# *Versant Vorkout Usage Guide*

**Release 7.0.1.4**

VERSANT

**Versant History, Innovating for Excellence**

In 1988, Versant's visionaries began building solutions based on a highly scalable and distributed object-oriented architecture and a patented caching algorithm that proved to be prescient.

Versant's initial flagship product, the Versant Object Database Management System (ODBMS), was viewed by the industry as the one truly enterprise-scalable object database.

Leading telecommunications, financial services, defense and transportation companies have all depended on Versant to solve some of the most complex data management applications in the world. Applications such as fraud detection, risk analysis, simulation, yield management and real-time data collection and analysis have benefited from Versant's unique object-oriented architecture.

For more Information please visit **www.versant.com**

**Versant US**

Versant Corporation

255 Shoreline Drive, Suite 450, Redwood City, CA 94065

Ph +1 650-232-2400, Fx +1 650-232-2401

**Versant Europe**

Versant GmbH

Wiesenkamp 22b, 22359 Hamburg, Germany

Ph +49.40.60990-0, Fx +49.40.60990-113

# Table of Contents

# *Versant Vorkout*

This usage guide gives a detailed explanation on the Versant's Online Database Reorganization Tool - Vorkout.

The following are covered in detail:

- Introduction
- Usage Notes
- Benefit of Using Vorkout
- Vorkout with Enhanced dbtool Utility
- Vorkout Restrictions

# INTRODUCTION

Versant's Online Database Reorganization Tool is called Vorkout. It is used with the help of `vcompactdb` utility.

When a database is freshly populated, the objects are efficiently packed on a disk, in storage units called pages.

A compact database enjoys performance advantages caused by better allocation of data on the disk. A compact database also utilizes its backend cache to the fullest extent because each page in the cache contains the maximum possible number of objects.

Over time as objects grow or are deleted, empty holes are created in the tightly packed database resulting in fragmentation of data segments. Thus, performance starts degrading and disk usage is also increased.

It is possible to compact a DB offline by using `vcopyDB` or `vstream,` see the Versant Database Administration Manual for more information.

In a high availability system, offline compaction is not an option. Versant has addressed this issue by introducing the Versant Online Database Reorganizer (Versant Vorkout).

Versant Vorkout reorganizes the data online for reduced fragmentation and restored performance. Besides compacting database objects, Versant Vorkout can also compact the system tables, also known as the AT table.

The enhanced `dbtool` utility provides the user the ability to analyze a database for wasted space. It thus helps in deciding whether running Vorkout is required or not.

## VORKOUT USAGE

```
vcompactdb
    -database <dbname>
            [-class <class name> | -all]
            [-batchsize <number of objects>]
            [-systables]
            [-old]
            [-nowait]
            [-pollonly]
            [-cancel]
            -version
```

**Options are:**

**`-class <class name>`**

Indicates that the reorganization should be executed on data segments for the non-system class specified by `<class name>`. Specifying an invalid class name will result in an error being thrown.

**`-all`**

Indicates that the reorganization should be performed on all data segments for all the non-system classes in the database.

**`-batchsize <number of objects>`**

Indicates that the reorganization should be performed by reorganizing specified number of objects at a time. This option can be used to tune the memory requirements and execution speed of `Vorkout`. The default batch size is 1000.

**`-systables`**

Indicates that the reorganization should be executed on System Tables. This option can be used in conjunction with the `-class` option.

For example:

```
vcompactdb -database <database name> -class Hello -systables
```

The system table is a hash table that is used by the database server to lookup physical object locations within the database volumes given an object’s LOID (logical object identifier) as the key. It is implemented using extensible hashing which is a form of dynamic hashing scheme. The LOID values are the keys for the hash table. Due to the random nature of insertions and deletions into this hash table, the table gets fragmented over time. The fill grade of the pages used for the hash table entries might go significantly below the average 50%, which can normally be reached with the hashing algorithm. This fragmented table can lead to increased memory consumption, slower lookups, and unnecessary space usage in the database.

The first time the system tables are compacted on a database that has been in use for a long time, the process may take a while, and result in a lot of freed pages.

But subsequent runs will be fast compared to the time it needs to compact the pages used for the objects in the database. Therefore, we recommend running the system table compaction regularly whenever Vorkout is run.

You may use the `dbtool -AT -analyze` option to get an indication of the current System Tables fill grade.

**Please refer to "Vorkout with Enhanced dbtool Utility" on page 12 for more information.**

### `-old`

Uses the old online de-fragmentation algorithm for backward compatibility.

### `-nowait [-systables] -class classname`

Starts a compaction of the given class, without polling any progress information. The compaction will be started on the server in the background. System tables compaction can be started additionally or alone, without a class.

### `-pollonly [-systables] -class classname`

Prints the progress of a running compaction of the given class. System tables compaction can be polled additionally or alone, without a class.

### `-cancel [-systables] -class classname`

Cancels a running compaction of the given class. System tables compaction can be cancelled additionally or alone, without a class.

### `-version`

Gives Vorkout version information.

