# Data Compression in Blackbaud CRM Databases

Len Wyatt
Enterprise Performance Team

## Executive Summary

Compression refers to applying mathematical techniques to make objects take less storage space than they normally require. Most people are familiar with using the ZIP utility to compress documents. This is useful to attach documents in email because they take up less space and are more likely to fit in a recipient's mailbox. SQL Server can apply similar methods to data in its tables and indexes. The data is still in the database, but it's stored in a way that requires less physical space. SQL Server can apply various techniques to compress data.

Compression is probably not applied by default because it requires extra CPU time to compress data when it is stored and to decompress data when it is read. The Microsoft SQL Server team may not have wanted to decide this trade-off for everyone. However, modern servers are rarely limited by CPU resources. The limiting factor in servers is more likely the amount of data that can be sent to and fetched from the disk subsystem (I/O), so making data smaller is usually a good tradeoff. And since compression saves memory on the server by compressing objects in memory as well as on disk, it also reduces the amount of I/O since more information can be retained in memory.

Tests by the Enterprise Performance Team show significant performance benefits from compression in the SQL Server databases that support ***Blackbaud CRM***. Benefits vary based on factors such as the database and workload, but we consistently found space savings. One test even reduced the space that the database used by 44 percent.

To help customers realize the benefits of compression technology, the Performance Team developed a set of T-SQL stored procedures (SPs) that find and compress the appropriate objects in SQL Server databases. We tested these procedures extensively, and they are now available to our customers. The scripts are intended for the database administrator (DBA) staff for self-hosted customers and the Blackbaud SDO team for hosted customers. These groups are the target audience for this paper.

One benefit of data compression is that the technology is applied at the database level, *below* the level of the ***Blackbaud CRM*** application. The application does not “know” about the internal storage mechanism of the database, and compression does not change the logical behavior of the application. This means you can apply compression to any version of ***Blackbaud CRM***.

## Compression in SQL Server

This section summarizes information that is available online. For more information about compression, Microsoft’s Data Compression: Strategy, Capacity Planning and Best Practices article is particularly useful.

SQL Server supports two types of compression in tables and indexes: Row compression and page compression.

Row compression reduces the space to store metadata and stores fixed-length data in variable-length formats. This isn’t just about storing CHAR data as VARCHAR; it even stores numeric data in fewer bits.

Page compression uses the same techniques as row compression, but it also uses prefix compression and dictionary compression to look for repeating patterns in data and store a single copy of each pattern. Page compression can substantially increase the space saved, but it uses additional CPU resources.

For any object in the database, you must decide which type of compression to use. We return to this question in **Choose Compression Types**.

When you apply compression to a database object such as a table or index in SQL Server, you alter the object’s metadata. Statements that access the object do not change in any way; the SQL Server Storage Engine deals with the object appropriately.

Here are some sample statements that apply compression to tables and indexes:

```
ALTER TABLE [dbo].[BATCHSPONSORSHIP]
REBUILD PARTITION = ALL WITH (DATA_COMPRESSION = ROW);
```

```
ALTER TABLE [dbo].[REVENUETRIBUTE]
REBUILD PARTITION = ALL WITH (DATA_COMPRESSION = PAGE);

ALTER INDEX [IX_MKTSEGMENTATION_ID] ON [dbo].[MKTSEGMENTATION]
REBUILD WITH (DATA_COMPRESSION = ROW);

ALTER INDEX [PK_REVENUESCHEDULE] ON [dbo].[REVENUESCHEDULE]
REBUILD WITH (DATA_COMPRESSION = PAGE);
```

When you compress an object, it is rebuilt. The object is re-written in the database in the compressed format, and then the uncompressed object is removed. This operation is logged, and unless you use simple logging, which is not recommended for ***Blackbaud CRM***, the transaction log file grows as you compress objects. The Performance Team's scripts stop compressing objects when the available log file space falls below a specified threshold so that you can perform backups to reclaim the transaction log file space. The goal behind this approach is to avoid growing the log file.

