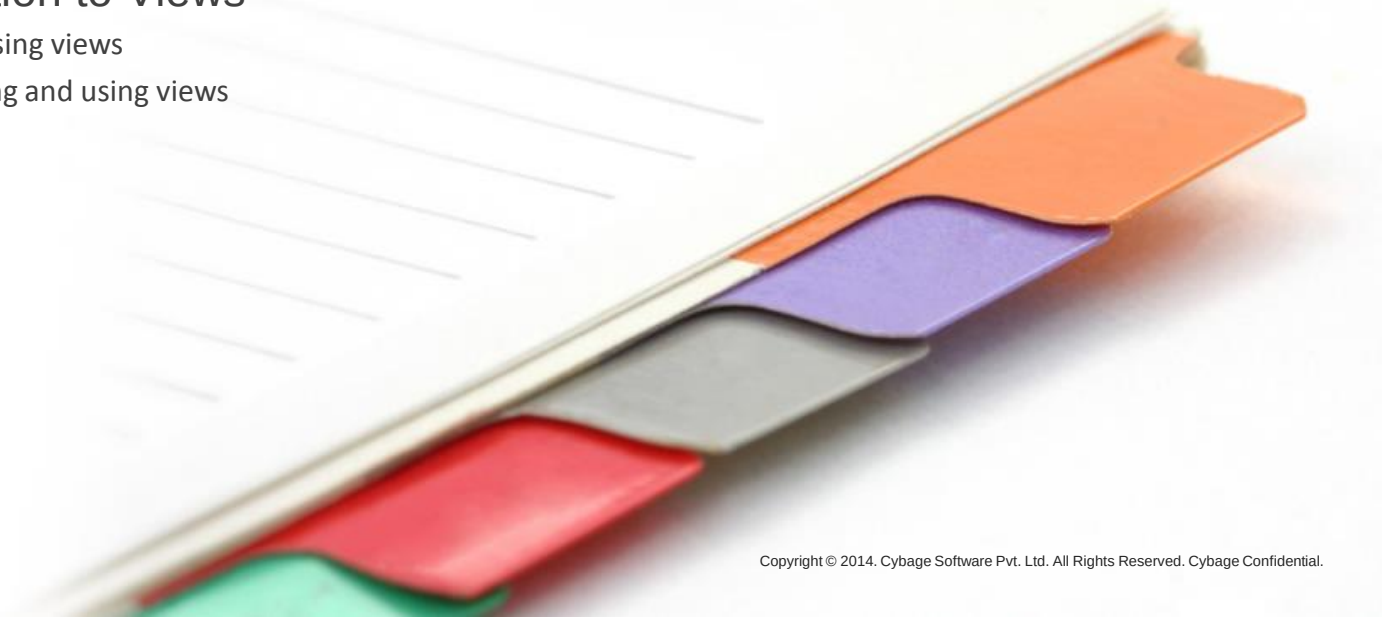
Advanced SQL Server

Authored by : Sushant Banerjee
Email : sushantba@cybage.com

Presented by : Sushant Banerjee
Extn. 7210

Agenda

- Introduction to Indexes
  - Why using Index
  - Types of indexes
  - Creating indexes
- Introduction to Views
  - Why using views
  - Creating and using views