**NOTE:-** If `vcompactdb` is executed without specifying `-class <class name>` or `-all` or `-systables` option, then ONLY list of classes in the database is displayed on the screen and no compaction will take place.

## Benefits

Fragmentation affects reads, writes, and un-optimized queries quite dramatically. Using Vorkout regularly enables your application to achieve better performance.

### Effect of fragmentation on performance

Fragmented pages affect performance for just about every database operation. For example, assume you have 50% fragmentation (i.e., only 50% of your disk pages are occupied with objects and 50% are empty). The effect of this type of fragmentation is as follows:

- The server cache is only 50% as efficient as it is with zero fragmentation, because the pages stored in the cache hold only half as many objects as they hold with zero fragmentation. This means that reads read from disk versus shared memory on the server twice as often. The result is a tremendous slow down of read performance.
- Write performance suffers as well since each page written to the physical log and the system files contains only half as many objects as  with zero fragmentation. This would not affect writes to the logical log as this log is not page based. The net slow-down in write performance is approximately a 35%.
- Un-optimized queries take up to twice as long because twice the number of pages are fetched from the database volumes to search through the same number of objects.

### Improved free Space re-use

As a result of fragmentation, it may happen that although you have plenty of free space, none of the spaces available is big enough for new objects to be placed.

While de-fragmenting, Vorkout will naturally free up larger lumps of space. The actual physical size of the database will not be reduced, but there will be more, larger lumps of space available for re-use when new objects are created in the database.

### When to run a de-fragmentation?

The following two commands determine the fragmentation level of the database quickly and efficiently (this functionality is available directly from Versant Vitness Graphical User Interface).

**To analyze the data pages for fragmentation**

```
dbtool -space -class -all -sample db
```

The `-sample` option looks at 1000(default) random pages taken as representatives for all the pages used by that class. The sample size can be changed using sub-option `-size` to get a more accurate estimate. However, 1000 should be an adequate representative.

**Typical output:**

```
VERSANT Utility DBTOOL Version X.X.X.X
Copyright (c) 1989-2006 VERSANT Corporation

Class: DBCompoundMsg
  bytes not in use = 129777664
  percentage empty = 37%

Class: DBExpectedMsg
  bytes not in use = 144556032
  percentage empty = 39%

Class: DBActualRecipient
  bytes not in use = 135561216
  percentage empty = 37%

Class: DBInternalAlarm
Empty class

Class: DBCell
  bytes not in use = 62865408
  percentage empty = 13%

Class: DBTCILayer
  bytes not in use = 32768
  percentage empty = 18%

Class: DBLogRecord
  bytes not in use = 0
  percentage empty = 0%
```

**Recommendation:**

The higher the percentage empty is and the higher the bytes not in use are, the more you will gain by defragmenting a class. As a rule it is recommended to run a de-fragmentation of a class, if the percentage empty is greater than 20% and the number of bytes not in use are greater than 100000.

Running a de-fragmentation depends on the use of the database specifically, how quickly it gets defragmented.

If you find that over time your classes get de-fragmented quickly, you could decide to first run a de-fragmentation if the percentage empty is greater than 30%.

In the above example, the Classes: `DBCompoundMsg`, `DBExpectedMsg`, `DBActualRecipient` will greatly benefit from a de-fragmentation.

**To analyze the AT (loid2poid table) for fragmentation**

```
dbtool -AT -analyze

Typical output :

VERSANT Utility DBTOOL Version X.X.X.X
Copyright (c) 1989-2006 VERSANT Corporation


** Analyzing AT **

6 AT table(s) are less than 50% filled. The space occupied by these
tables is 10354835456 bytes. The space occupied by the complete AT is
10355146752 bytes.
```

**Recommendation:**

A System Tables de-fragmentation should be run when at least one of the System Tables is filled below 50% capacity.

In the example above, 6 system tables are less than 50 % full so you will greatly benefit from a System Tables de-fragmentation. In theory, you could expect to gain back more than 50% of the 10354835456 bytes currently occupied, not to forget the performance improvement.

## How often should one run a de-fragmentation?

It really depends on the use of your database. Running the dbtool analysis at regular intervals will give you an indication on how quickly your database gets de-fragmented.

Please note that the first time a database is compacted, the process will take longer than the subsequent runs.

## Example

```
$ vcompactdb -database test
Compacting DB test
4 classes to compact

Please specify the class for which you would like to defragment the
segment using the -class <class name> option or specify all classes
using the -all option.

Choose from the following classes:
c0      c1      c2      c3

$ vcompactdb -database test -all -systables
Compacting DB test
4 classes to compact
115488 KB free before compaction
vcompactdb for class c0 has 12.8358 percent (state is running)
vcompactdb for class c0 has 89.2537 percent (state is running)
vcompactdb for class c0 has 100 percent (state is complete)
has finished with error code 0

vcompactdb for class c1 has 25 percent (state is running)
vcompactdb for class c1 has 75 percent (state is running)
vcompactdb for class c1 has 100 percent (state is complete)
has finished with error code 0
vcompactdb for class c2 has 15 percent (state is running)
vcompactdb for class c2 has 50 percent (state is running)
vcompactdb for class c2 has 100 percent (state is complete)
has finished with error code 0

vcompactdb for class c3 has 13 percent (state is running)
vcompactdb for class c3 has 29 percent (state is running)
vcompactdb for class c3 has 100 percent (state is complete)
has finished with error code 0
Defragmenting system tables . . .
```

```
0 percent (state is scheduled)
100 percent (state is complete)
has finished with error code 0
Done.
118240 KB free after compaction. 2752 KB freed up
```

## VORKOUT WITH ENHANCED DBTOOL UTILITY

With the enhanced `dbtool` utility, the DBA can check the fragmentation of the database before initiating reorganization of the database.