If users need to access the database during compression, you can perform online rebuilds to reduce contention. This increases the amount of work to perform compression, but it avoids exclusive locks on objects.

```
ALTER TABLE [dbo].[BATCHSPONSORSHIP]
REBUILD PARTITION = ALL WITH (DATA_COMPRESSION = PAGE, ONLINE = ON);
```

The Performance Team's scripts include an option to specify online rebuilds. However, you cannot perform online operations on objects that include columns with the data types `text`, `ntext`, `image`, `varchar(max)`, `nvarchar(max)`, `varbinary(max)`, `xml`, or `large CLR`. You must compress objects with these data types without the online option. Because of these data type limitations and additional overhead, we do not recommend online rebuilds for ***Blackbaud CRM*** databases.

## Perform Compression in Blackbaud CRM Databases

In many SQL Server databases, DBAs choose the tables and indexes to compress on a case-by-case basis based on principles like those outlined in Data Compression: Strategy, Capacity Planning and Best Practices. However, a ***Blackbaud CRM*** database likely has thousands of tables and even more indexes, so case-by-case decisions are not practical. You must automate compression.

In addition, the compression process must maintain state information so that you can resume it at any time. This too must be automated.

To meet these needs, the Performance Team created a set of T-SQL stored procedures (SPs) to manage the process. They allow DBAs to expeditiously handle the compression of thousands of objects within the log file constraints of the local system. The SPs are posted on the Blackbaud CRM Data Compression Tools Downloads page of the Blackbaud website. The download explains the SPs, but here is a summary:

### Initial Compression

- Connect to the ***Blackbaud CRM*** database with ***SQL Server Management Studio*** and execute the statements in the file. This creates the SPs.

- Run `usp_CompressionSizeInfo` before and after compression to monitor the space saved. This step is optional.
- Run `usp_PrepareCompression` to create a few helper tables to automate compression.
- Run `usp_UpdateCompressionSettings` to change compression settings if necessary. The most likely settings to change are `LogFileFactor`, `OnlineOperation`, and `CompressionType`.
    - `LogFileFactor` controls the threshold where compression stops because the log file is nearly full. When less than 1/`LogFileFactor` of the log file's space remains, the process stops. The default value is 10, so compression continues until less than 1/10th of the space remains. To reserve more space, you can set this to a lower number.

      ```
      usp_UpdateCompressionSettings @Variable = 'LogFileFactor', @Value = 5
      ```

    - `OnlineOperation` determines whether to perform online rebuilds. By default, this is set to "0" and online rebuilds are not performed. If compression occurs while users are active, you can set `OnlineOperation` to "1" to use online rebuilds to reduce the risk of blocking.

      ```
      usp_UpdateCompressionSettings @Variable = 'OnlineOperation', @Value = 1
      ```

      Keep in mind that even if you set `OnlineOperation` to "1,"online operations are not available for objects that include columns with the `text`, `ntext`, `image`, `varchar(max)`, `nvarchar(max)`, `varbinary(max)`, `xml`, and `large CLR` data types.

    - `CompressionType` controls the type of compression for database objects. For complete details, see Choose Compression Types. By default, this is set to "1" for page compression, but you can also set it to "2" for row compression or "3" to apply heuristics that seek optimal system performance (response time) instead of maximum space savings.

      ```
      usp_UpdateCompressionSettings @Variable = 'CompressionType', @Value = 3
      ```

      At this time, the Performance Team recommends compression type 1.

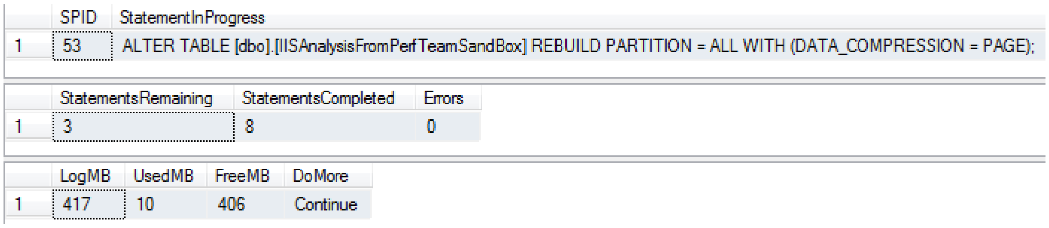
  The remaining settings are `OverwriteTableData`, `MaxUpdateRateForPage`, `MinScanRateForPage`, and `MaxRelOpRateForPage`. These settings are only relevant if you set `CompressionType` to "3," as discussed in Choose Compression Types.

