Indexes Best Practices (II)
More T-SQL
Control-Of-Flow Language

S6

Indexes Best Practices (II)

Indexed Views

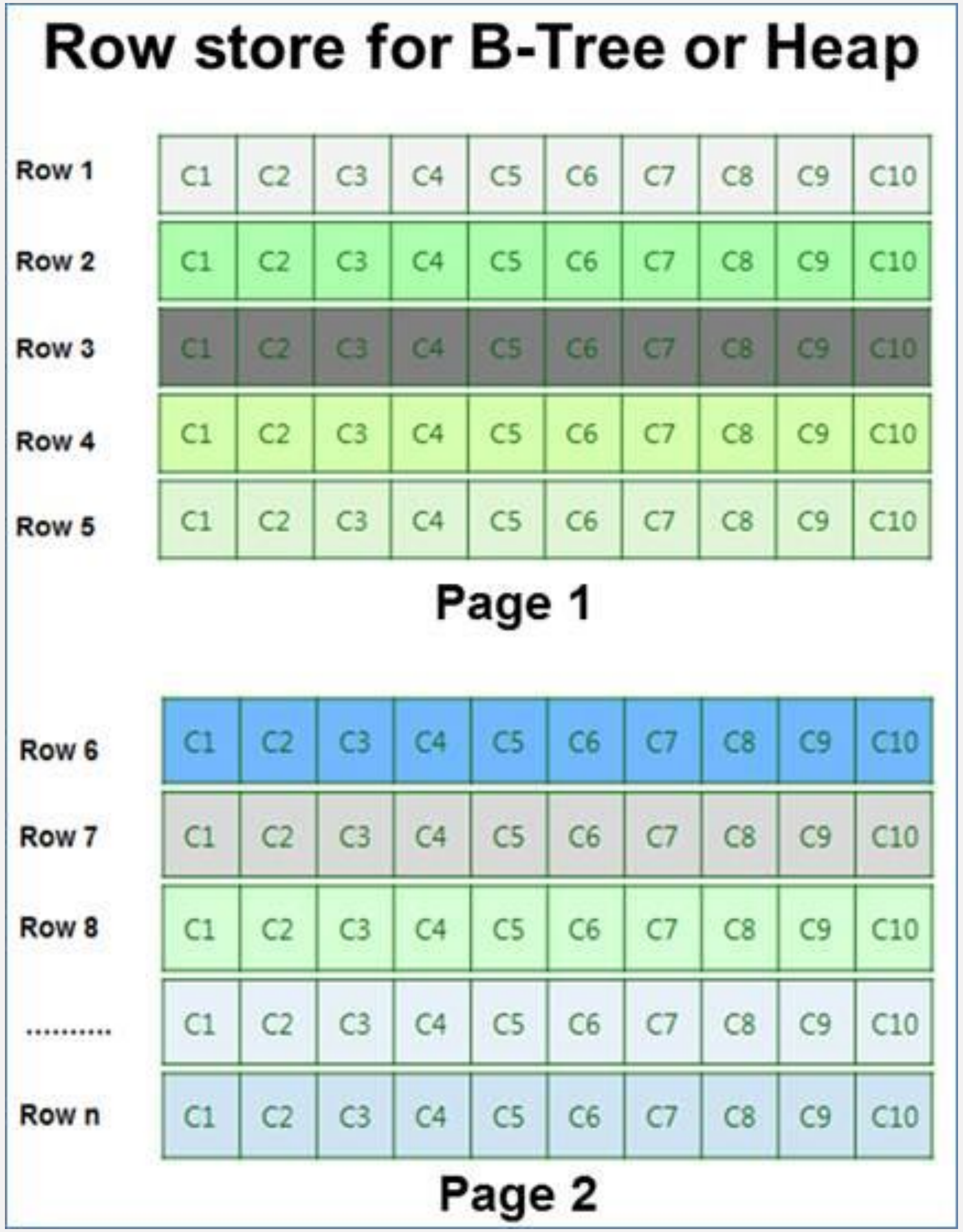
SET options	Required value	Default server value
ANSI_NULLS	ON	ON
ANSI_PADDING	ON	ON
ANSI_WARNINGS	ON	ON
ARITHABORT	ON	ON
CONCAT_NULL_YIELDS_NULL	ON	ON
NUMERIC_ROUNDABORT	OFF	OFF
QUOTED_IDENTIFIER	ON	ON

