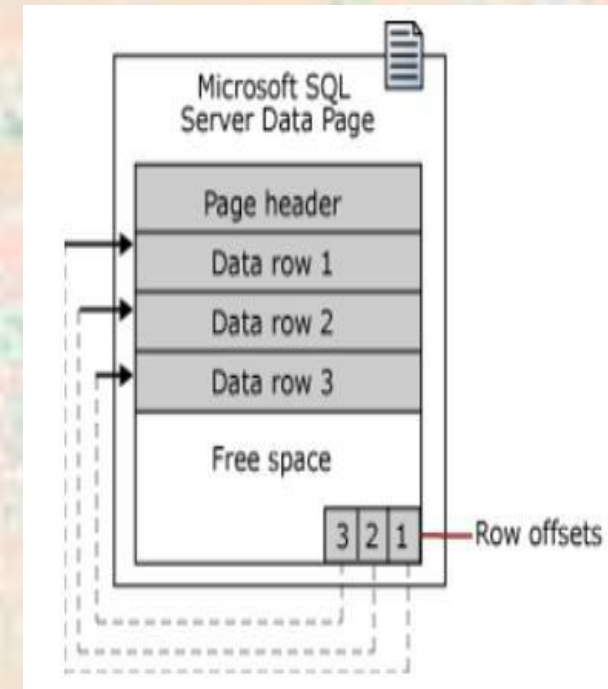
- There are other types of indexes such as Hash, Memory optimize nonclustered, Index with included column, Index on computed columns, and Full text.
- A table scan is not always troublesome, but it is sometimes unavoidable.

Overview of Data Storage

- SQL Server stores data in storage units known as data pages. These pages contain data in the form of rows.
- A page begins with a 96-byte header, which stores system information about the page.

This information includes the following:

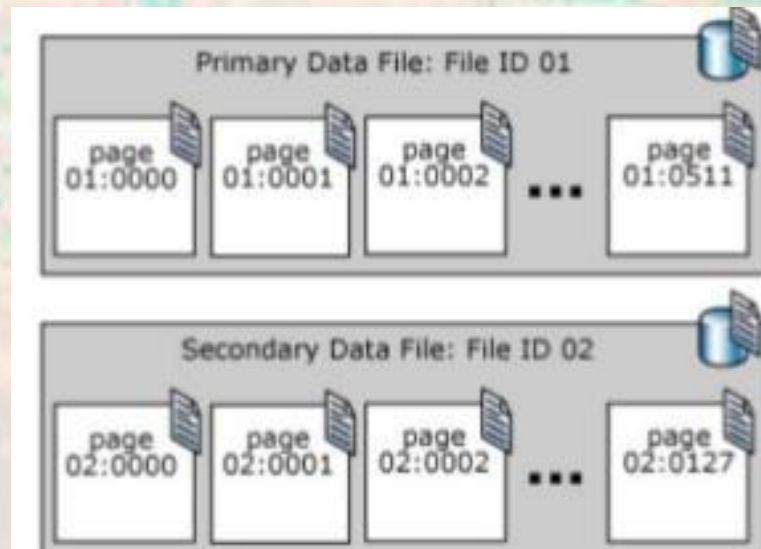
- Page number
- Page type
- Amount of free space on the page
- Allocation unit ID of the object to which the page is allocated



Data Storage

Data Files 1-2

- All input and output operations in the database are performed at the page level.
- SQL Server stores data pages in files known as data files. The space allotted to a data file is divided into sequentially numbered data pages.



Data Files

Data Files 2-2

There are three different types of data files:

Primary

- A primary data file is automatically created at the time of creation of the database. This file has references to all other files in the database. The recommended file extension for primary data files is **.mdf**.

Secondary

- These are optional user-defined data files. Data can be spread across multiple disks by putting each file on a different disk drive. Recommended file name extension for secondary data files is **.ndf**.

Transaction Log

- Log files contain information about modifications carried out in the database. This information is useful in recovery of data in contingencies such as sudden power failure or the need to shift the database to a different server. There is at least one log file for each database. The recommended file extension for log files is **.ldf**.

Requirement for Indexes

- To facilitate quick retrieval of data from a database, SQL Server provides the indexing feature.
- An index in SQL Server database contains information that allows to find specific data without scanning through the entire table.

Index			
A			
Adapter	1	Border	19
Aggregate	10	Bullet	58
Analysis	13		
Average	23	C	
B		Consistency	20
		Connect	22
Board	17	Communication	24
Brilliant	18	Character	30

Index in a Book

Indexes

Records in a table are stored in the order in which they are inserted, this storage is unsorted.

When data is to be retrieved from such tables, the entire table requires to be scanned thus, slowing down the retrieval process.

To speed up data retrieval process, indexes are required.

Index	Employee_Details		
EmployeeID	EmployeeID	EmployeeName	DepartmentID
CN00012	CN00016	John Keena	Purchase
CN00015	CN00015	Smith Jones	Accounts
CN00016	CN00020	Albert Walker	Sales
CN00020	CN00012	Rosa Stines	Administrator

Scenario

- In a telephone directory, where large amount of data is stored and is frequently accessed, the storage is done in alphabetical order. If such data were unsorted, it would be nearly impossible to search for a specific telephone number.
- Similarly, in a database table having a large number of records that are frequently accessed, the data is to be sorted for fast retrieval.
- When an index is created on the table, the index either physically or logically sorts the records.

Accessing Data Group-wise

- Indexes are useful when data is accessed group-wise.

For example, you make modifications to the conveyance allowance for all employees based on the department they work in.

Department Name	Employee Name
Marketing	Jenny Woods
Marketing	Merry Thomas
Marketing	John Updeeke
Marketing	Robert Williamson
Sales	Smith Gordon
Sales	Albert Wang

Accessing Data Group-wise

Index Architecture

In SQL Server, data can be stored either in a sorted or random manner.

If it is stored in a sorted manner then, data is present in clustered structure.

If stored at random then, its present in a heap structure.

Employee_Details		
EmpID	EmpName	DeptID
CN00020	Rosa Stevens	BN0001
CN00018	John Updeeke	BN0020
CN00019	Smith Gordon	BN0021
CN00012	Robert Tyson	BN0011

Heap Structure

Employee_Details		
EmpID	EmpName	DeptID
CN00012	Robert Tyson	BN0011
CN00018	John Updeeke	BN0020
CN00019	Smith Gordon	BN0021
CN00020	Rosa Stevens	BN0001

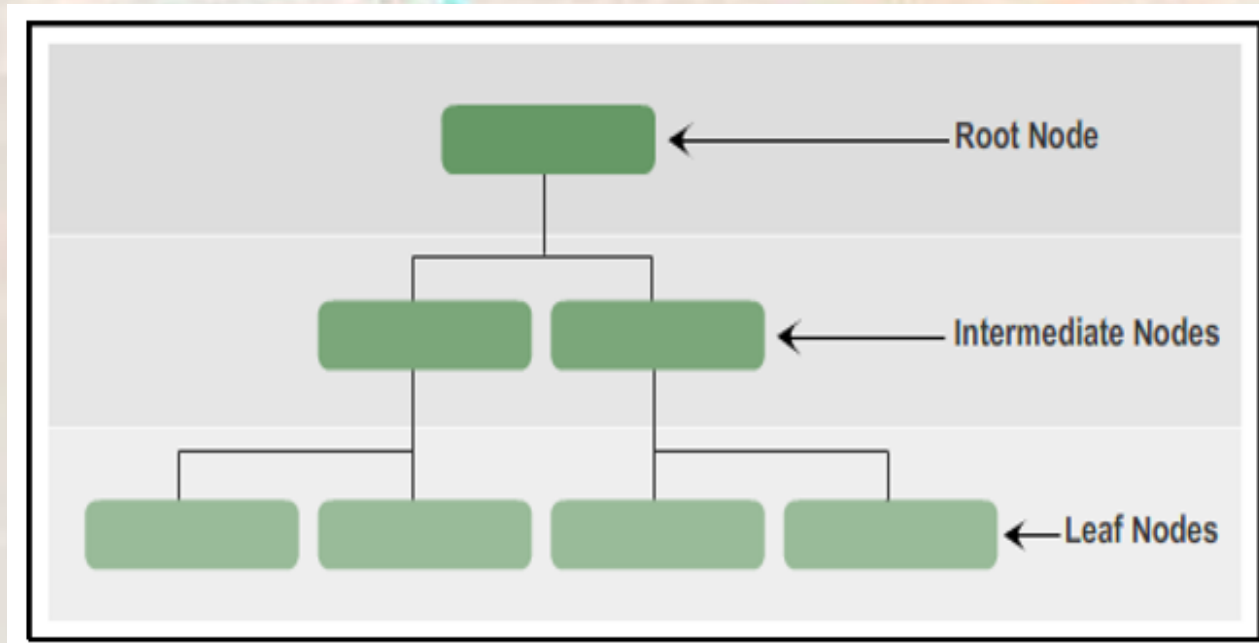
Clustered Structure

Index Architecture

B-Tree

Each page in an index B-tree is called an index node.