Indexes

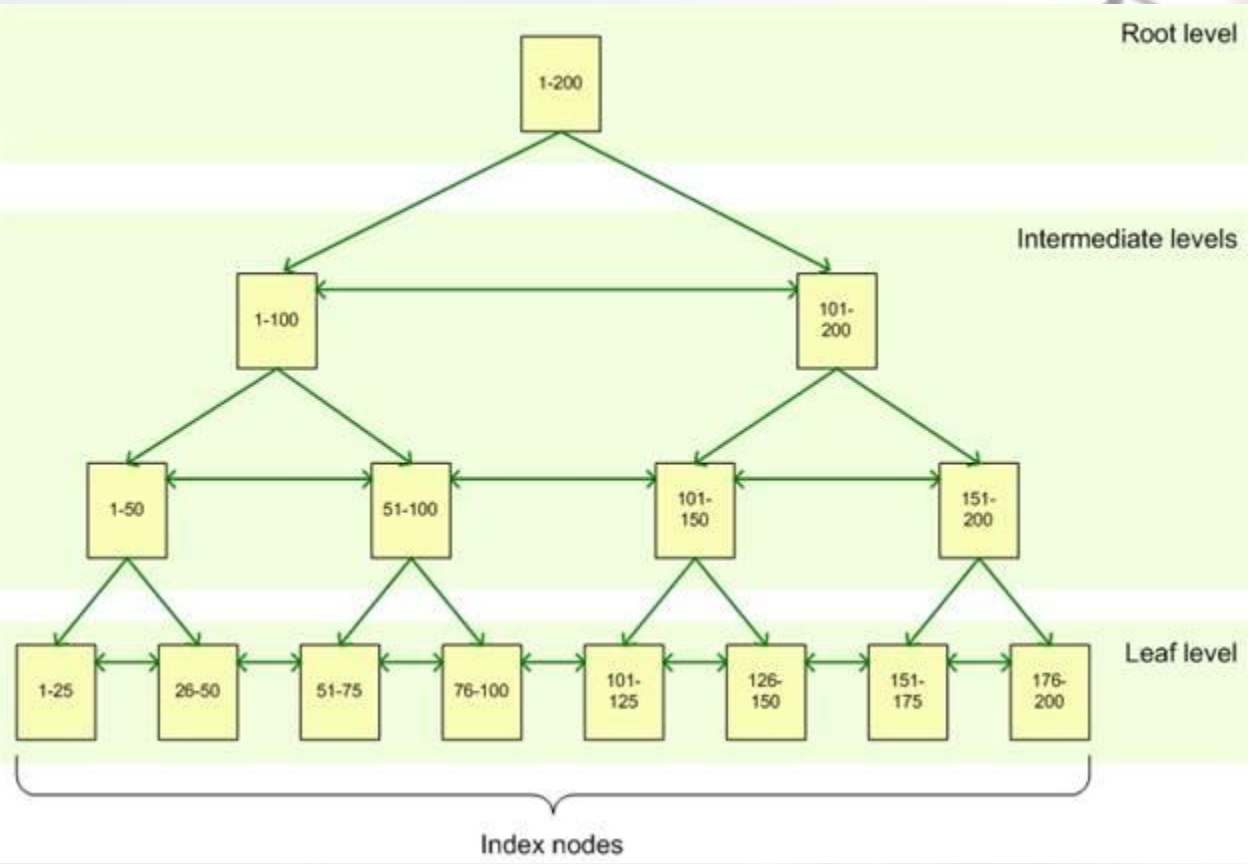
What is an Index

- An index enables you to **quickly search** and find information stored in the database just like an index in a book.
- An index contains **keys and pointers** that map to the storage location of the specified data.
- Using indexes you can **improve the performance** of your queries that access data from the database.
- Indexes are also used to enforce **data integrity** by making rows unique in the tables of your database.

What is an Index B-tree

- An index is consists of set of pages or index nodes that are organized in a B-tree structure.
- Each index node contains a set of keys and values that are used by SQL Server to locate the data row.
- A B-tree is a hierarchical structure having root node at the top of the hierarchy and leaf node at the bottom of the hierarchy.

B-tree Structure



B-tree Processing

- The query engine starts at the root node and navigates down through the intermediate nodes until it reaches the leaf node.
- The leaf node contains either the entire row of data or a pointer to that rows, depending on whether the index is **Clustered** or **Non-Clustered**.