Indexed Views Restrictions

- SELECT statement cannot reference other views
- All functions must be deterministic
- AVG, MIN, MAX, STDEV, STDEVP, VAR and VARP are not allowed
- The index must be both clustered and unique
- SELECT statement must not contain subqueries, outer joins, EXCEPT, INTERSECT, TOP, UNION, ORDER BY, DISTINCT etc

Columnstore Indexes

- Groups and stores data for each column and then joins all the columns to complete the whole index
- Suited for warehouses (read only tables)
- Up to 10x query performance (vs. traditional row-oriented storage)
- Up to 10x data compression over the uncompressed data size
- The same table can have both row store index and column store index, The *Query Optimizer* will decide when to use the column store index and when to use other types of indexes



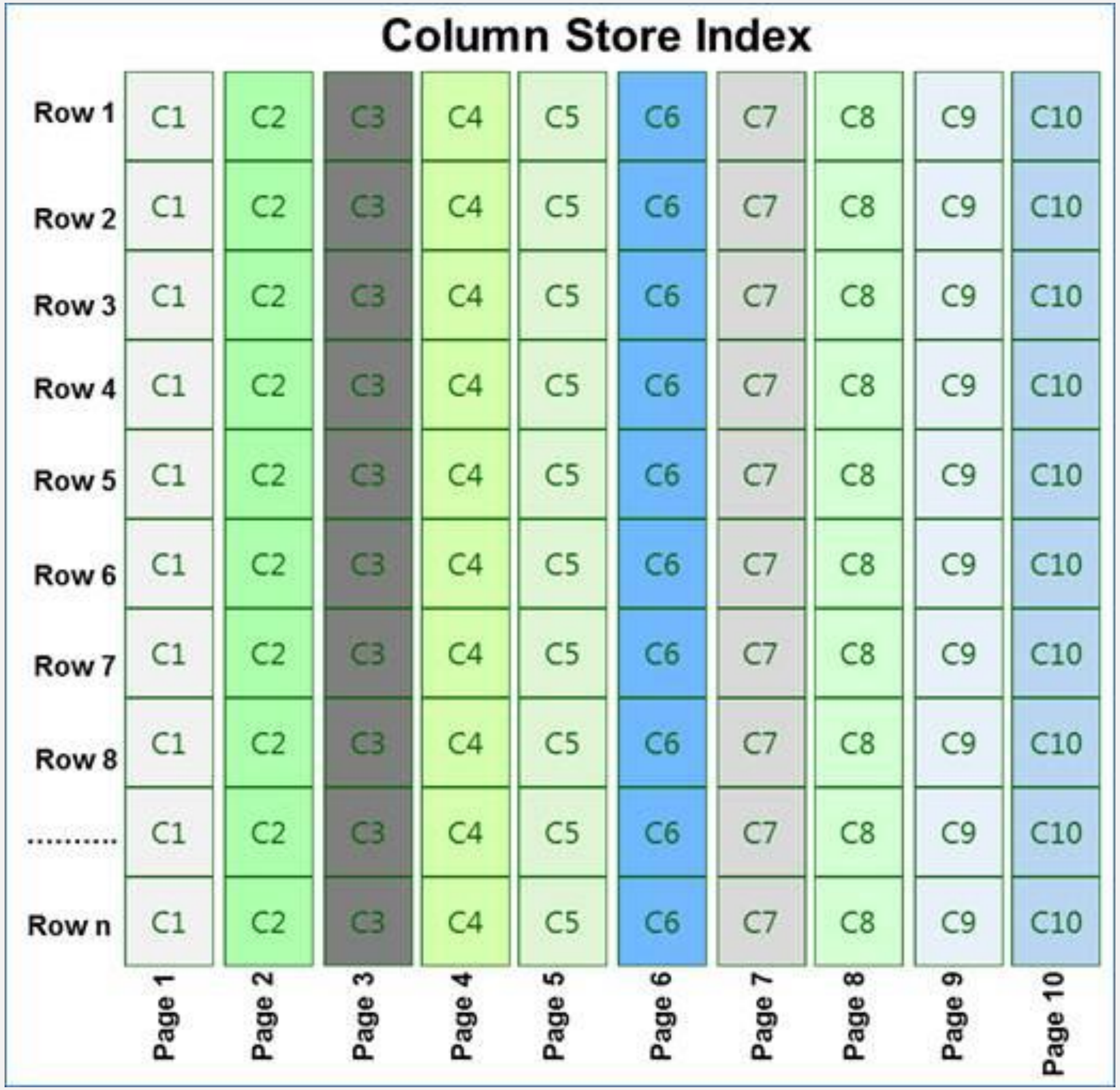
Row store for B-Tree or Heap

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

Page 1

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 2



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

Hard and fast rules for indexing

- Each table should have a clustered index that is (ideally) small, selective, ever increasing, and static. (a table without a clustered index is called a *heap*.)
- Implement nonclustered indexes on foreign key relationships
- Implement nonclustered indexes on columns that are frequently used in WHERE clauses.
- Do not implement single-column indexes on every column in a table. This will cause high overhead.
- In multi-column indexes, list the most selective (nearest to unique) first in the column list.
- For most often-used queries create covering nonclustered index.

Fragmentation

- *Internal Fragmentation*: records are stored non-contiguously inside the page. Internal fragmentation occurs if there is unused space between records in a page. The fullness of each page can vary over time. This unused space causes poor cache utilization and more I/O, which ultimately leads to poor query performance.

Fragmentation

- *External Fragmentation*: When on disk, the physical storage of pages and extents is not contiguous. When the extents of a table are not physically stored contiguously on disk, switching from one extent to another causes higher disk rotations.

Fragmentation

- *Logical Fragmentation*: Every index page is linked with previous and next page in the logical order of column data. Because of Page Split, the pages turn into *out-of-order* pages.
- An *out-of-order* page is a page for which the next physical page allocated to the index is not the page pointed to by the next-pag*e* pointer in the current leaf page.