Top node of the B-tree is called the root node, whereas the Bottom nodes are called the leaf nodes.



B-Tree

Index B-Tree Structure 1-2

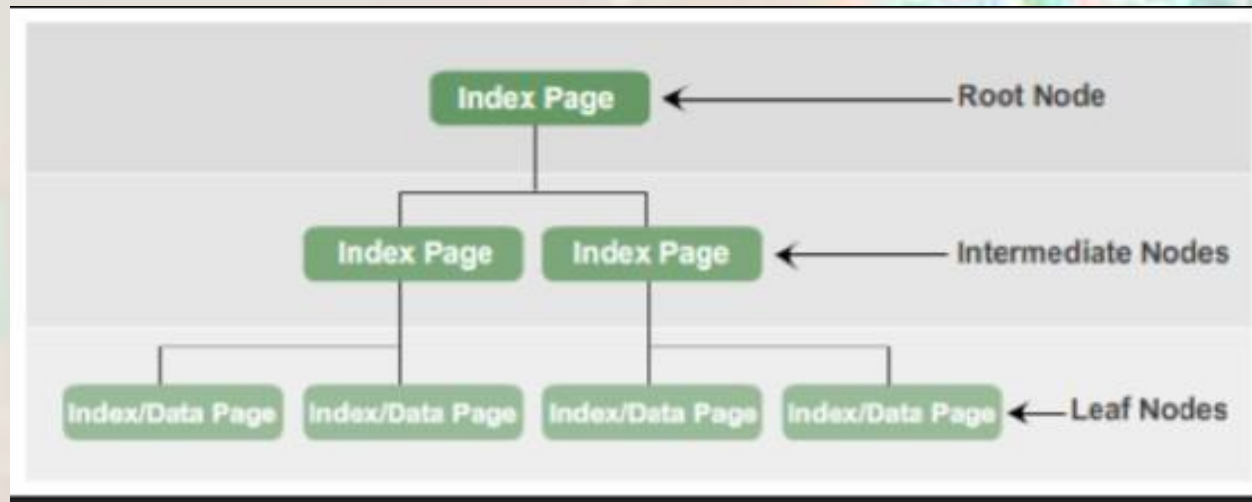
In the B-Tree structure of an index, the root node consists of an index page.

This index page contains pointers present in the first intermediate level.

There can be multiple intermediate levels in an index B-Tree.

The leaf nodes either data pages containing data rows or index pages containing index rows that point to data rows.

Index B-Tree Structure 2-2



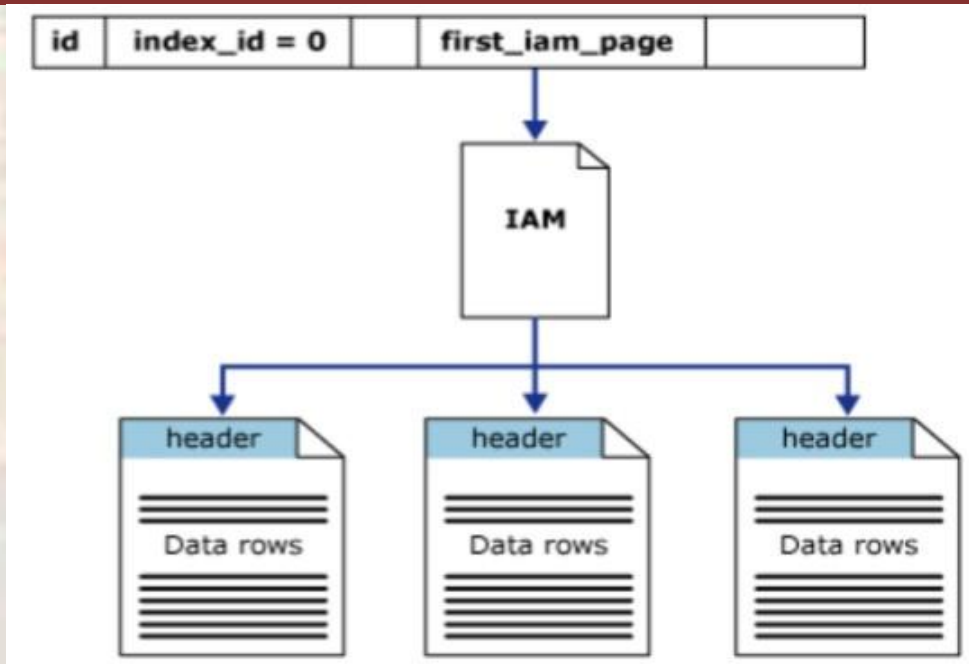
Index B-Tree Structure

Different types of nodes are as follows:

- Root Node - Contains an index page with pointers pointing to index pages at the first intermediate level.
- Intermediate Nodes - Contain index pages with pointers pointing either to index pages at the next intermediate level or to index or data pages at the leaf level.
- Leaf Nodes - Contain either data pages or index pages that point to data pages.

Heap Structures

- A heap is a table without a clustered index.
- In a heap structure, the data pages and records are not arranged in sorted order.
- The only connection is the information recorded in the Index Allocation Map (IAM) pages.



Heap Structures

Clustered Index Structures 1-2

Clustered indexes are organized as B-Trees. Each page in an index B-tree is called an index node.

Clustered Index

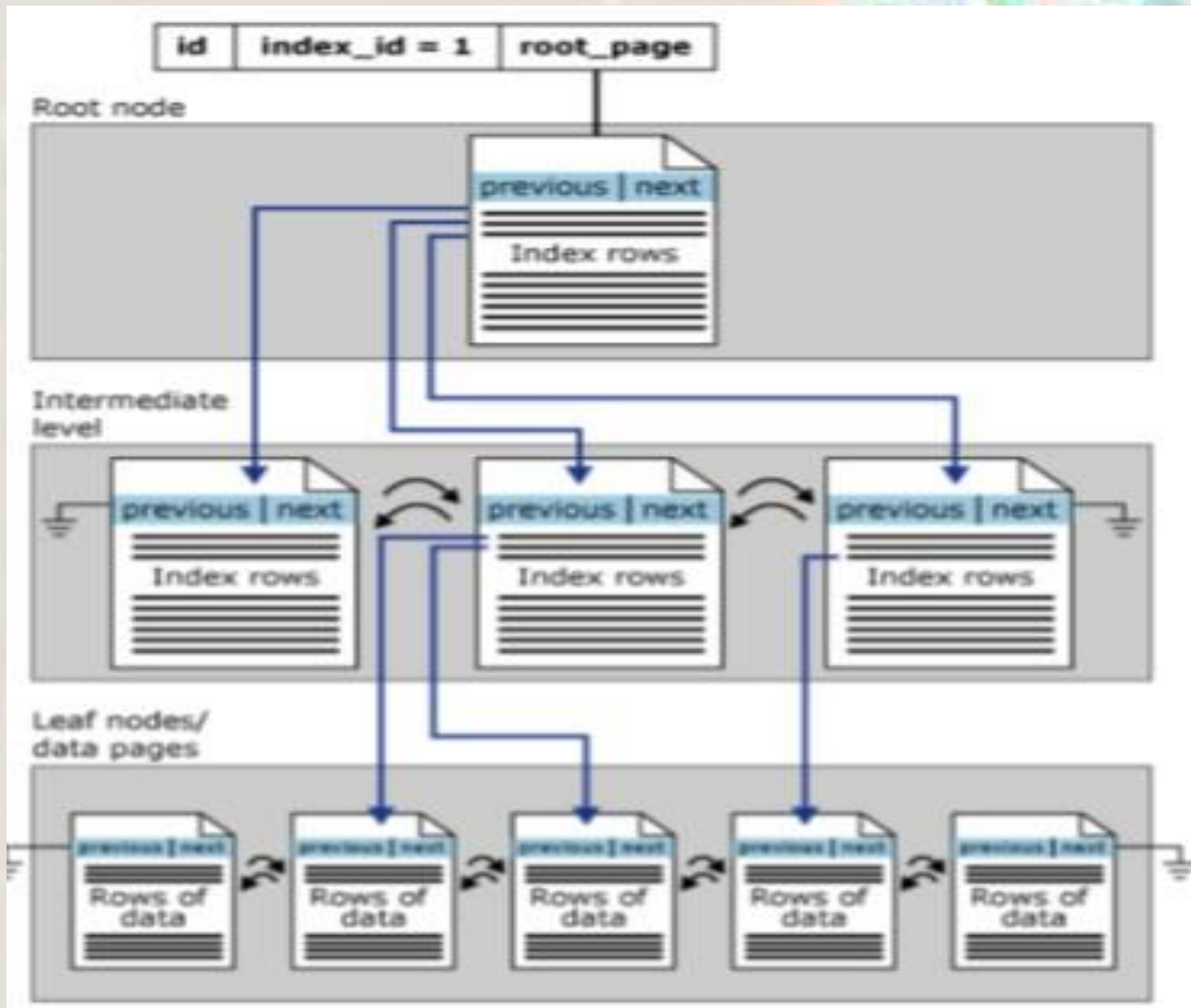
Leaf nodes contain data pages of the underlying table, root, and intermediate level nodes contain index pages holding index rows.