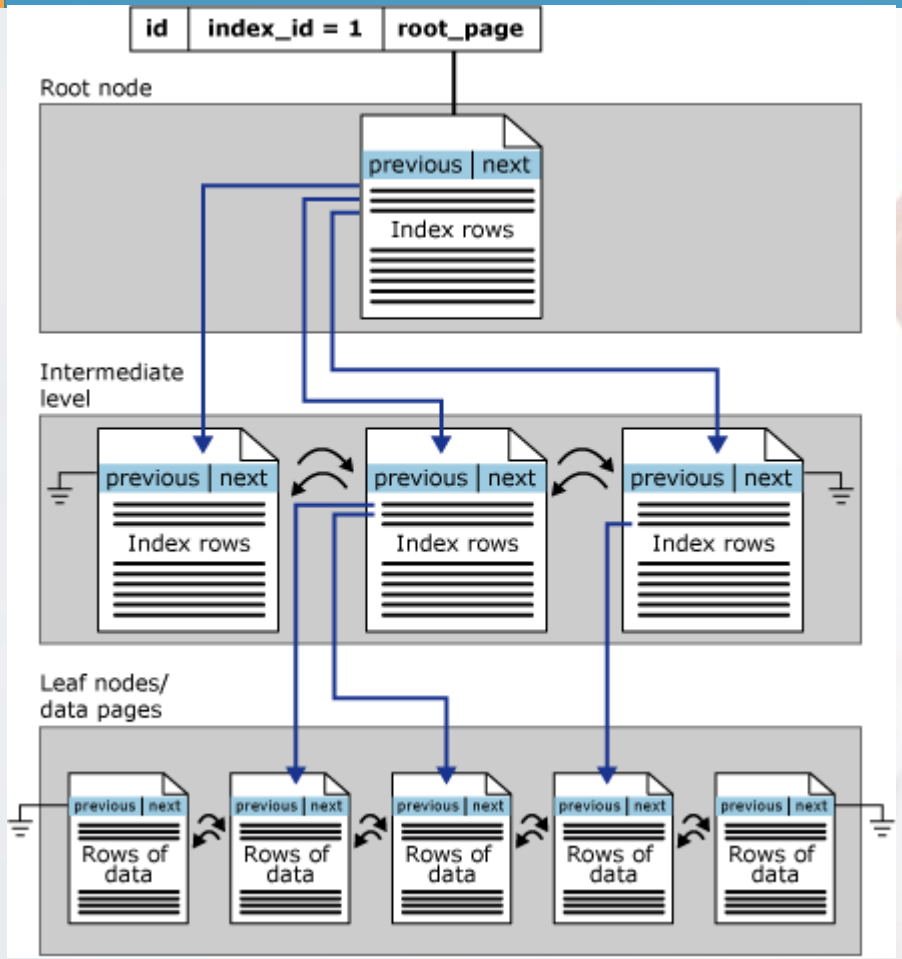
Types of Indexes

- Clustered Index
  - Stores actual data rows
- Non-Clustered Index
  - Doesn't store data
  - Has row locator

Clustered Index

- A clustered index stores **actual data rows** at the leaf level of the index.
- Data in a table is **sorted** only if a clustered index has been created on a table.
- There can be **only one** clustered index per table.
- A table is also called a **clustered table** if it has a clustered index defined on it.

