MS Sql Server Indexes

Author: Nemanja Kojic MScEE
Indexing

- Vital for system performance
- Improves query execution performance
- NOT one size fits all – trade offs must be made
- Penalties during INSERT/UPDATE – index update
- Two types of indexes:
  - Clustered Indexes
  - NonClustered Indexes
Index example
Non-Clustered Index

• Data in pages in random order
• Logical data order in index
• NonClustered index tree
  – Keys in sorted order
  – Leaf pages contain pointers to rows in data pages
• Typically created on column used in JOIN, WHERE, ORDER BY
• Good for tables whose values may be modified frequently
NonClustered Index (cont.)

• MS Sql Server:
  CREATE INDEX -> nonClustered by default
• Allowed more than index on a db table
• MS Sql Server 2008:
  up to 999 nonClustered indexes per table
Non-Clustered Index example
Non-Clustered Index - summation

• Create index on columns which are:
  – Frequently used in search criteria
  – Used to JOIN different tables
  – Used as foreign key fields
  – Of having high selectability
  – Used in ORDER BY clause
  – Of type XML (primary and secondary indexes)
Clustered Index

• Re-orders data rows to match the index (rows in sort order on disk)
• Only one clustered index per table!
• Leaf level of the index tree - actual data rows
• Good for sequential access, and range selection
Clustered Index (cont.)

• MS Sql Server INSERTS data according to the way a clustered index was created

• Most often: PRIMARY KEY => Clustered Index

• Every table SHOULD have clustered index

• w/o clustered index: records added to the end of the last page

• w/ clustered index: data added to suitable position dictated by the index
Clustered Index example
Clustered and Non-Clustered Index combined
Covering indexes

- Extending functionality of nonCIs indexes
- Adding non-key columns to the leaf level
- Index covers more types of queries
- Covering Indexes = Indexes w/ incl. columns
- Great performance benefits
Filtering indexes

• NonClustered index with a record filter
• Covers a subset of records in a table
• Reduces storage space for index
• Better performance
• Decreased INSERT penalty
Index selectivity and Density

- **Selectivity:**
  number of distinct key values in the table
- **PRIMARY KEY, UNIQUE** – perfectly selective
- The higher selective Index, the better perform.
- **Density:**
  number of duplicate key values in the table
- **Query optimizer:** index seek, index scan
Fill factor

• Tuning storage and performance
• Fill factor = % of space for data in leaf pages
• Remainder of the page for future growth
• E.g. Fill factor=80%  =>  20% page empty
• Reserved space between index rows (rather than at the of the index)
• Applied on CREATE or REBUILD INDEX
Fill factor - guidelines

• Depends on how data are accessed
• Data inserted at the end of the table =>
  FILL FACTOR = 90%-100%
• Data inserted anywhere =>
  FILL FACTOR = 60%-80%
• The lower FF, the higher storage for the index
• In general: appropriate FF requires a lot of testing and probing
Creating indexes – Best Practices

• Keep indexes narrow (one or few columns)
• Clustered index on every table
• Clustered index on a highly selective column
• Clustered index on a column that is never upd.
• Default: clustered index on PRIMARY KEY col.
• Be aware of penalties during INSERT/UPDATE
• Eliminate duplicate indexes.
• Check the default FILL FACTOR
• Non-clustered indexes can be created in different file groups, which may increase performance
Order of fields on each index?

- Bad order => index is not useful
- Most selective columns go first
- Sql Server knows data distribution only for the first column!
- Don’t place column from clustered index to a non-clustered index
EXAMPLES
Table with NO indexes

```
select * …
```
Table w/ non-clust. index on LastName

```
CREATE NONCLUSTERED INDEX [IX_Person_Test_LastName] ON [Person].[person_test]
(  
    [LastName] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
```

Query 1: Query cost (relative to the batch): 100%
```
select * from [Person].[Person_test] where lastname='Brown'
```
Table w/ clust. index on LastName

```sql
CREATE CLUSTERED INDEX [IX_Person_Test_LastName_Clustered] ON [Person].[person_test]
(
    [LastName] ASC
)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF,
     DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
```

Query 1: Query cost (relative to the batch): 100%
```sql
select * from [Person].[Person_test] where lastname='Brown'
```
Table w/ non-clust. index on LastName selecting LastName

Query 1: Query cost (relative to the batch): 100%
select lastname from [Person].[Person_test] where lastname='Brown'

Index Seek (NonClustered)
(person_test).[IX_Person_Test_LastN...}
Cost: 100 %

SELECT
Cost: 0 %

SELECT
Cached plan size 16 B
Estimated Operator Cost 0 (0%)
Estimated Subtree Cost 0.0033832
Estimated Number of Rows 92

Statement
select lastname from [Person].[Person_test]
where lastname='Brown'
Table w/ clust. Index on LastName selecting LastName