- Run `usp_CreateCompressionQueue` to generate a list of objects in the database and create the T-SQL command to compress each object based on parameters in the `CompressionAdmin.Settings` table.
- Run `usp_RunCompression` to run statements from the compression queue sequentially and to log results in a table of completed statements. Run this as many times as necessary to complete all statements in the compression queue, subject to the transaction log space constraints of your system. Depending on your situation, it might be best to run this each day as part of your daily maintenance procedures, just before backups. Or you might want to crank through it on a

weekend and run log backups between calls to `usp_RunCompression`. If the database is set to simple logging, which is not recommended for ***Blackbaud CRM***, `usp_RunCompression` runs until all statements are executed. For more information, see Log Files and Compression.

The procedure `usp_RunCompression` allows multiple instances to run in parallel without interfering with each other. This speeds up the compression process, but the Performance Team's tests show a system running compression quickly becomes I/O limited. More than two parallel instances are of limited value.

- Run `usp_MonitorCompression` at any time. This SP is for informational purposes. It shows several things of interest.



| | SPID | StatementInProgress |
|---|---|---|
| 1 | 53 | ALTER TABLE [dbo].[IISAnalysisFromPerfTeamSandBox] REBUILD PARTITION = ALL WITH (DATA_COMPRESSION = PAGE); |

| | StatementsRemaining | StatementsCompleted | Errors |
|---|---|---|---|
| 1 | 3 | 8 | 0 |

| | LogMB | UsedMB | FreeMB | DoMore |
|---|---|---|---|---|
| 1 | 417 | 10 | 406 | Continue |

This example shows that an `ALTER TABLE` statement is currently running. Next, it shows that three statements are still in the queue and that eight statements completed with no execution errors. Finally, it shows the total space for the transaction log, how much is used, and how much remains. The **DoMore** column also indicates whether the `usp_RunCompression` procedure will continue to run statements.

If errors occur during compression, `MonitorCompression` also lists them for DBAs to investigate.

- Run `usp_MonitorCompression` to check for errors after the compression process. Errors do not usually occur, but the list can display up to 5. If additional errors occur, you can see the complete list with this statement:

```
select *
from CompressionAdmin.StatementsCompleted
where ReturnStatus is not null or ErrorMessage is not null
```

One possible error is duplicate data in a table that prevents the creation of a unique index. The message for this error is similar to:

```
The CREATE UNIQUE INDEX statement terminated because a duplicate key was
found for the object name 'dbo.MKTSOURCECODEMAP' and the index name '
IX_USR_MKTSOURCECODEMAP_SEGMENTATIONSEGMENTID_SEGMENTATIONTESTSEGMENTID_L
ISTID_WHITEMAILSEGMENTID'. The duplicate key value is (fe4718f0-d84d-
4b07-8fdf-7fd3f27ee1bd, <NULL>, <NULL>, <NULL>).
```

This message indicates a problem with the data in the table or the index definition, not the compression process. In this example, the error occurs with a custom index and not an index in the ***Blackbaud CRM*** base product. Diligent DBAs will likely want to investigate this, but no harm results from leaving the index uncompressed.

- Run `usp_CompressionSizeInfo` before and after compression to monitor the space saved. This step is optional.
- Run `usp_CleanupCompression` to remove all artifacts of the compression SPs from the system, including the log. This step is optional. For ongoing maintenance purposes, we recommend that you leave the tables and procedures in place.

## Log Files and Compression

As noted above, `usp_RunCompression` stops when the available log space falls below a threshold. The goal is to avoid growing the log file because that can have undesirable performance and storage space consequences. This means DBAs need a strategy to deal with log space and the likely need to resume the compression procedure. Here are some strategies to consider, depending on the needs of your site.

**Option 1: Simple Logging.** The simplest and fastest approach is to set the database to use simple logging, perform compression, and then set the database back to full logging. This method includes some risk because the change to simple logging breaks the log chain for the database and prevents you from rolling forward from older backups.  If you use this method, it is vital to perform a full backup before you switch to simple logging and then perform another backup when you return to full logging mode. This method is appropriate if you want to complete compression as quickly as possible during a system down time window. If down time is not an option, use a different method.

**Option 2: Frequent Backups.** For databases managed by SDO, backups occur every 15 minutes by default. Depending on how fast `usp_RunCompression` generates log data, the backup process may free log space faster than `usp_RunCompression` generates it. In that case, `usp_RunCompression` can run nonstop. In one experiment by SDO, however, the backup process could not keep up with a single `usp_RunCompression` process. The good news is that if `usp_RunCompression` stops due to lack of log space, you just run it again to resume the process.