Each index row contains a key value and a pointer to either an intermediate level page in the B-tree or a data row in the leaf level of the index.

By default, a clustered index has a single partition. When a clustered index has multiple partitions, each partition has a B-tree structure that contains the data for that specific partition.

The clustered index will also have one *LOB_DATA* allocation unit per partition if it contains large object (LOB) columns. It will also have one *ROW_OVERFLOW_DATA* allocation unit per partition if it contains variable length columns that exceed the 8,060 byte row size limit.

Clustered Index Structures 2-2



Clustered Indexes

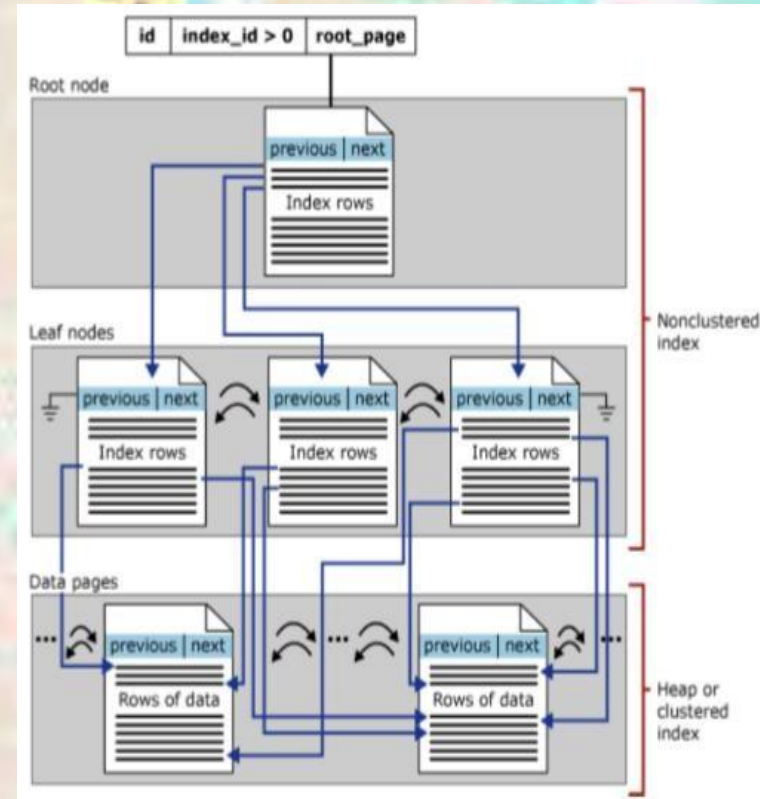
Nonclustered Index Structures

➤ A nonclustered index is defined on a table that has data in either a clustered structure or a heap.

➤ Each index row in the nonclustered index contains a nonclustered key value and a row locator.

Nonclustered indexes have a similar B-Tree structure as clustered indexes, but with the following differences:

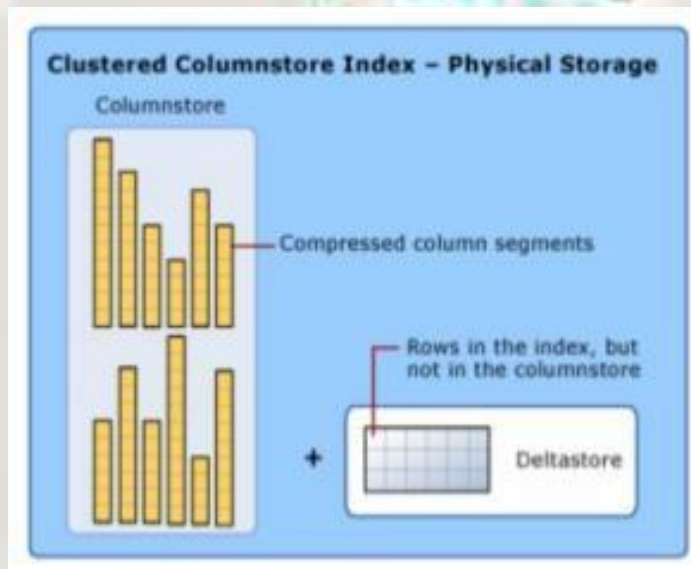
- In a nonclustered index structure, the leaf level contains index rows.
- The data rows of the table are not physically stored in the order defined by their nonclustered keys.



Nonclustered Index

Column Store Index

- A column store index is a feature in SQL Server for storing, retrieving, and managing data by using a columnar data format



Clustered Column Store Index

Columnstore

It is logically organized data as a table with rows and columns and physically stored in a column-wise data format.

Rowstore

It is logically organized data as a table with rows and columns and then, physically stored in a row-wise data format.

