Incremental Reorganization:

Adaptive Server 15.7 SP100 includes a new parameter—defrag—for the reorg command, which lets you schedule and resume reorganization, also allowing concurrent reads or writes on the data being reorganized.

**reorg defrag** locks each row or page, as per the locking scheme of the object, and begins and commits a transaction for every data chunk processed. The data reorganization space requirement does not exceed the size of an allocation unit (256 data pages). Index updates for the reorganized data do not consume extra space. For every data partition undergoing incremental reorganization, a row is stored in sysattributes.

- **Running reorg defrag**
  reorg defrag executes multiple reorganization transactions one right after the other, compared with traditional reorg, which reorganizes the entire table data in a single transaction. You can specify a time interval during which it reorganizes the data. This allows other processes to run concurrently without affecting reorganization.

- **Checking the Reorganization Status**
  Use information from sp_helpdefrag or the defrag_status built-in function to decide if reorganization must be resumed on a table or if it should start from the beginning.

- **Clearing reorg defrag Information from sysattributes**
  For every data partition undergoing incremental reorganization, a row is stored in sysattributes. Use the dbcc command zapdefraginfo to delete this information from sysattributes before performing a downgrade.

- **Logging Behavior**
  Default incremental reorganization logging includes the extent allocation, extent version update, index row deletion, index row insertion, extent deallocation, and scan position update, in that order.

- **Restrictions**
  There are several restrictions that apply to reorg defrag.

**Running reorg defrag**

**reorg defrag** executes multiple reorganization transactions one right after the other, compared with traditional reorg, which reorganizes the entire table data in a single transaction. You can specify a time interval during which it reorganizes the data. This allows other processes to run concurrently without affecting reorganization.

Syntax:

```
reorg defrag [partition {partition_list}]
[with {time = hh:mm} resume | skip_compact_extents [= pct_value]]
```

where:

- **defrag** – reorganizes each partition list or partition in the table while allowing concurrent reads or writes on the data being reorganized.
- **partition** – is the subset of the table that shares the same column definitions, constraints, and defaults as the table.
- **partition_list** – is the list of partition names.
- **time** – reorganizes the table or the list of partitions for the specified interval of time. hh is the number of hours and has no limit, and mm is the number of minutes 0–59.
- **resume** – resumes table reorganization from the end of the last reorganized data page invoked from the previous reorg defrag, resume continues until the entire table or list of partitions is reorganized, processing only those partitions that are currently either unvisited or partially reorganized. Running `time = hh:mm` with `resume` indicates that you are running reorganization from the previous position of reorganization and running it only for the specified interval of time.
- **skip_compact_extents** – skips compact extents. The compactness of an extent is measured as the percentage occupancy in that extent.
- **pct_value** – is the compactness of an extent measured as the percentage occupancy in that extent with a value of 1–100.

Compactness = (Total space occupied in an extent / Total space in an extent) x 100

If **skip_compact_extents** is used, all the extents with compactness greater than or equal to the threshold occupancy percent value specified would be skipped for reorganization. If no threshold percent value is specified, the default percent value is 80%.

Examples:

In the following examples, 'partition list' is the list of data partitions specified in the command or if none specified, the list of all existing data partitions in the table.

- Example 1 reorganizes the partition list, starting from the beginning of each partition and continuing until its end:

  `reorg defrag salesdetail [partition (seg1 [,seg2[, seg3]])]

  `Example 2 reorganizes the partition list, starting from the beginning of each partition and continuing until the specified time interval is reached.

  `reorg defrag salesdetail [partition (seg1 [,seg2[, seg3]]) with (time = 0:00)]

  `Example 3 reorganizes the partition list, starting from the beginning of each partition and continuing until the specified time interval is reached.

  `reorg defrag salesdetail [partition (seg1 [,seg2[, seg3]]) with (time = 0:00)] resume

  `Example 4 reorganizes the partition list, starting from the beginning of each partition and continuing until the specified time interval is reached.

  `reorg defrag salesdetail [partition (seg1 [,seg2[, seg3]]) with (time = 0:00) resume]

  `Example 5 reorganizes the partition list, starting from the beginning of each partition and continuing until the specified time interval is reached.