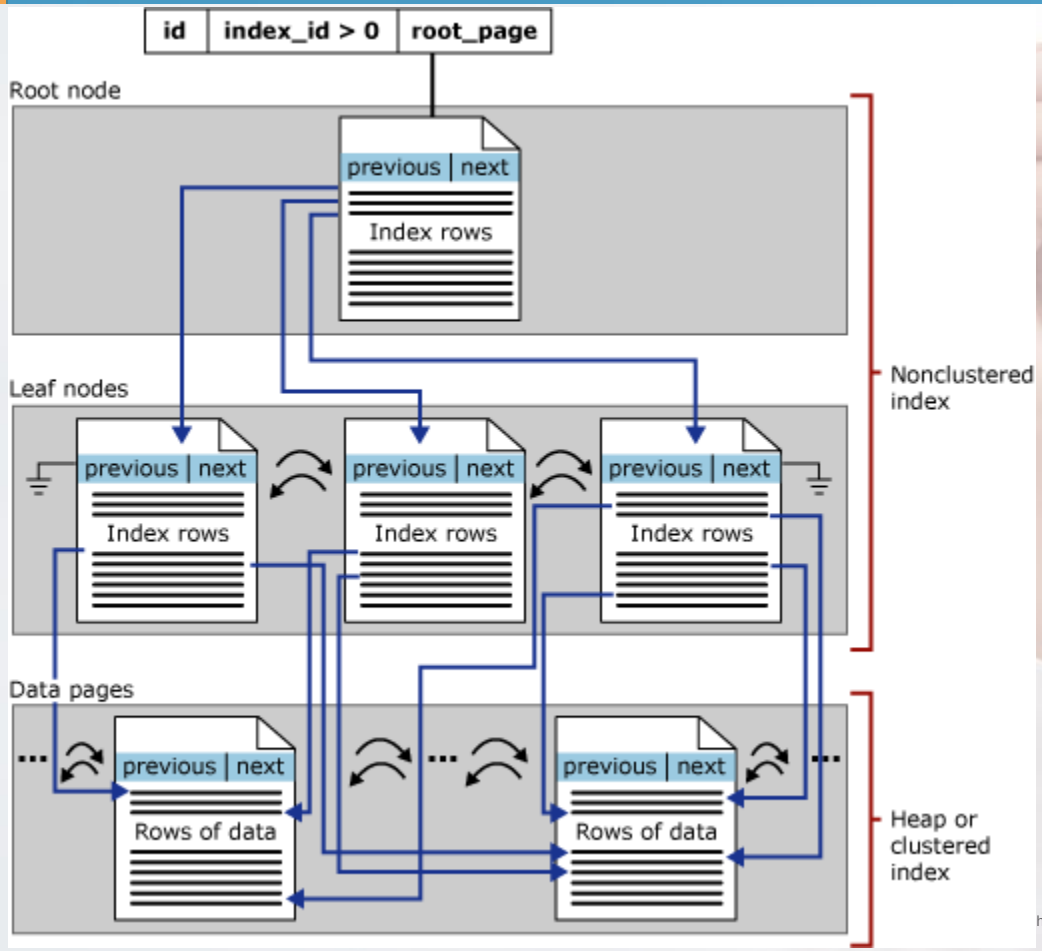
Clustered Index Architecture



Non-Clustered Index

- A Non-Clustered index stores the values from the indexed columns and **row locators** that point to the actual data rows.
- A table is also called a **heap** if it has no clustered index defined on it.
- The row locators in a Non-Clustered index are either a **pointer to a clustered table** or **to a heap**.
- The row locators **point to the clustered index** if the table already has a clustered index defined on it.
- The row locators **point to the actual data row** in the heap if the table has no clustered index.
- The row locators or pointers are also known as **Row ID or RID**.

Nonclustered Index Architecture



Clustered Vs. Nonclustered

- Both the clustered and Non-Clustered indexes have the same B-tree structure except the following differences:

Clustered Index	Non-Clustered Index
Data rows are sorted based on their clustered keys.	Data rows are not sorted.
The leaf layer of clustered index is made up of **data pages**.	The leaf layer of Non-Clustered index is made up of **index pages**.
Only one clustered index per table can be created.	Up to 249 in SQL Server 2005 and up to 999 in SQL Server 2008 can be created per table.

Composite and Unique Indexes

- Both the clustered and Non-Clustered indexes can be of following types:

- **Composite Index**
  - An index that contains more than one column.
  - Can include up to 16 columns per index unless it exceeds 900 bytes max size limit of each indexed column.

- **Unique Index**
  - An index that ensures the uniqueness of each value in the indexed column.
  - If the index is a composite the uniqueness is enforced across the columns as a whole, not on the individual columns.

Unique Index

- A unique index is created automatically when you create a **primary key or a unique constraint.**
- When you create a primary key SQL Server automatically creates a **unique clustered index**.
- When you create a unique constraint SQL Server automatically creates **a unique Non-Clustered index**.

General Index Design Guidelines

- SQL Server automatically updates indexes after each DML (INSERT, UPLDATE, DELETE) statements.
- Large number of indexes on a table results in performance problem.
- Create indexes as few as possible on tables which are heavily updated.
- Create many indexes on tables having large volume of data with low update requirements, such as history data.

Clustered Index Design Guidelines

- Every table should have a clustered index defined ideally on the following types of column or columns :

- Columns that are used for frequently used queries.
- Columns that provide high degree of uniqueness.
- Columns that can be used in range queries.
- Columns that is defined as IDENTITY.
- Columns that are used frequently to sort data in the result set.

- **Note** : Clustered indexes are not good choice for the columns that are frequently updated.

