



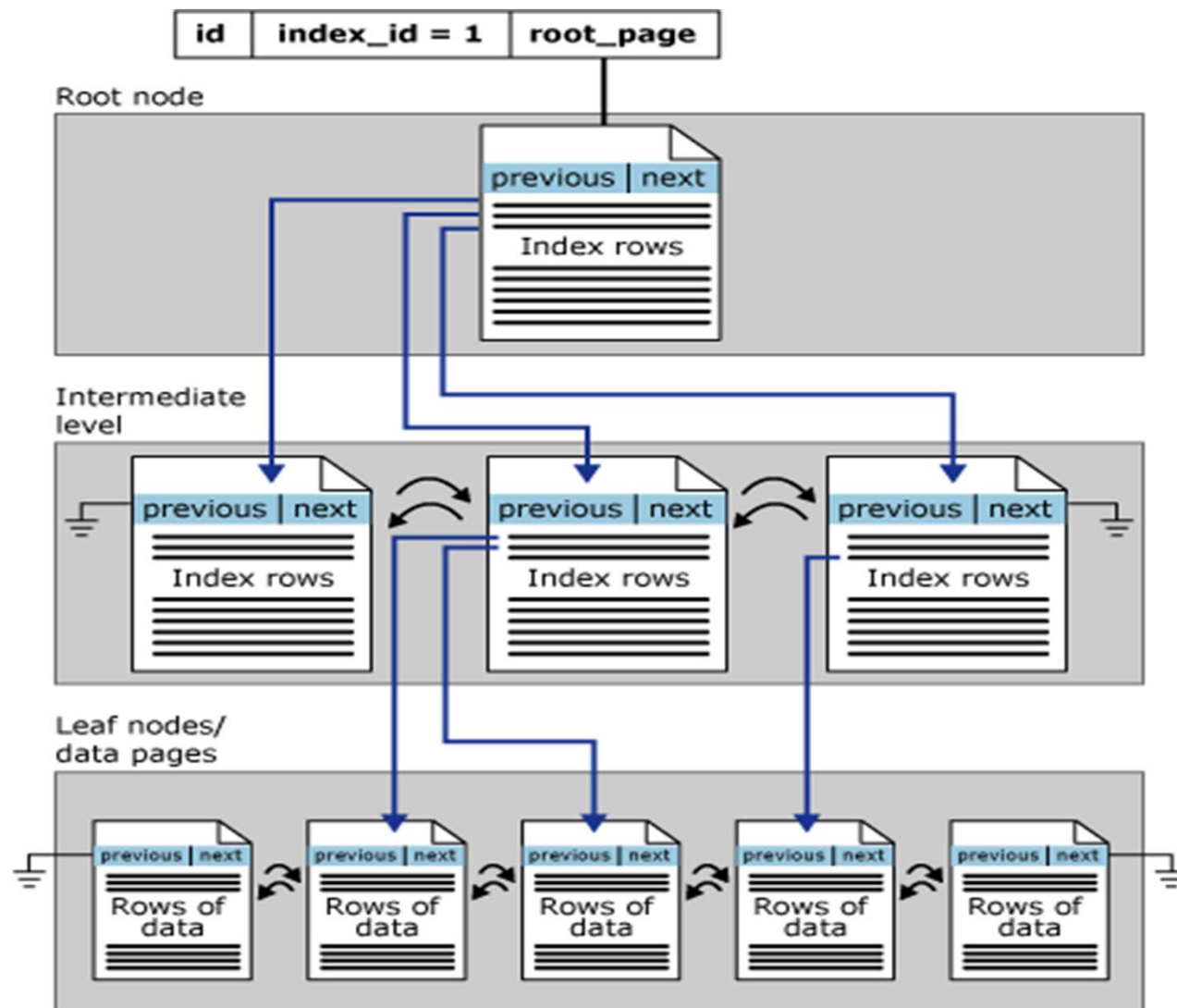
Index Maintenance



What is Index Maintenance?

- Once Indexes are created they can become fragmented over time
- Fragmentation happens when the logical order of pages in an index does not match the physical order in the data file
- This can become a major issue on the performance of your database over time
 - You need to have a strategy in place to deal with Index fragmentation
- We need ways that will reorganize our indexes to clean things up.