**Option 3: Backups When Compression Stops.** If backups normally occur daily but you want to complete compression in a short window (such as overnight or during the weekend), it is easy to run a T-SQL script to start a backup when `usp_RunCompression` stops, then run `usp_RunCompression` again after the backup finishes. Here is an example that starts backups as often as necessary. You can customize it to your needs (especially your backup strategy).

```
use MyDatabase

declare @database varchar(100) = 'MyDatabase'
declare @backupfile varchar(100) = 'D:\Backups\LogBackup_MyDatabase.bak'
declare @remaining int

set nocount on

select @remaining = count(*) from CompressionAdmin.StatementQueue where SPID is null
print convert(varchar(20), getdate(), 20) + ', Remaining: ' + convert(varchar(20), @remaining)

COMPRESSION1:
```

```
exec usp_RunCompression
exec ('backup log ' + @database + ' to DISK = ''' + @backupfile + ''' with compression')

select @remaining = count(*) from CompressionAdmin.StatementQueue where SPID is null
print convert(varchar(20), getdate(), 20) + ', Remaining: ' + convert(varchar(20), @remaining)
if @remaining > 0 goto COMPRESSION1
```

**Option 4: Daily Backup Cycle.** If your site does backups nightly, you may want to perform compression at night to avoid creating load during the day. In this scenario, you can add a call to usp_`RunCompression` in your daily maintenance scripts *before* the daily backup. Each night, the system compresses objects as long as log space is available, and then it stops until the next night. Over the course of several days, space becomes available – and performance improves – as the system works through all the objects.

## Prepare for Compression

Compression doesn't change any logic in the database, but it requires a lot of physical work. You should test compression to make sure you understand the process and to confirm that nothing goes wrong. All ***Blackbuad CRM*** customers have staging systems in addition to their production systems, and we recommend that you use your staging system for testing.

- Set up the staging system with a recent copy of the production database.
- Familiarize yourself with the performance of key operations on the staging system. Use a stopwatch to time operations that are part of your acceptance criteria for upgrades.
- Compress the database with the scripts described in this paper. The process should take a few hours, although this varies based on your machine and the size of the database.
- Use the stopwatch again to time key operations. If they aren't faster, you still saved a bunch of space. And if they are faster, you are a hero.
- Validate the functioning of maintenance processes and other operational tools.
- After everything checks out, schedule a time to repeat the process on the production system.

## Maintain Compression

***Blackbaud CRM*** is a complex application that dynamically generates new objects in the database over time. As of version 4.0, none of these objects are compressed. Customizations can also create objects in your database that are not compressed. To take advantage of compression for these objects, we recommend that you periodically perform the compression steps. Perhaps on a monthly basis, run `usp_CreateCompressionQueue` and `usp_RunCompression`. After the initial mass conversion to compress objects, periodic re-checks are quick. We also recommend that you perform the compression steps after upgrades because until version 4.0, patches and service packs are not aware of the compression option.

## What to Do with the Saved Space

After you implement compression, it may be tempting to reclaim the saved space on the disks of your server system. Before you do that, please consider two things:

- Databases tend to grow over time, so the saved space is valuable as a buffer against future growth. When you leave the space in place, you create extra time before you need additional space. Of course, it is still wise to monitor space usage as suggested in Manage File Growth in Blackbaud CRM Databases. That article includes a script to monitor the space that files use so that you can plan for growth.
- The way to reclaim the space is to “shrink” the data files. This operation is bad for performance because it severely fragments the storage. For more information, see Why you should not shrink your data files by SQL Server guru Paul Randal. If you absolutely must shrink data files, then follow that action with a process to defragment the indexes. Shrinking and defragmenting are time-consuming operations. They are also logged operations that carry some risk of expanding the transaction log file, which is exactly what we took care to avoid in the compression operations!

## Choose Compression Types

The `usp_CreateCompressionQueue` SP determines the set of objects to compress. It evaluates each table and index in the database to determine whether they are eligible for compression, and then it determines the type of compression to use. The evaluation process is controlled by settings that an experienced DBA can adjust with `usp_UpdateCompressionSettings`. The Performance Team recommends page compression because it saves the most storage space and has excellent performance characteristics. Our tests do not shown a compelling reason to choose the other compression types over page compression.