Deltastore

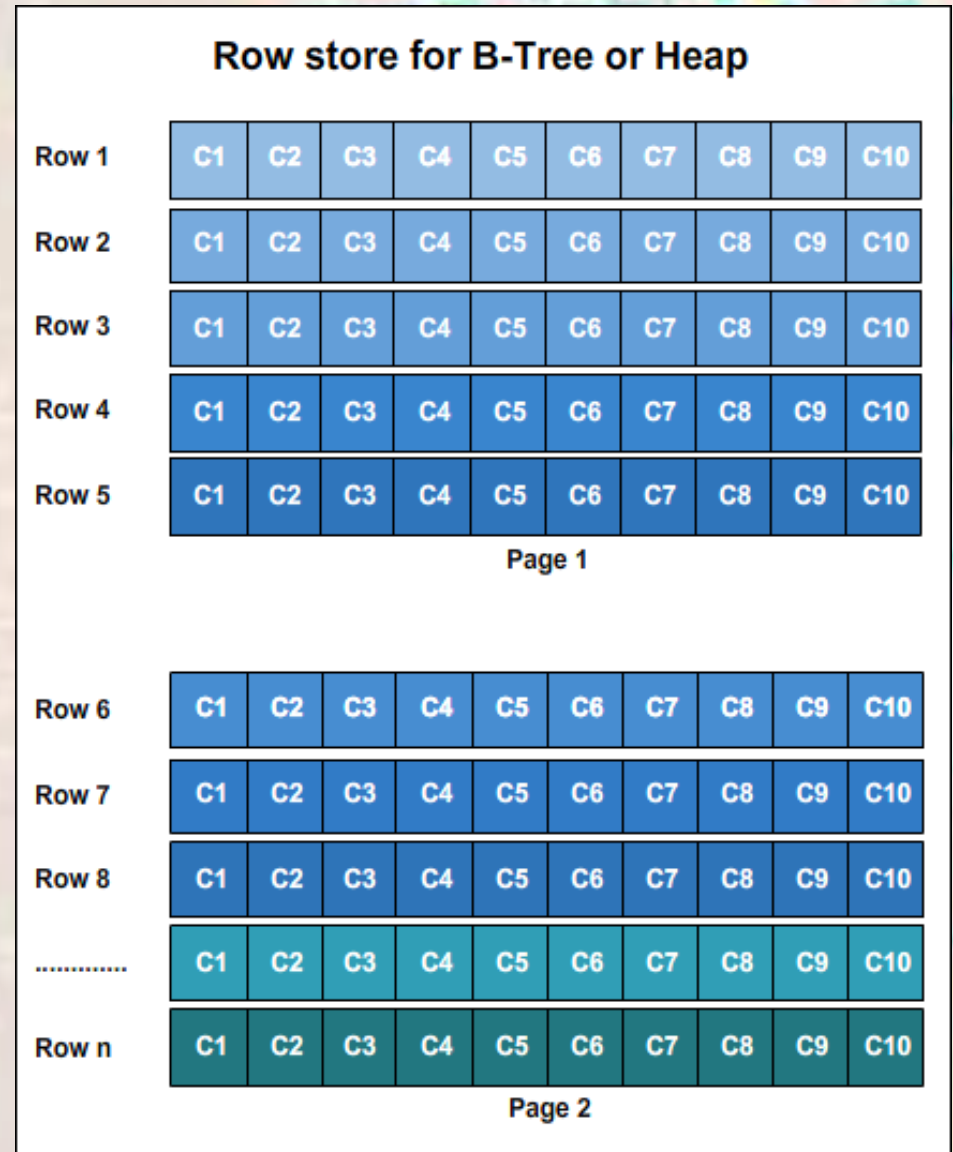
It is a holding place for rows that are too few in number to be compressed into the columnstore. The deltastore stores the rows in rowstore format.

Columnstore indexes are mainly used for following reasons:

- To reduce storage costs
- Better performance

Column Store Index 2-4

- For example, if there is a table with ten columns (C1 to C10), the data of all the ten columns from each row gets stored together continuously on the same page as shown in the aside figure.



Column Store Index 3-4

- When column store index is created, the data is stored column-wise, which means data of each individual column from each row is stored together on same page.
- For example, the data of column C1 of all the rows gets stored together on one page and the data for column C2 of all the rows gets stored on another page and so on as shown in the aside figure

Column Store Index										
Row 1	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 2	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 3	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 4	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 5	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 6	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 7	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row 8	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
.....	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
Row n	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
	Page 1	Page 2	Page 3	Page 4	Page 5	Page 6	Page 7	Page 8	Page 9	Page 10

Hash Index 1-2

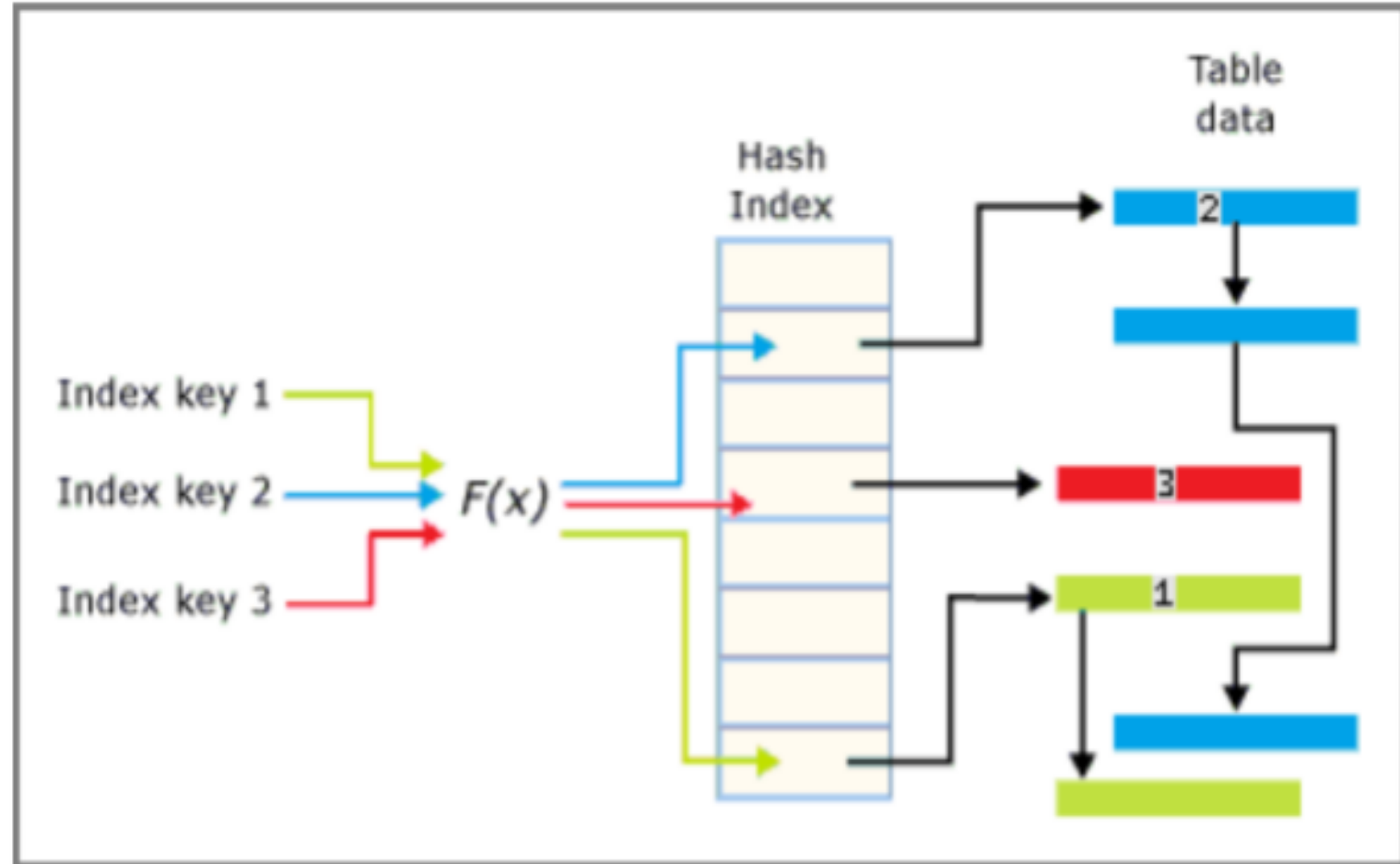
Hash index consists of an array of pointers and each element of the array is called a hash bucket.

- Each entry is a value for an index key, and address of its corresponding row in the underlying memory-optimized table.
- Each entry points to the next entry in a link list of entries, all chained to the current bucket.

The number of buckets must be specified at index definition time:

- The lower the ratio of buckets to table rows or to distinct values, the longer the average bucket link list will be.
- Short link lists perform faster than long link lists.