Nonclustered Index Guidelines

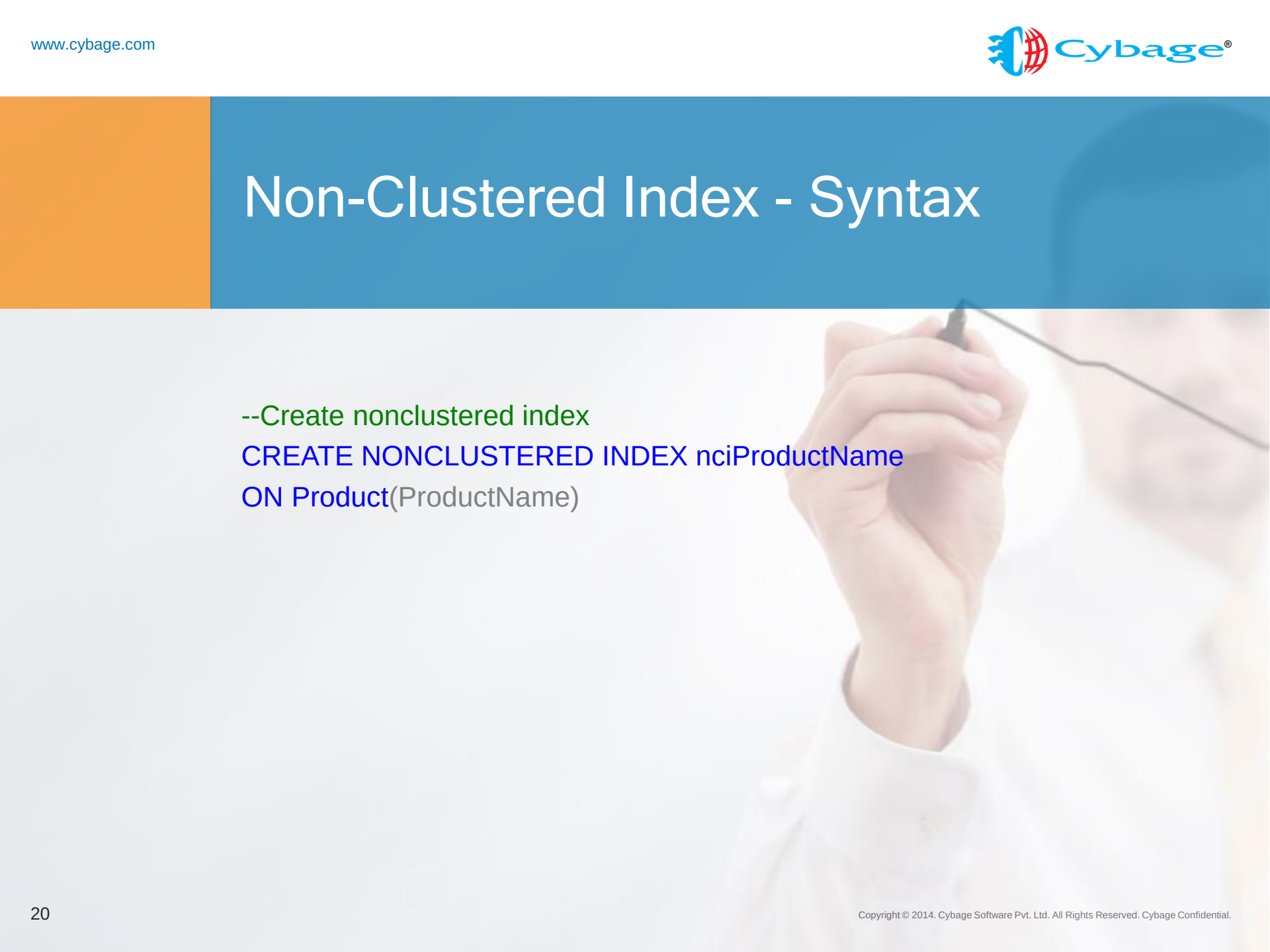
- Create multiple Non-Clustered indexes on columns involved in join an grouping operations.
- Create Non-Clustered indexes on columns frequently involved in search conditions of a query, such as WHERE clause that return exact matches.