In `usp_UpdateCompressionSettings`, you set `CompressionType` to “1” to use page compression for all objects. You set `CompressionType` to “2” uses row compression for all objects. And if your site wants to improve performance by focusing on response time rather than maximum space savings, you set `CompressionType` to “3” to use a hybrid compression approach that applies heuristics on an object-by-object basis to seek optimal system performance. ***Note:*** *Initial tests do not indicate any performance benefits from this method beyond the benefits from page compression. We left the option in place, but we cannot recommend it over page compression without additional testing.*

When you set `CompressionType` to “3,” the hybrid approach uses statistics about the usage of individual objects to determine their compression types. Statistics are collected from the running system when `usp_CreateCompressionQueue` is run. To get a complete picture, you should run the SP after the system runs for a significant time – at least a day. To retain statistics from previous runs, set `OverwriteTableData` to “0.” (The default is “1.”)

Hybrid compression determines the compression type to use for each object based on the following values:

- If more than **MaxUpdateRateForPage** percent of the operations on an object are updates, use row compression. Objects that are updated frequently use fewer CPU cycles with row compression. The default value is 20 percent.
- If more than **MinScanRateForPage** percent of the operations on an object are scans, use page compression. Objects that are scanned frequently require fewer I/O cycles if they take fewer pages. The default value is 50 percent.
- If the object is used more than **MaxRelOpRateForPage** percent as much as the most used object in the database, use row compression. Objects that are used most frequently may require fewer CPU cycles with row compression. The default value is 1 percent.
- If the object is used less than **MaxRelOpRateForPage** percent as much as the most used object in the database, use page compression. Objects that are used infrequently take less space with page compression.

## Benefits of Compression in Blackbaud CRM Databases

As noted above, compression is a good idea for two reasons: To save space and to improve performance. The Performance team conducted tests to measure these benefits, and early results support the real-world value of compression. We also measured how long it takes to compress a database.

We used two approaches to test the value of compression. First, in the Enterprise Performance Lab, we experimented with representative databases to measure the time to compress data and the amount of compression. On one database, we also measured the performance of the system under a workload of interactive user actions.

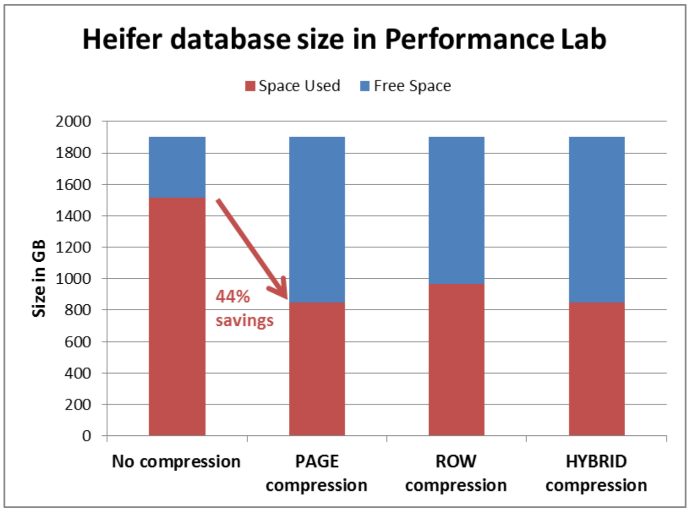
Second, our SDO team compressed the staging database for a hosted customer. The customer took performance measurements of key business processes and user actions before and after the compression to gauge the real-world value of compression.

These measurements are highly dependent on the database, server system, and workload.  As they say in the auto business, “Mileage may vary.” But we believe the results to be fairly representative.

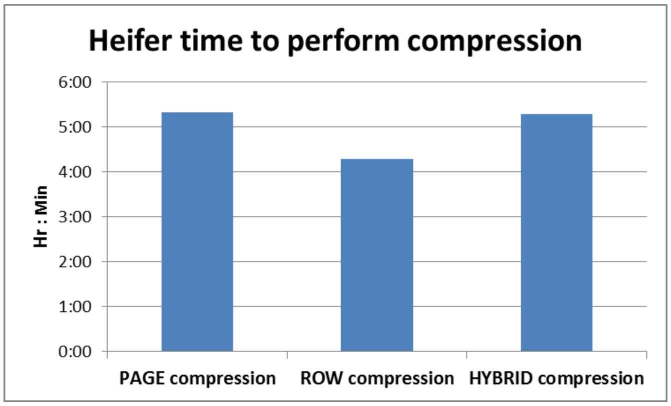
### Storage Space Reduction and Compression Time

We measured the space that compression saved in databases for two customers. The first customer, Heifer International, is a large organization with 4.6 million constituents and 1.5 TB of data in their database before compression.