Clustered Indexes





Creating an Index

■ Syntax

```
CREATE      [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ]  
INDEX      index_name  
ON         TableName ( column [ ASC | DESC ] [ ,...n ] )  
WITH  
    PAD_INDEX = { ON | OFF }  
    | FILLFACTOR = fillfactor
```



FILL Factor

- Fillfactor specifies a **percentage** that indicates how full the Database Engine should make the leaf level of each index page during index creation or rebuild.
- Fillfactor must be an integer value from 1 to 100.
- If fillfactor is 100, the Database Engine creates indexes with leaf pages filled to capacity.
- The FILLFACTOR setting applies only when the index is created or rebuilt.



FILL Factor

■ FILL Factor

- Use FILL FACTOR to avoid page splits
- Specifies the percentage of free space that is reserved
- A fill factor is a specification done during the creation of indexes
- It specifies that a particular amount of space is left on a leaf level pages
- FILLFACTOR applies to the bottom layer (leaf level page)
- FILLFACTOR values
 - can be from 1 through 100
 - If a value is not specified, the default is 0
 - Fill factor values 0 and 100 are the same



FILL Factor - Example

```
CREATE INDEX IX_Employee ON Employee  
(OrganizationLevel, OrganizationNode)  
WITH FILLFACTOR = 80)
```

```
ALTER INDEX IX_Employee ON Employee  
REBUILD WITH (FILLFACTOR = 80);
```

```
CREATE INDEX IX_Employee ON Employee  
(OrganizationLevel, OrganizationNode)  
WITH (DROP_EXISTING = ON, FILLFACTOR = 80);
```



PAD Index

■ PAD Index

- PAD_INDEX ON means "Apply FILLFACTOR to all layers"
- PAD_INDEX is only useful if FILLFACTOR is set
- You set PAD_INDEX ON when you expect a lot of changes

■ PAD_INDEX = { ON | OFF }

- Specifies index padding
- The default is OFF.
- ON: The percentage of free space that is specified by fillfactor is applied to the intermediate-level pages of the index.



Example

- Rebuilds the with a fill factor of 80 on the Employee table.

```
ALTER INDEX IX_Emp ON Employee  
REBUILD  
WITH (FILLFACTOR = 80);  
GO
```



ALTER INDEX Command

- Most ALTER commands are about changing the definition of the object
 - For Example: ALTER TABLE is used to add or disable constraints and columns
- ALTER INDEX is different
 - it is all about maintenance and
 - zero about structure
- If you need to change the makeup of your index
 - you need to either DROP and CREATE it or



ALTER Index - Syntax

```
ALTER INDEX { <name of index> | ALL }  
ON      <table name>  
{  
    REBUILD  
    |  DISABLE  
    |  REORGANIZE  
}
```

■ Index Name

- ☐ Name of a specific index if you want to maintain one specific index, or
- ☐ use ALL to perform maintenance on every index on with the named table.

■ Table Name: the name of the specific table on which the index is built



Index REBUILD Option

- The old index is completely thrown away and reconstructed from scratch
- every page in both the leaf and non-leaf levels of the index will be reconstructed as you have defined them
- The index cannot be used until the rebuild is complete
 - the creation of the index will be a significant load on the system
 - Schedule INDEX REBUILDing to run in off hours
 - The Enterprise version of SQL Server includes a special ONLINE version of this option that will keep the index alive and build the new one parallel to the old



Index DISABLE Option

- This command essentially marks the index as unusable.
- Once an index has been disabled, it must be rebuilt
 - **Not reorganized**, but rebuilt - before it will be active again.
- If you disable the clustered index for your table
 - it has the effect of disabling the table
 - The data will remain, but will be inaccessible by all indexes
 - Since they all depend on the clustered index
- The entire table is effectively offline until you rebuild the clustered index.



Index REORGANIZE Option

- Reorganize index is an online operation
- Users can still utilize the index during the operation
- REORGANIZE works only on the **leaf level** of your index
 - Non-leaf levels of the index go untouched.
- This means that we're not quite getting a full optimization
- This is usually the tool you'll want to use as part of your regular maintenance plan;



Examples

- Reorganize all the indexes on the table

```
ALTER INDEX ALL  
ON      TransactionHistory  
REORGANIZE
```

- Rebuild an existing index and applies the specified fill factor during the rebuild operation

```
ALTER INDEX IX_Employee_Org ON Employee  
REBUILD WITH (FILLFACTOR = 80)
```



Example

- Add a column to an index
- The following example creates index with two columns from the FactFinance table.
- The next statement rebuilds the index with one more column and keeps the existing name.

```
CREATE INDEX IX_FF ON FactFinance (FinanceKey, DateKey);
```

```
-- Rebuild and add the another column to the index
```

```
CREATE INDEX IX_FF ON FactFinance (FinanceKey, DateKey, OrgKey)  
WITH (DROP_EXISTING = ON);
```