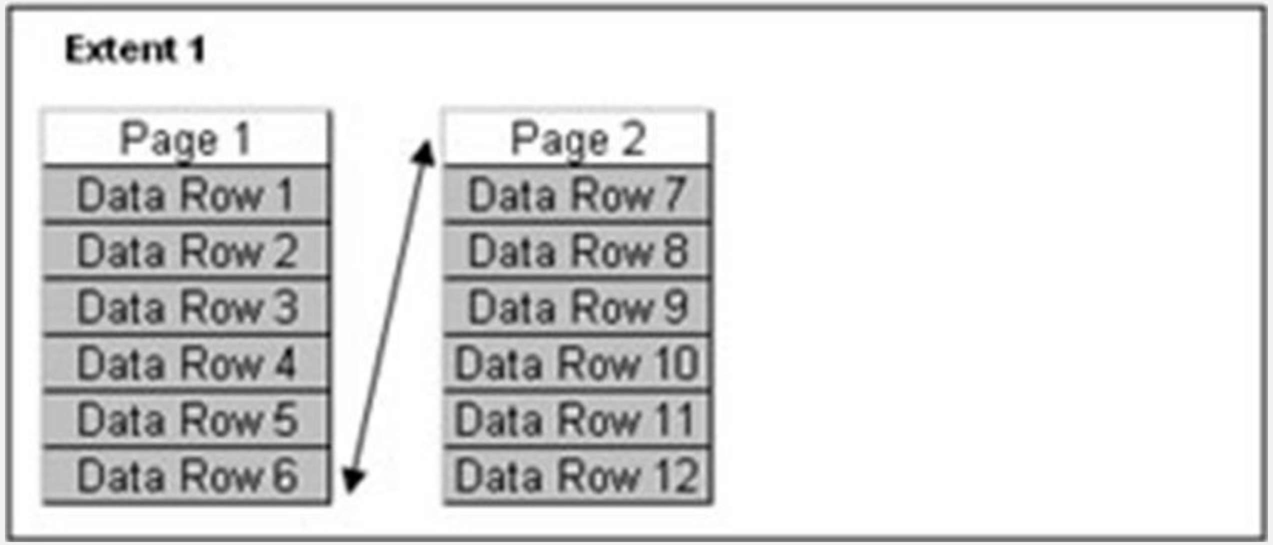
Page read requests: 2

Extent switches: 0

Disk space used by table: 16 KB

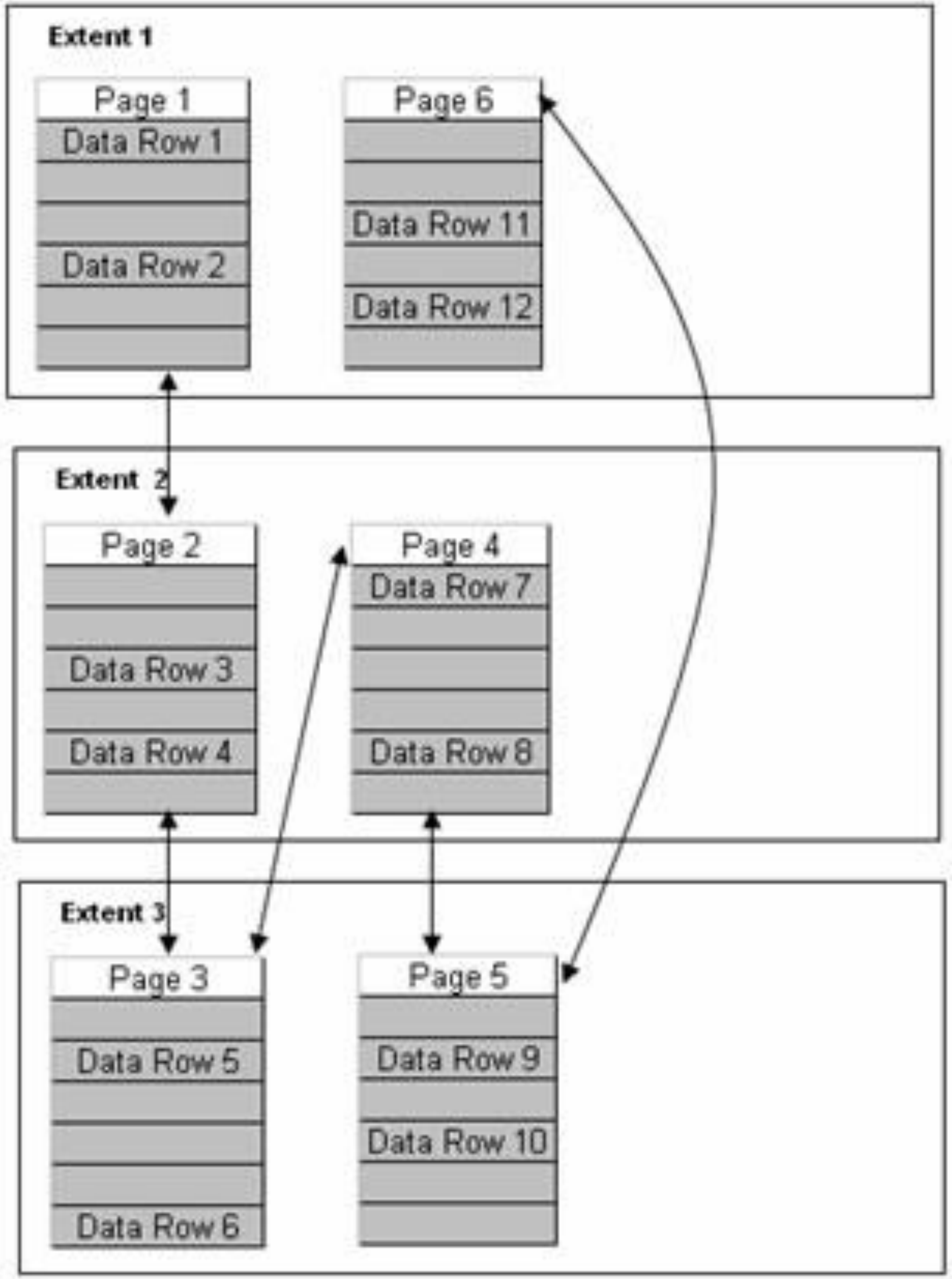
avg_fragmentation_in_percent: 0

avg_page_space_used_in_percent: 100



Extent 1

Page 1	Page 2
Data Row 1	Data Row 7
Data Row 2	Data Row 8
Data Row 3	Data Row 9
Data Row 4	Data Row 10
Data Row 5	Data Row 11
Data Row 6	Data Row 12



Page read requests: 6

Extent switches: 5

Disk space used by table: 48 KB

avg_fragmentation_in_percent > 80

avg_page_space_used_in_percent: 33

Fragmentation

- *sys.dm_db_index_physical_stats*
  - **avg_fragmentation_in_percent**: This is a percentage value that represents external fragmentation.
  - **avg_page_space_used_in_percent:** This is an average percentage use of pages that represents to internal fragmentation.