To determine the degree of fragmentation of a data segment, every page occupied by the segment needs to be inspected.

This is done with the `-space` option of `dbtool` utility. Its usage is as follows:

```
dbtool -space {-segment { -all | -name <segment name> [-name <segment
name> ...]}
              |
               -class { -all | -name <class name> [-name <class name>
...]}
              }
              [-fillfactor <percentage>]
              [-sample][-size <sample size>]
              [-verbose | -extraverbose]
              [-highlevel]
<dbname>
```

**Options are:**

**-space**

Determine the level of fragmentation of data segments by inspecting the pages in the database <dbname> and appropriately advise the DBA on reorganization of data.

Specify `-segment` or `-class` sub-option (mandatory).

**`-segment {-all | -name <segment name> [-name <segment name>...]}`**

Determine the fragmentation information for segments of the database <dbname>. If the `-all` option is specified then the fragmentation information for all the segments will be inspected. To specify a particular segment, use the `-name <segment name>` option. More than one segment name can be provided by adding more "`-name <segment name>`" pairs to the command line. If the segment name does not exist, the error `SM_E_SEG_NOT_FOUND` will be thrown.

**`-class {-all | -name <class name> [-name <class name>...]}`**

Determine the fragmentation information only for the segment in which objects of `class <class name>` reside. If the class name provided does not exist then the error `SM_E_CLS_NOT_FOUND` will be thrown. More than one class name can be provided as follows:

```
dbtool -space -class -name <classname1> -name <classname2> <dbname>
```

**Optional sub-options are:**

**`-fillfactor <percentage>`**

Define the percentage of bytes occupied in a page that will render a page to be considered full. The default percentage is 70%.

**`-sample [-size < sample size >]`**

The -sample option looks at 1000 random pages (can be changed with sub-option -size) taken as representative for all pages used by that class.

**`-verbose`**

Print a graphical representation of pages of segment on the screen alongwith the fragmentation information. Pages that are empty will be marked 'E', partially full pages will be marked 'P' and full pages will be marked 'F' followed with the fragmentation information for the segment/class.

**`-extraverbose`**

This option is an extension of the `-verbose` option where the `PID` of the page will also be printed in the graphical representation of pages followed with the fragmentation information for the data.

**`-highlevel`**

The -highlevel option looks at the internal directory of empty pages for that class, which does not take partially filled pages into account. In verbose mode, a page marked 'A' indicates that it is allocated (not empty) and those marked 'E' are empty.

This option is even quicker than the -sampling option but less accurate.

The -sample and the -highlevel option are of course not 100% accurate. The amount of reported free space with both sub-options differs since not all pages get examined for the available free space left.

**WARNING:-** Since with -class or -segment options all pages of all given classes, or segments are loaded, the work of other database clients might get delayed significantly. That happens in the case their pages still in use get removed from the cache. There is a lot additional work for the disk drive to load all pages as well.

For example, `C:\_VOD\Labs>dbtool -space -class -all db1` will read all pages used by all classes in the databases, which means almost the entire databases on disk.

To avoid this effect, **it is strongly recommended** to use the `-class` or `-segment` option together with the less accurate sub-options, `-sample` or `-highlevel`.

For example, `dbtool -space -class -all -sample db1`.

## System Table defragmentation

It is also possible to determine the level of fragmentation of the System Tables (internally known as the AT tables).

This is done with the `-AT -analyze` option of `dbtool` utility.

Its usage is as follows:

```
dbtool -AT -analyze
```

Analyze the AT (loid2poid table) for fragmentation.

It is recommended to run a System Tables defragmentation when at least one of the System Tables is filled below 50% capacity.

## Example

```
C:\_VOD\Labs>dbtool -space -class -all -sample db
VERSANT Utility DBTOOL Version 7.0.1.4
Copyright (c) 1989-2008 VERSANT Corporation
Class: Company
  bytes not in use = 0
  percentage empty = 0%

Class: Department
  bytes not in use = 49152
  percentage empty = 0%

Class: Employee
  bytes not in use = 65536
  percentage empty = 0%
```

```
C:\_VOD\Labs>dbtool -AT -analyze db1
VERSANT Utility DBTOOL Version 7.0.1.4
Copyright (c) 1989-2008 VERSANT Corporation

** Analyzing AT **

All AT table(s) are filled above 50% capacity. The space occupied by
these tables is 24690688 bytes.
```

## VORKOUT RESTRICTIONS

- Only one instance of `vcompactdb` should work on a database at time.

# Index