  `reorg defrag salesdetail [partition (seg1 [,seg2[, seg3]]) with (time = 0:00) resume]

  `Example 6 reorganizes the partition list, starting from the beginning of each partition and continuing until the specified time interval is reached.

  `reorg defrag salesdetail [partition (seg1 [,seg2[, seg3]]) with (time = 0:00) resume]

  `Example 7 reorganizes the partition list, starting from the beginning of each partition and continuing until the specified time interval is reached.

  `reorg defrag salesdetail [partition (seg1 [,seg2[, seg3]]) with (time = 0:00) resume]

  `Example 8 reorganizes the partition list, starting from the beginning of each partition and continuing until the specified time interval is reached.
• reorg defrag salesdetail [partition
{seg1 [,seg2[,seg3]]}] with time = 01:20

Example 3 restarts reorganization for each partition in the partition list that is either unvisited or partially reorganized, from where the previous invocation of reorganization has stopped, and if no point for resuming is available, reorganization starts from the beginning of the partition. Reorganization continues until the end of each partition.
• reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]

with resume

Example 4 reorganizes the partition list, starting from the beginning of the partition and continuing until its end, skipping the extents with more than the specified occupancy threshold. If a percentage is not specified, extents exceeding 80% occupied are considered compact, and skipped.
• reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]

with skip_compact_extents [ = <1-100>]

Example 5 reorganizes each partition in the partition list that is either unvisited or partially reorganized, starting from the point where the previous invocation of reorganization stopped. Reorganization continues until the end of each partition. Reorganization is considered completed if no point is available for resuming, reorganization starts from the beginning of the partition.
• reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]

with time = 01:20, resume

Example 6 reorganizes the partition list, starting from the beginning of the partition and continuing until the specified interval of time completes, skipping each extent that qualifies to be compact as per the pct_value specified for skip_compact_extents. If a percentage is not specified, extents exceeding 80% occupied are considered compact, and skipped.
• reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]

with time = 01:20, skip_compact_extents [ = <1-100>]

Example 7 reorganizes each partition in the partition list that is either unvisited or partially reorganized, starting from the point where the previous invocation of reorganization stopped. If no point is available for resuming, reorganization starts from the beginning of the partition. Reorganization continues until the end of each partition. It skips extents that qualify to be compact as per the pct_value provided for skip_compact_extents. If the pct_value is not provided, extents exceeding 80% occupied are considered compact and hence skipped.
• reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]

with resume, skip_compact_extents [ = <1-100>]

Example 8 reorganizes each partition in the partition list that is either unvisited or partially reorganized, starting from the point where the previous invocation of reorganization has stopped. If no point is available for resuming, reorganization starts from the beginning of the partition. Reorganization continues until the specified time interval completes. It skips extents that qualify to be compact as per the pct_value specified for skip_compact_extents. If a percentage is not specified, extents exceeding 80% occupied are considered compact and hence skipped.
• reorg defrag salesdetail [partition {seg1 [,seg2[,seg3]]}]

with time = 01:20, resume, skip_compact_extents [ = <1-100>]

For more information about reorg, see the Reference Manual: Commands.

For every 10% of a partition processed, this information appears:

Examined n allocation unit(s). Processed x pages out of y data pages. z% completed, resulting in p% space compaction.

At the end of processing each partition, the time elapsed in the current invocation is printed as:

Elapsed time 1m : 56s : 623ms.

Checking the Reorganization Status

Use information from sp_helpdefrag or the defrag_status built-in function to decide if reorganization must be resumed on a table or if it should start from the beginning.

Additionally, sp_helpdefrag and defrag_status() can also be used to track the status of reorganization of the table when reorg defrag is in progress.

sp_helpdefrag and defrag_status give the percentage of the table or partition reorganized and whether the reorganization is in progress on the table or in an intermittent state to resume.

You must execute sp_helpdefrag in the context of the database where the table resides. If you do not specify a table name, output includes all of
the tables in the database which have valid information about defragmentation status.

The syntax for **sp_helpdefrag** is:

```
sp_helpdefrag [table_name][, partition_name]
```

The syntax for **defrag_status** is:

```
defrag_status( dbid, objid [, ptnid | -1 [, "tag"] ])
```

### Clearing reorg defrag Information from sysattributes

For every data partition undergoing incremental reorganization, a row is stored in `sysattributes`. Use the `dbcc` command `zapdefraginfo` to delete this information from `sysattributes` before performing a downgrade.