Hash Index 2-2

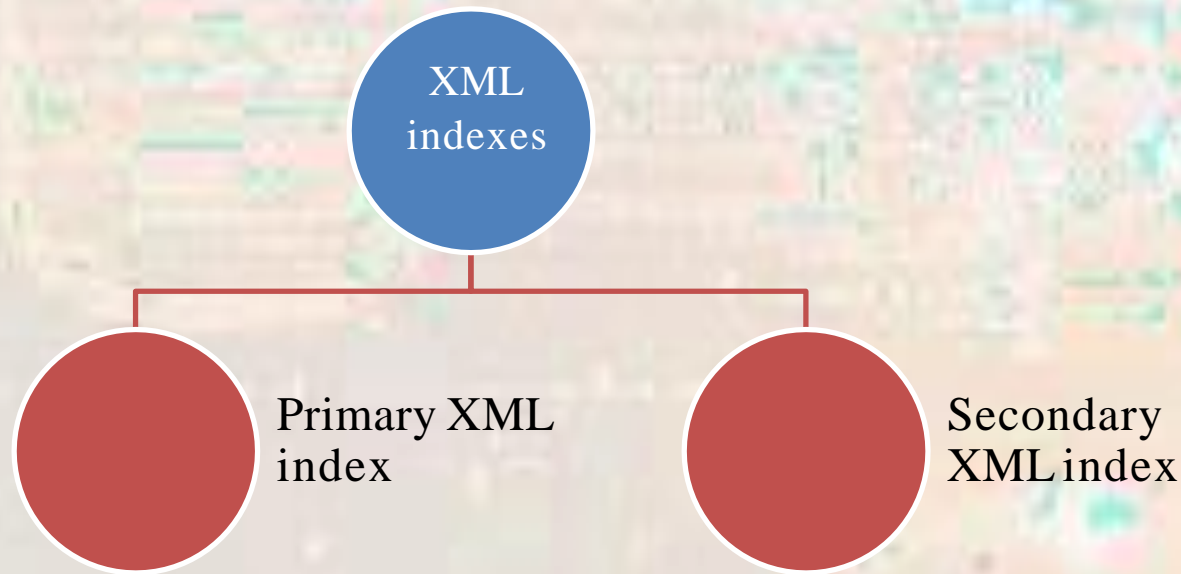


Hash Index Architecture

XML Indexes

XML indexes can be created on xml data type columns. They index all tags, values, and paths over the XML instances in the column and benefit query performance.

Queries on XML columns are common in your workload. XML index maintenance cost during data modification must be considered.



Spatial Indexes

- In SQL Server, spatial indexes are built using B-trees, which means that the indexes must represent the 2dimensional spatial data in the linear order of B-trees.

- The index-creation process decomposes the space into a four-level grid hierarchy.
- These levels are referred to as level 1(the top level), level 2, level 3, and level 4.



Four Levels of 4x4 Grids

Full-Text Indexes

- Creating and maintaining a full-text index involves populating the index by using a process called a population also known as a crawl.

Types of population

A full-text index supports the following types of population:

- Full population
- Automatic or manual population based on change tracking
- Incremental population based on a timestamp

Index Management

Index management allow user to manage indexes in terms various operation such as CREATE, ALTER, DROP, and so on.

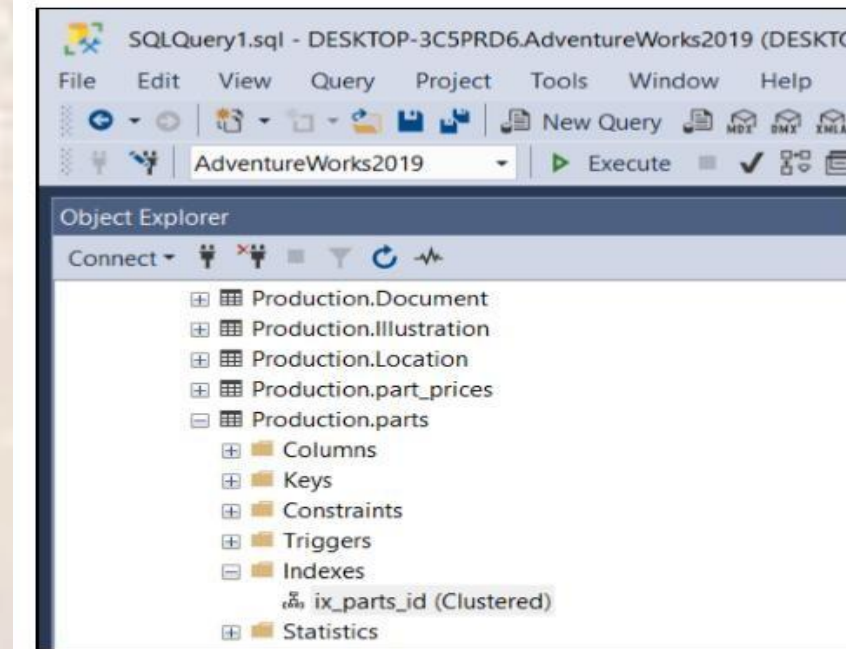
Create Clustered Index 1-5

- CREATE CLUSTERED index statement allow users to create CLUSTERED index on specified columns and table.

```
CREATE CLUSTERED INDEX index_name ON <table> (column_list)
```

```
CREATE CLUSTERED INDEX IX_CustID ON Customer(CustID)
```

- A clustered index is automatically created when a primary key is defined on the table.
- In a table without a primary key, clustered index should ideally be defined on:
 - Key columns that are searched on extensively.
 - Columns used in queries that return large resultsets.
 - Columns having unique data.
 - Columns used in table join.



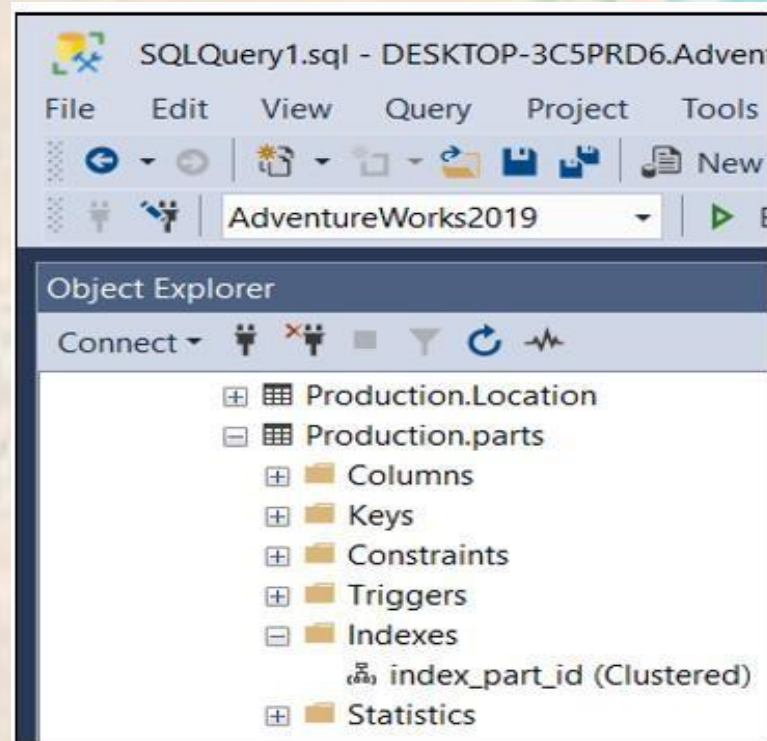
Create Clustered Index 2-5

RENAME INDEX

- The `sp_rename` is a system stored procedure that allows you to rename any user-created object in the current database including table, index, and column.

Syntax

```
EXEC sp_rename  
index_name,  
new_index_name,  
N'INDEX';
```



Renamed Index

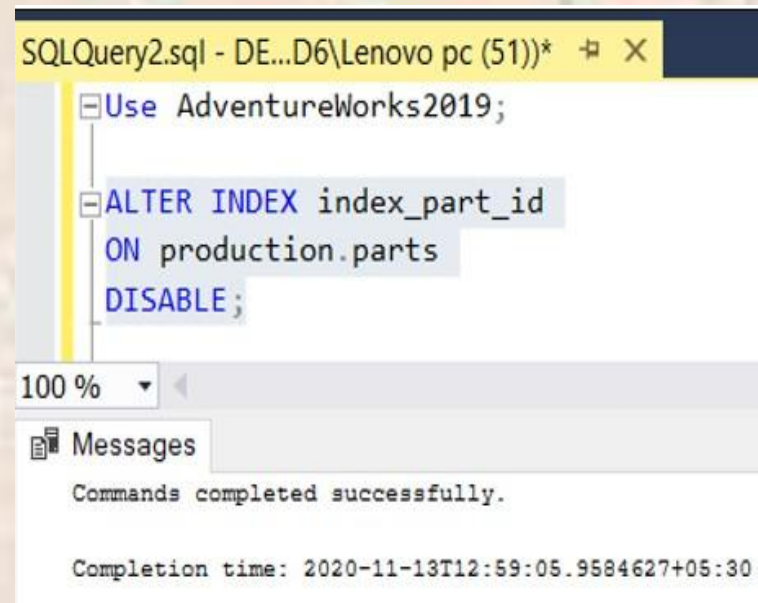
Create Clustered Index 3-5

DISABLE INDEX

To disable an index, **ALTER INDEX** statement is used as follows:

Syntax

```
ALTER INDEX index_name ON  
table_name  
DISABLE;
```

A screenshot of a SQL Server query window titled 'SQLQuery2.sql - DE...D6\Lenovo pc (51))'. The query text is: 'Use AdventureWorks2019; ALTER INDEX index_part_id ON production.parts DISABLE;'. The window shows a zoom level of 100% and a 'Messages' pane at the bottom with the text 'Commands completed successfully.' and 'Completion time: 2020-11-13T12:59:05.9584627+05:30'.