Query 1: Query cost (relative to the batch): 100%
select lastname from [Person].[Person_test] where lastname='Brown'

Clustered Index Seek (Clustered)
[person_test].[IX_Person_Test_LastN... Cost: 100 %

SELECT
Cached plan size  16 B
Estimated Operator Cost  0 (0%)
Estimated Subtree Cost  0.0155815
Estimated Number of Rows  92

Statement
select lastname from [Person].[Person_test]
where lastname='Brown'
Table w/ non-clust. Index on LastName selecting LastName and FirstName

Query 1: Query cost (relative to the batch): 100%
select lastame, firstame from [Person].[Person_test] where lastame
Table w/ non-clust. Index on LastName including FirstName

CREATE NONCLUSTERED INDEX [IX_Person_Test_LastName_Include_FirstName] ON [Person].[person_test] ( [LastName] ASC )
INCLUDE (FirstName) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO

Query 1: Query cost (relative to the batch): 100%
select lastname, firstname from [Person].[Person_test] where lastname

Index Seek (NonClustered)
[person_test].[IX_Person_Test_LastN...
Cost: 100%

SELECT
Cached plan size  16 B
Estimated Operator Cost  0 (0%)
Estimated Subtree Cost  0.0033832
Estimated Number of Rows  92

Statement
select lastname, firstname from [Person]. [Person_test] where lastname='Brown'
Table with filtered index

```
CREATE NONCLUSTERED INDEX [IX_Person_Test_LastName_Filtered_ModifiedDate] ON [Person].[person_test] (LastName) ASC
WHERE ModifiedDate < '2005-01-01' WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
```

```
SELECT LastName FROM person.person_test WHERE modifieddate < '2005-01-01'
```

Query 1: Query cost (relative to the batch): 100%
```
SELECT lastname FROM person.person_test WHERE modifieddate < '2005-01-01'
```
Building indexes in Asc vs. Desc Order selecting all records

Query 1: Query cost (relative to the batch): 100%
select OrderDate from Purchasing.PurchaseOrderHeader

SELECT
Cost: 0 %

Clustered Index Scan (Clustered)
[PurchaseOrderHeader].[PK_PurchaseO...
Cost: 100 %

SELECT
Cached plan size 16 B
Estimated Operator Cost 0 (0%)
Estimated Subtree Cost 0.0380656
Estimated Number of Rows 4012

Statement
select OrderDate from Purchasing.PurchaseOrderHeader
Building indexes in Asc vs. Desc Order

select w/ ORDER BY ASC, no INDEX
Building indexes in Asc vs. Desc Order

select w/ ORDER BY ASC, with INDEX

CREATE NONCLUSTERED INDEX [IX_PurchaseOrderHeader_OrderDate]
ON [Purchasing].[PurchaseOrderHeader]
( [OrderDate] ASC )

Query 1: Query cost (relative to the batch): 100%
select OrderDate from Purchasing.PurchaseOrderHeader order by OrderDate

SELECT
Cost: 0 %

SELECT
Cached plan size 8 B
Estimated Operator Cost 0 (0%)
Estimated Subtree Cost 0.0128804
Estimated Number of Rows 4012

Statement
select OrderDate from Purchasing.PurchaseOrderHeader order by OrderDate
Building indexes in Asc vs. Desc Order
select w/ ORDER BY DESC, no INDEX

Query 1: Query cost (relative to the batch): 100%
select OrderDate from Purchasing.PurchaseOrderHeader order by OrderDate desc

SELECT Cost: 0 %
Sort Cost: 69 %
Clustered Index Scan (Clustered)
[PurchaseOrderHeader].[PK_PurchaseOrderHeader]
Cost: 31 %

SELECT
Cached plan size 16 B
Estimated Operator Cost 0 (0%)
Estimated Subtree Cost 0.124344
Estimated Number of Rows 4012

Statement
select OrderDate from
Purchasing.PurchaseOrderHeader order by OrderDate desc
Building indexes in Asc vs. Desc Order

select w/ ORDER BY DESC, with INDEX

```
CREATE NONCLUSTERED INDEX [IX_PurchaseOrderHeader_OrderDate]
ON [Purchasing].[PurchaseOrderHeader]
( [OrderDate] DESC )
```

Query 1: Query cost (relative to the batch): 100%
```
select OrderDate from Purchasing.PurchaseOrderHeader order by OrderDate desc
```

Index Scan (NonClustered)
```
[PurchaseOrderHeader].[IX_PurchaseO...
Cost: 100 %
```

SELECT
```
Cached plan size  8 B
Estimated Operator Cost  0 (0%)
Estimated Subtree Cost  0.0128804
Estimated Number of Rows  4012
```

Statement
```
select OrderDate from
Purchasing.PurchaseOrderHeader order by
OrderDate desc
```