Clustered Index - Syntax

```
--Create a unique clustered index
CREATE UNIQUE CLUSTERED INDEX uciProductId
ON Product(ProductId)
```

Non-Clustered Index - Syntax

```
--Create nonclustered index
CREATE NONCLUSTERED INDEX nciProductName
ON Product(ProductName)
```



View Index - Syntax

```
--View all indexes created on a table
SP_HELPINDEX Product
```

Composite Index - Syntax

```
--Creating nonclustered Composite index
CREATE INDEX idxUnitPriceProductName
ON Product(UnitPrice, ProductName)
```

Unique Index - Syntax

```
--Create Unique index
CREATE UNIQUE INDEX uidxCustomerPhone
ON Customer(Phone)
```

Renaming Index - Syntax

```
--Rename an index
SP_RENAME 'Product.uciProductId','uciIdxProductId’
```

Deleting Index - Syntax

```
--Delete an index
DROP INDEX Product.uciIdxProductId
```

Views

What is a View?

- A view is a virtual table whose contents are defined by the SELECT statements.
- A view is just like a real table with set of columns and rows but a view does not store data.
- A view produces rows and columns of data from the base tables dynamically whenever it is referenced.
- A view is stored in the database as a query not as a table.
- A view acts like a filtered data from the base table referenced in the view.
- A view can have maximum of 1024 columns.

Why Views?

- To represent **focused, simplified, and customized data**
  - Each user view only the data he or she supposed to view
- As a **security mechanism**
  - **By** allowing users to access data through the view
  - Without granting the users permissions to directly access the underlying base tables.
- To provide a **backward compatibility**
  - No need to update a view for a table whose schema has changed.

Types of Views

- Standard Views
- Indexed Views
- Partitioned Views

Creating a View

```
CREATE VIEW vwEmployee
AS
SELECT iEmployeeId, cFirstName, cLastName, cJobTitle
FROM Employee
```

Restrictions

- The SELECT statement in a view definition can not include the followings :
- COMPUTE or COMPUTE BY clauses
- An ORDER BY clause, unless there is a TOP clause used in the SELECT statement.
- The INTO keyword.
- A reference to a Temporary table.

Modify Views

```
--Modifying the select statement
ALTER VIEW vwEmployee
AS
SELECT iEmployeeId, cFirstName, cLastName
FROM Employee
```

Rename Views

```
--Rename View(SP_RENAME Old_View_Name, New_View_Name)
SP_RENAME vwEmployee, vwEmp
```

Deleting a View

```
--Deleting View
DROP VIEW vwEmployee
```

Modifying Data Through Views

- You can modify data of an underlying base table through a view.
- You can use INSERT, UPDATE and DELETE statements just like you use them with tables.

- Following operations are possible through views :
- Inserting data
- Updating data
- Deleting data

Update Restrictions

- Following restrictions apply while modifying data through views :
- Any modification must refer columns from only one base table.
- The column in views that are being modified can not use
  - An Aggregate Function
  - Set operators (UNION, UNION ALL, EXCEPT, INTERSECT)
  - GROUP BY, HAVING, DISTINCT
- TOP clause can not be used when WITH CHECK OPTION is used to create a view.

SCHEMABINDING Clause

- Creating a view with SCHEMABINDING clause **binds** the view to the schema of the underlying tables.
- If a view is created using SCHEMABINDING clause, **you can not modify (ALTER TABLE)** the underlying base table in a way that would affect the view definition.
- The SCHEMABINDING clause also **does not allow you to drop** the underlying base table unless you remove the underlying table reference from view definition.
- You need to use **two part names(schema.object)** of tables or views in the SELECT statement in case of SCHEMABINDING.

SCHEMABINDING Clause(contd…)

```
CREATE VIEW vwEmployee
WITH SCHEMABINDING
AS
SELECT EmpId, Name FROM dbo.Employee
```

A View with ENCRYPTION

- When you create views using WITH ENCRYPTION clause, the view text is encrypted.
- This clause also prevents views to be published as part of SQL Server replication.

```
CREATE VIEW vwEmployee
WITH ENCRYPTION
AS
SELECT EmpId, Name FROM Employee
```