Disable Index

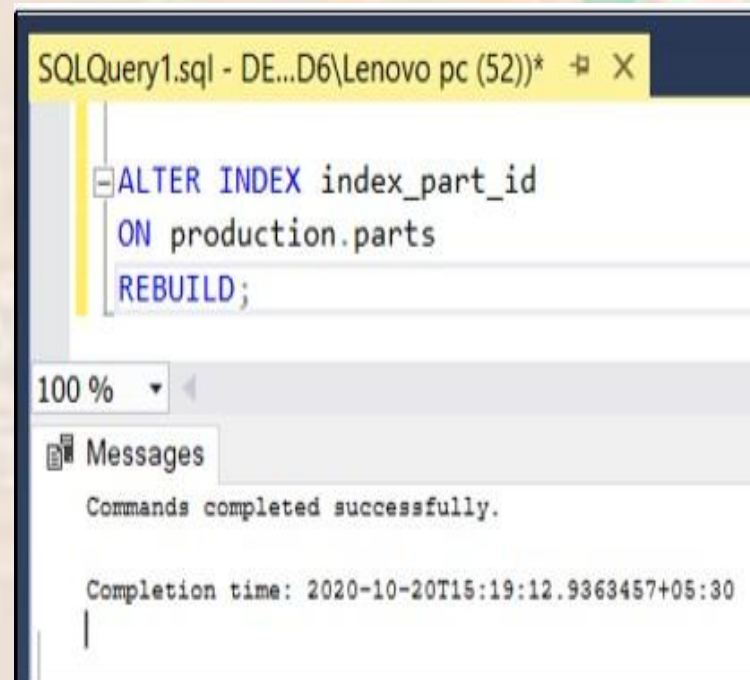
Create Clustered Index 4-5

ENABLE INDEX

This statement uses the **ALTER INDEX** statement to 'enable' or rebuild an index on a table.

Syntax

```
ALTER INDEX index_name  
ON table_name  
REBUILD;
```

A screenshot of a SQL Server query window titled 'SQLQuery1.sql - DE...D6\Lenovo pc (52))'. The query text is 'ALTER INDEX index_part_id ON production.parts REBUILD;'. Below the query editor, a 'Messages' pane shows the output: 'Commands completed successfully.' and 'Completion time: 2020-10-20T15:19:12.9363457+05:30'.

```
SQLQuery1.sql - DE...D6\Lenovo pc (52))  
  
ALTER INDEX index_part_id  
ON production.parts  
REBUILD;  
  
100 %  
Messages  
Commands completed successfully.  
  
Completion time: 2020-10-20T15:19:12.9363457+05:30
```

Enable Index

Create Clustered Index 5-5

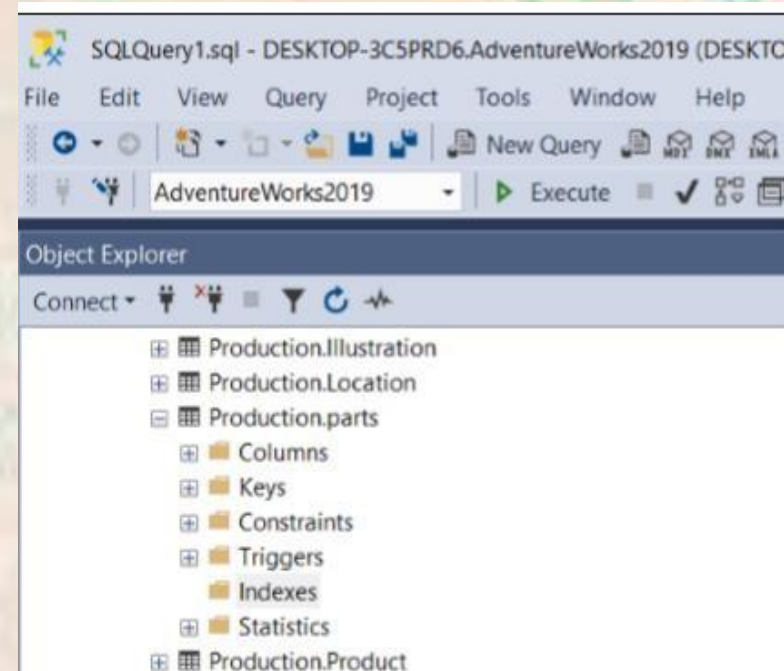
DROP INDEX

DROP INDEX statement removes one or more indexes from the current database.

Following is the syntax for **DROP INDEX** statement:

Syntax

```
DROP INDEX [IF EXISTS]  
index_name  
ON table_name;
```



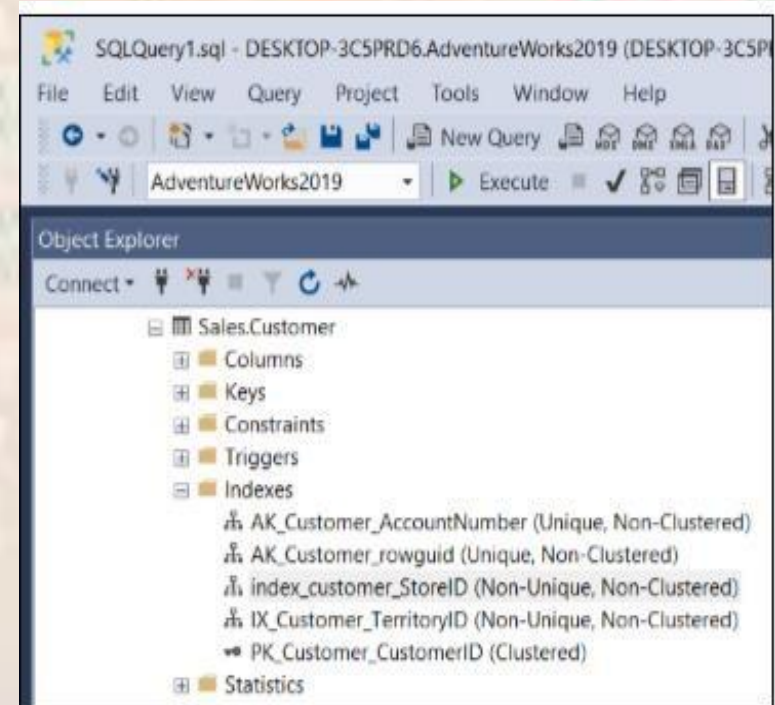
Drop Indexes

NonClustered Indexes

- It improves speed of data retrieval from tables. Unlike a clustered index, it sorts and stores data separately from the data rows in table.

Syntax

```
CREATE [NONCLUSTERED] INDEX index_name  
ON table_name(column_list);
```



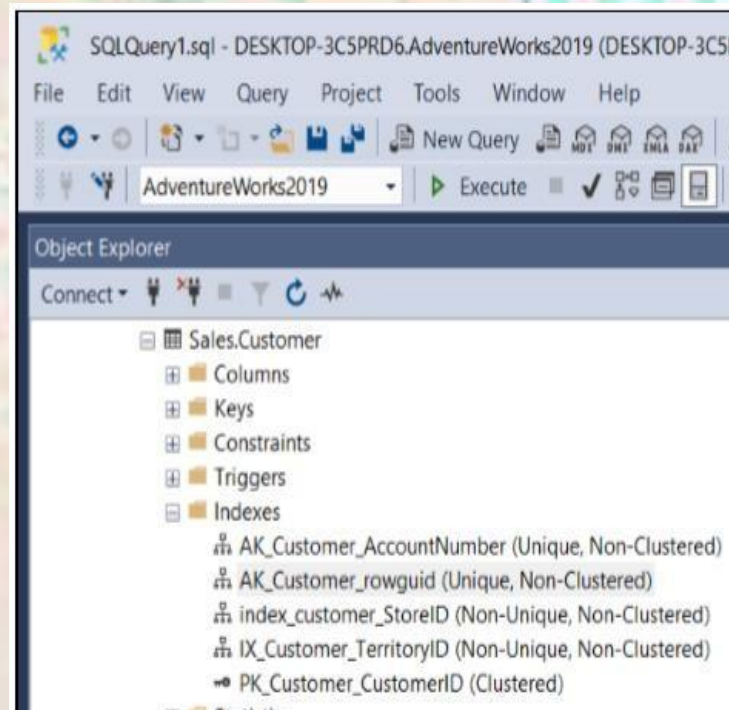
Nonclustered Index

Unique Indexes

- Ensures that index key columns do not contain any duplicate values.
- It may consist of one or many columns.
- In case the unique index has multiple columns, the combination is unique.