Also in a running server, if the rows with defragmentation information for a specific object are accidentally lost from `sysattributes` due to unknown reasons, use `dbcc zapdefraginfo` to reset the extent version information for the specific object so that a later `reorg defrag` will not fail to consider all the extents of the object.

The **System Changes** section in this guide provides syntax information.

### Logging Behavior

Default incremental reorganization logging includes the extent allocation, extent version update, index row deletion, index row insertion, extent deallocation, and scan position update, in that order.

This logging sequence ensures complete recoverability of the reorganization. However, this sequence may not enforce the `dump tran sequence`.

Turn on either **full logging for reorg rebuild** or the general **full logging for all** options in the existing database to ensure that the incremental reorganization enforces the `dump tran sequence`, an extra log of the page image is required for every destination data page created by the reorganization.

The default logging behavior consumes the following log space by **reorg defrag**:

- Number of extent allocation log records –
  - Max value: `(32 / ‘number of preallocated extents’).`
  - Min value: 1.
- Number of extent version update log records – Number of the destination extents allocated in this transaction (Maximum 32).
- Total number of index delete/insert log records – `2 x (number of indexes on the table) x (number of data rows de-fragmented in this transaction)`.
- Number of extent deallocation log records –1 (for all the extents deallocated in this transaction).
- Number of page deallocation log records – Number of unused destination pages in the transaction.
- Number of scan position update log records – 1 (for the last scan position of this transaction).
- Extra logging to achieve full logging behavior:
  - Number of page image log records – Number of used destination pages in the transaction (Maximum 255).

### Restrictions

There are several restrictions that apply to **reorg defrag**.

- Tables must be data-only-locking (DOL) and have at least one index to use **reorg defrag**. Sybase recommends that the table has a unique index to efficiently accommodate the movement of the forwarded locations during reorganization.
- You can run only one instance of **reorg defrag** at a time on a table.
- When **reorg defrag** is in progress, DDLs cannot run on the table, for example add/drop partition, create/drop index, add/modify/drop columns. Also, other existing **reorg** subcommands like `rebuild/reclaim/compact/forwarded rows` cannot run when **reorg defrag** is in progress.

### Creating Indexes without Blocking Access to Data:

Adaptive Server 15.7 SP100 includes the `create index ... online` parameter, which lets you create indexes without blocking access to the data you are indexing.

The syntax is:

```
create [unique] [clustered | nonclustered] index index_name
    on database.]owner.]table_name
    [with {...
    online,
    ...}]
```
For example, to create the index `pub_dates_ix` on the `titles` table with the `online` parameter, use:

```sql
create index pub_dates_ix
on titles (pub_id asc, pubdate desc)
with online
```

Except for the `sorted_data` parameter, Adaptive Server processes other `create index` parameters the same way, both with or without the `online` parameter. For example, if you include the `reservepagegap` parameter with the `online` parameter, Adaptive Server reserves the pages while creating the new data layer. However, if you create the index using the `sorted_data` option, Adaptive Server creates the index on the existing data layer.

**Restrictions:**

- User tables must include a unique index to use the `create clustered index ... online` command (creating nonclustered indexes does not have this restriction).
- You can run `create index ... online` with a `pll sort` only on round robin partitioned tables.
- If you issue an `insert`, `delete`, `update`, or `select` command while `create index ... online` or `reorg ... online` are in the logical synchronization blocking phase:
  - The `insert`, `delete`, `update`, or `select` commands may wait and execute after `create index ... online` or `reorg ... online` are finished.
  - Adaptive Server may issue error message 8233.
- You cannot:
  - Run `dbcc` commands and utility commands, such as `reog rebuild`, on the same table while you are simultaneously running `create index ... online`.
  - Run more than one iteration of `create index ... online` simultaneously.
  - Perform a `dump transaction` after running `create index ... online`. Instead, you can:
    - Run `create index ... online`, then dump the database, or
    - Run a blocking `create index`, then issue `dump transaction`.
  - Run `create index ... online` within a multistatement transaction.
  - Create a functional index using the `online` parameter.