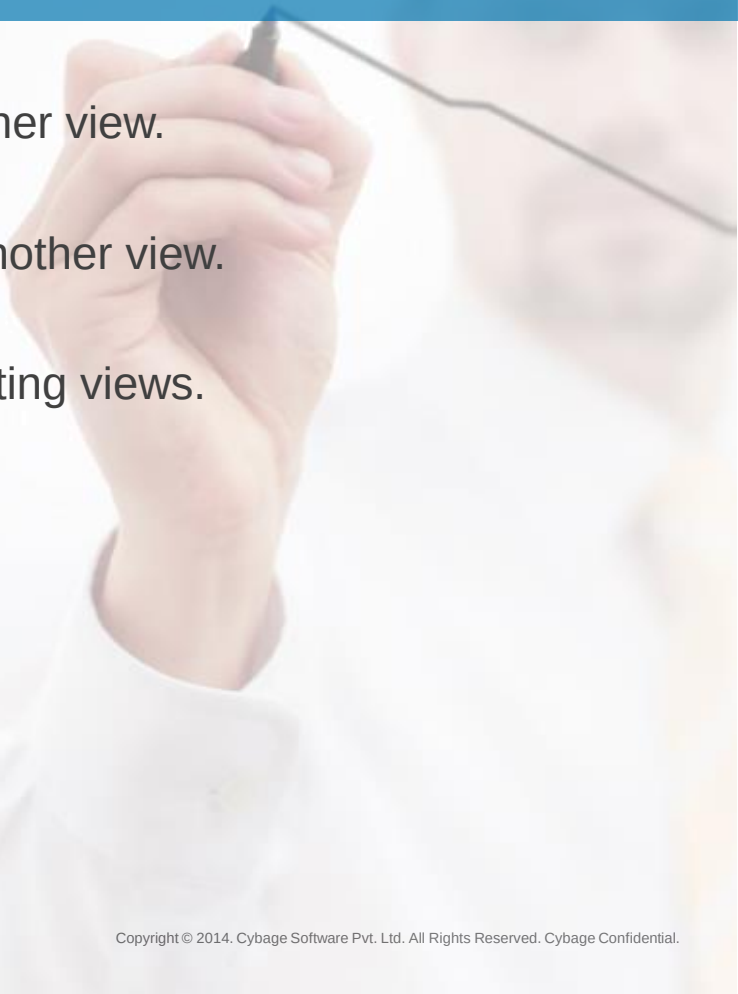
WITH CHECK OPTION

- WITH CHECK OPTION ensures that the data remains visible through the view after a row is modified through the view.
- SQL Server will not verify if you directly modify data in the underlying base table even after creating a view using WITH CHECK OPTION.

```
CREATE VIEW vwTestEmp
AS
SELECT EmployeeId, Name FROM Employee
WHERE EmployeeId = 2
WITH CHECK OPTION
```

Nested Views

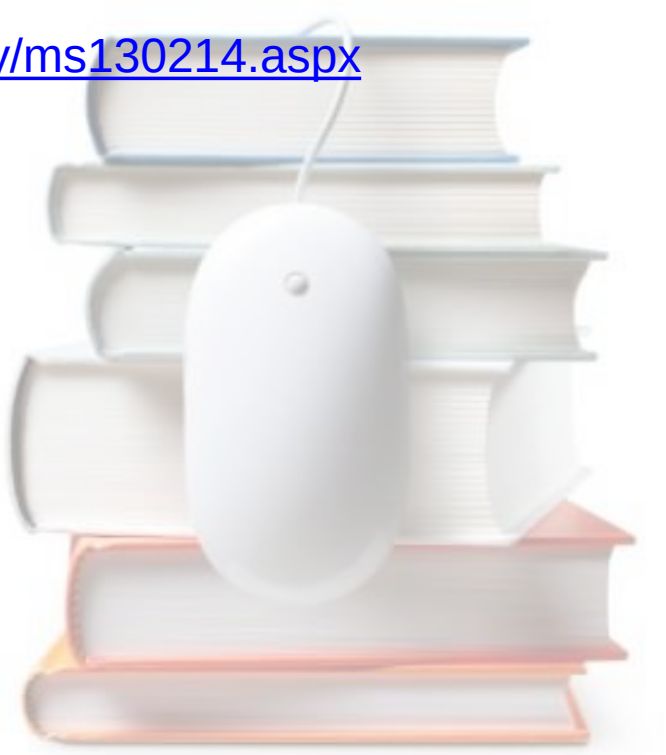
- You can create a view based on another view.
- A nested view access data through another view.
- SQL Server supports 32 levels of nesting views.



Bibliography, Important Links

WWW.MSDN.COM (SQL SERVER 2012 BOOKS ONLINE)

http://msdn.microsoft.com/en-us/library/ms130214.aspx



Any Questions?

Thank you!