Syntax

```
CREATE UNIQUE INDEX  
index_name  
ON table_name(column_list);
```



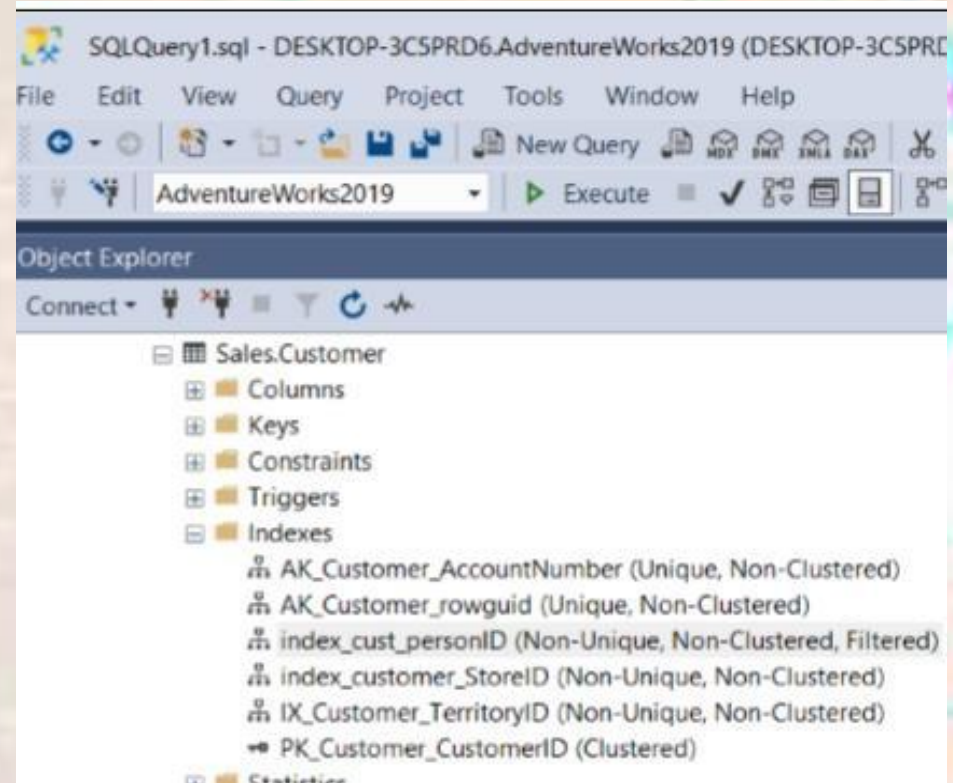
Unique Index

Filtered Indexes

- It is a nonclustered index with a predicate that allows to specify which rows should be added to the index.

Syntax

```
CREATE INDEX index_name  
ON table_name(column_list)  
WHERE predicate;
```



Filtered Index

Partitioned Table and Indexes

- SQL Server supports both table and index partitioning.
- Partitioning large tables or indexes can have the following manageability and performance benefits.

Benefits of Partitioning

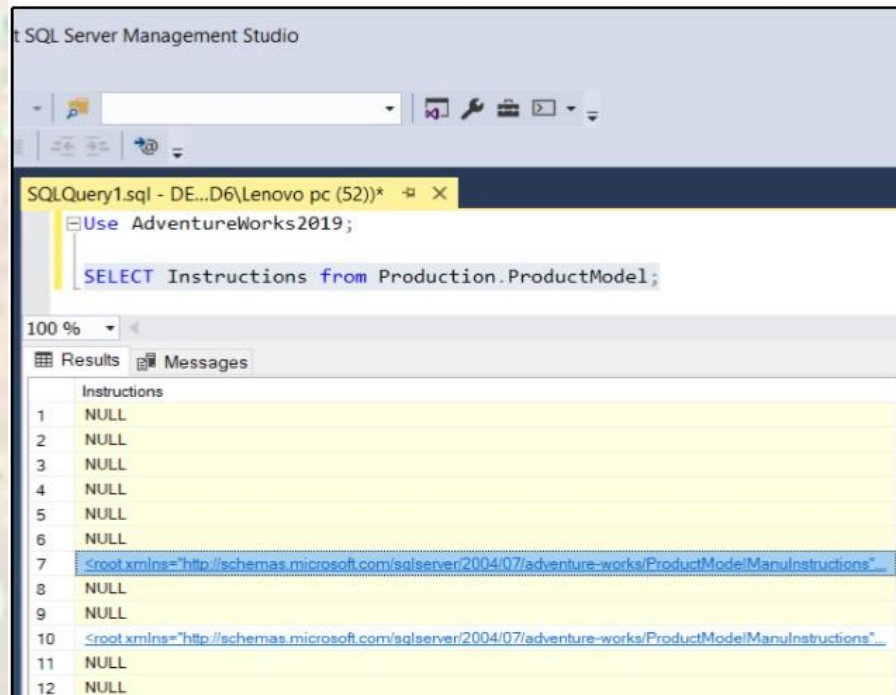
Transfer or access subsets of data quickly and efficiently.

Perform maintenance operations on one or more partitions. These operations are more efficient because they target only these data subsets, instead of the whole table.

Improve query performance, based on the types of queries you frequently run and on your hardware configuration.

XML Indexes 1-2

- XML data is stored in xml type columns as large binary objects (BLOBs).
- These XML instances can be large and the stored binary representation.