- **Reducing Fragmentation in a Heap:**
  - To reduce the fragmentation of a heap, create a clustered index on the table.
  - Creating the clustered index: rearrange the records in an order, and then place the pages contiguously on disk.

Fragmentation

Reducing Fragmentation in a Index:

- If avg_fragmentation_in_percent > 5% and < 30%, then use ALTER INDEX REORGANIZE:
  - reorder the leaf level pages of the index in a logical order.
- If avg_fragmentation_in_percent > 30%, then use ALTER INDEX REBUILD:
  - replacement for DBCC DBREINDEX to rebuild the index online or offline. In such case, we can also use the drop and re-create index method.
- Drop and re-create the clustered index:
  - Re-creating a clustered index redistributes the data and results in full data pages. The level of fullness can be configured by using the FILLFACTOR option in CREATE INDEX.

More T-SQL
Control-Of-Flow Language

Control-of-Flow Language

BEGIN…END

RETURN

BREAK

THROW

CONTINUE

TRY…CATCH

GOTO label

WAITFOR

IF…ELSE

WHILE

RETURN

RETURN [integer_expression]

- exits unconditionally from a query or procedure
- returning from a procedure
- returning status codes
  - stored procs return 0 (success), or
  - a nonzero value (failure)

RETURN

```
CREATE PROCEDURE checkstate @param varchar(11)
AS
 IF @param= 'WA'
  RETURN 1
 ELSE
  RETURN 2;
GO
------------------------------
DECLARE @return_status int;
EXEC @return_status = checkstate ‘AK';
GO
```

WHILE

WHILE Boolean_expression

{ sql_statement | statement_block | BREAK | CONTINUE }

- sets a condition for the repeated execution of an SQL statement or statement block

BREAK

- exits the innermost loop in a WHILE statement or an IF…ELSE statement inside a WHILE loop.

CONTINUE

- restarts a WHILE loop; any statements after the CONTINUE keyword are ignored

GOTO

- alters the flow of execution to a label

```
Label:
GOTO Label
```

WAITFOR

```
WAITFOR { DELAY 'time_to_pass' |
          TIME 'time_to_execute' |
          [ ( receive_statement ) |
          ( get_conversation_group_statement ) ]
          [ , TIMEOUT timeout ] }
```

- blocks the execution of a batch, stored procedure, or transaction

WAITFOR

- execution continues at 08:35

```
WAITFOR TIME ‘08:35';
```

- execution continues after 2 hours

```
WAITFOR DELAY '02:00';
```

- if the server is busy → the counter does not start immediately → the delay may be longer than specified.

THROW

```
THROW [
        { error_number | @local_variable},
        { message | @local_variable},
        { state | @local_variable } ] [ ; ]
```

- raises an exception and transfers execution to a CATCH block of a TRY…CATCH construct
- the exception severity is always set to 16.

```
THROW 51000, ‘Record does not exist', 1;
```

TRY … CATCH

```
BEGIN TRY
        { sql_statement | statement_block }
END TRY
BEGIN CATCH
        [ { sql_statement | statement_block } ]
END CATCH [ ; ]
```

- implements error handling for Transact-SQL
- catches all execution errors that have a severity >10 that do not close the database connection

TRY … CATCH

- ERROR_NUMBER() returns the error number
- ERROR_SEVERITY() returns the severity
- ERROR_STATE() returns the error state number
- ERROR_PROCEDURE() returns the name of the stored procedure/trigger where the error occurred
- ERROR_LINE() returns the line number that caused the error
- ERROR_MESSAGE() returns the error message

Error messages

- error number
    - integer value between 1 and 49999
    - custom error messages: 50001…
- error severity
    - 26 severity levels
    - error with severity level ≥ 16 are logged automatically
    - error with severity level between 20 and 25 are fatal and the connection is terminated
- error message: up to 255 chars