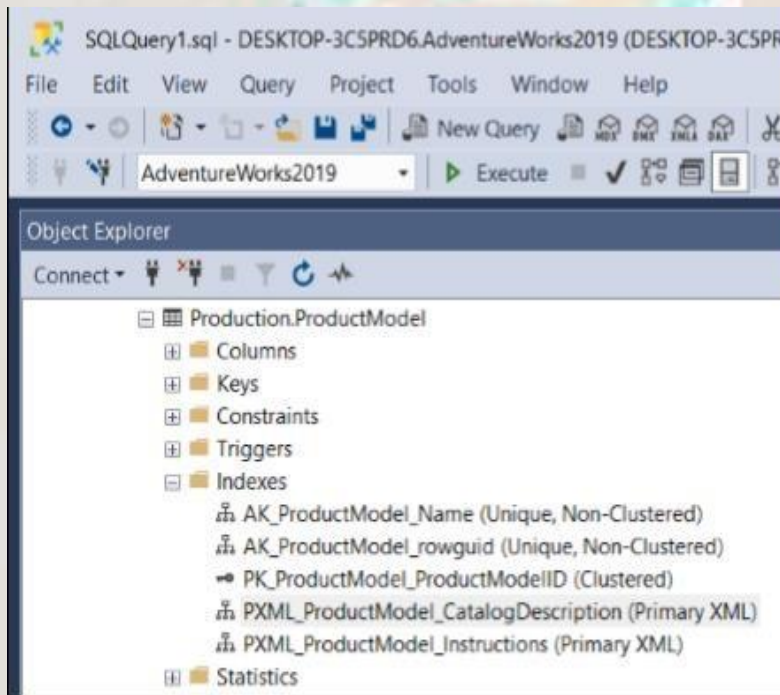


XML Data

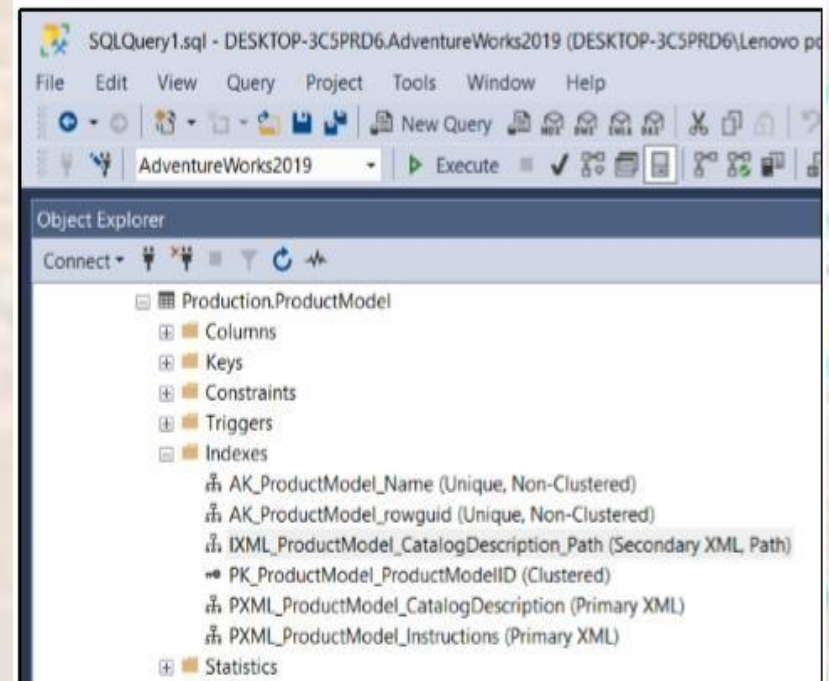
XML Indexes 2-2

- Primary XML index contains all the data in XML column.

- Secondary XML index creates a more specific index, based on the primary index.



Primary XML Index



Secondary XML Index

Types of XML Indexes 1-2

Primary XML Indexes

It is a special index created on each XML column to speed up these queries.

```
CREATE PRIMARY XML INDEX index_name ON <table_name> (column_name)
```

➤ Example

```
CREATE PRIMARY XML INDEX XML_Product ON ProductModel (CatalogDesc) ;
```

Secondary XML Indexes

Can only be created on columns that already have a primary XML index, uses for :

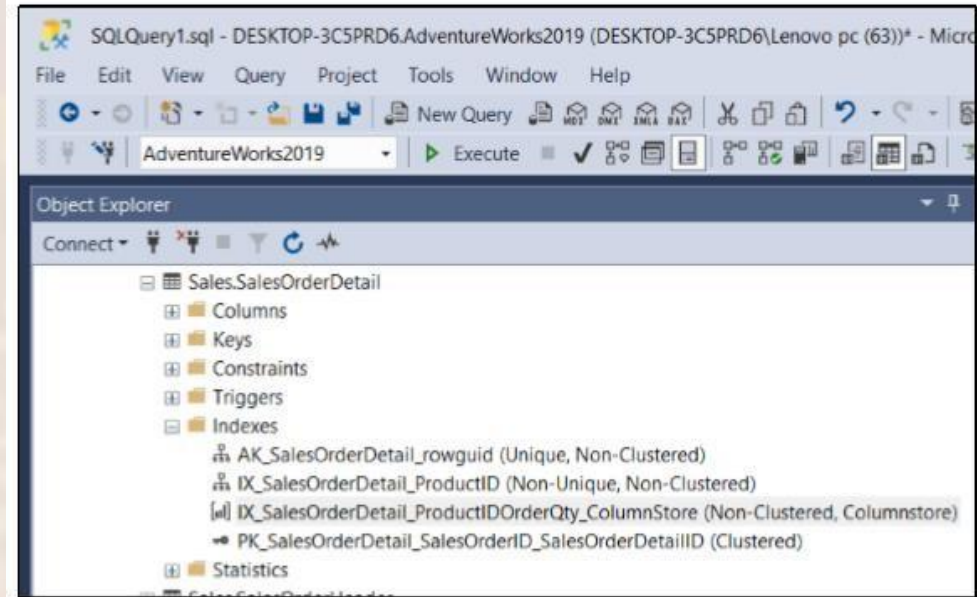
- Searching for values anywhere in the XML document.
- Retrieving particular object properties from within an XML document.

➤ Example

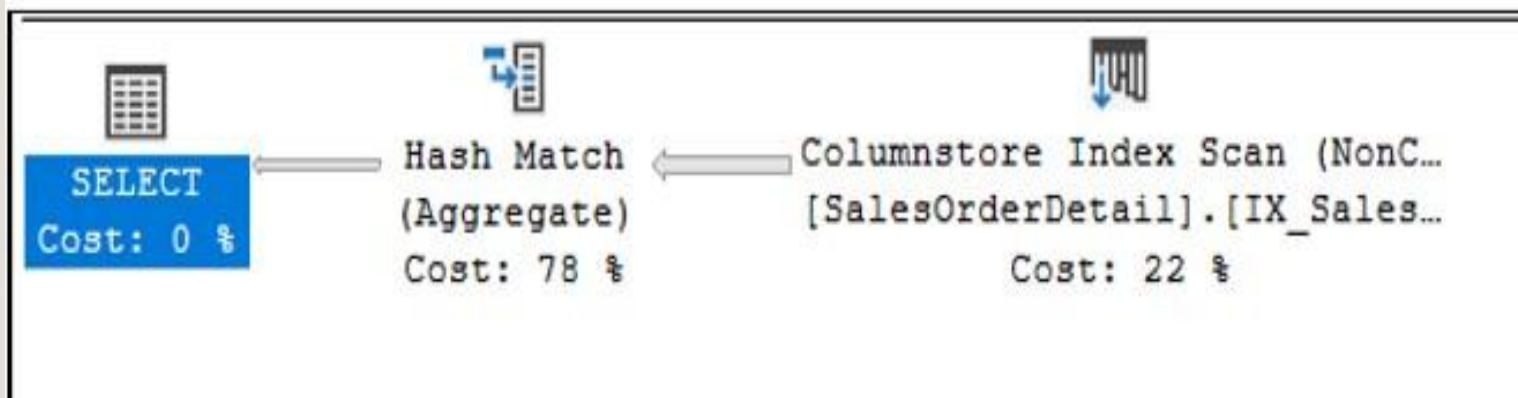
```
CREATE XML INDEX IXML_Product_Path ON  
ProductModel (CatalogDesc)  
    USING XML INDEX XML_Product FOR PATH ;  
GO
```

Columnstore Indexes

- Creating a column store index is done by using **CREATE COLUMNSTORE INDEX** command and has many of the same options as a regular index.



Columnstore Index



Estimated Execution Plan for Columnstore Index

Columnstore Indexes

```
CREATE [ NONCLUSTERED ] COLUMNSTORE INDEX index_name ON <object>
    ( column list )
    [ WITH ( <column_index_option> [ ,...n ] ) ]
```

- Example:

```
CREATE COLUMNSTORE INDEX [csindx_ResellerSales] ON
[ResellerSalesPtnd]
(
    [ProductKey], [OrderDateKey], [DueDateKey],
    [ShipDateKey], [CustomerKey], [EmployeeKey],
    [PromotionKey], [CurrencyKey], [SalesTerritoryKey],
    [SalesOrderNumber], [SalesOrderLineNumber],
    [RevisionNumber], [OrderQuantity], [UnitPrice],
    [ExtendedAmount], [UnitPriceDiscountPct], [DiscountAmount],
    [ProductStandardCost],
    [TotalProductCost], [SalesAmount], [TaxAmt],
    [Freight], [CarrierTrackingNumber], [CustomerPONumber],
    [OrderDate], [DueDate], [ShipDate]
);
```

Summary

- Indexes increase the speed of querying process by providing quick access to rows or columns in a data table.
- SQL Server stores data in storage units known as data pages.
- All input and output operations in a database are performed at the page level.
- A clustered index causes records to be physically stored in a sorted or sequential order.
- A nonclustered index is defined on a table that has data either in a clustered structure or a heap.
- XML indexes can speed up queries on tables that have XML data.
- Column Store Index enhances performance of data warehouse queries extensively.