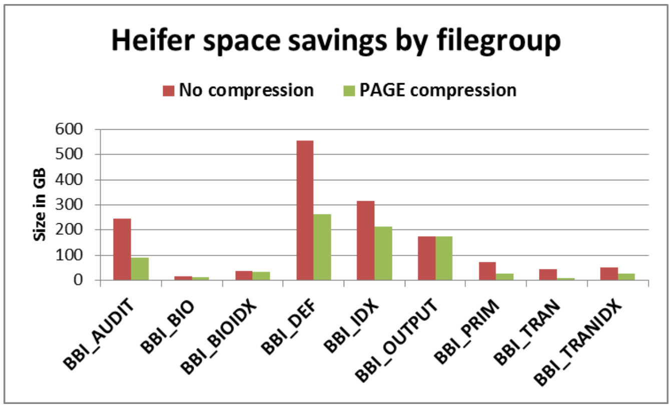
The second customer, “Site 2,” is a university with 1 million constituents and



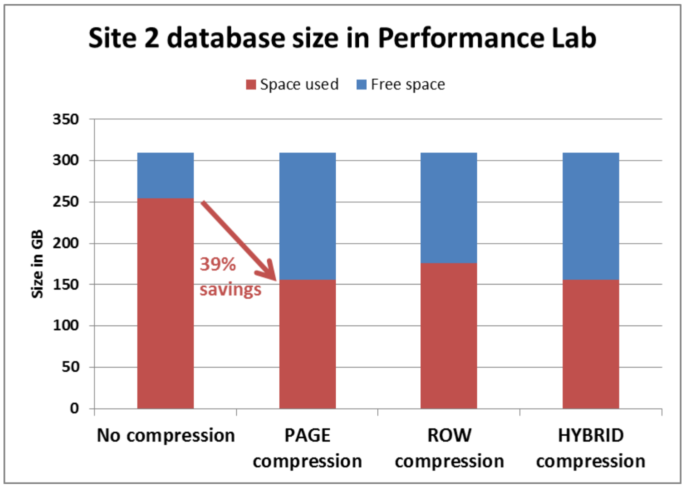
250 GB of data in their database before compression. The databases for these sites were brought in to the Enterprise Performance Lab and used for controlled measurements.



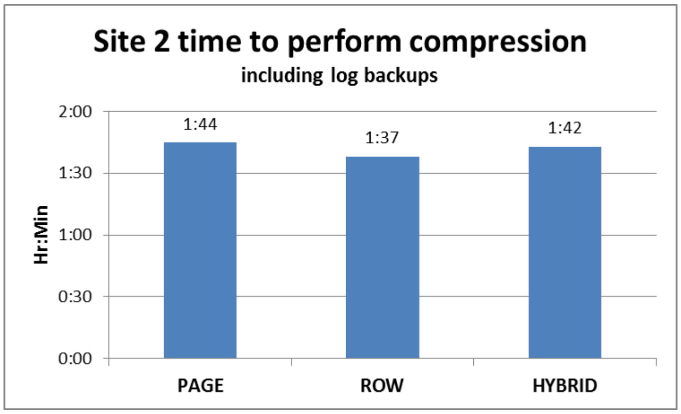
For Heifer International, applying page compression to all objects in the database reduced the amount of space used by ***44 percent***, as shown in the **Heifer database size in Performance Lab** chart. Compressing this database took 5 hours 20 minutes running continuously but not in parallel. We used option 3 from Log Files and Compression, so our results include time for log backups when necessary. We could compress a database this size overnight on our server. Row compression did not save as much space as page compression. It reduced the amount of space used by 36 percent, but the compression process ran faster at 4 hours 17 minutes. The hybrid approach's results were very similar to page compression.



The gains from compression were not uniform for all filegroups, so DBAs should continue to monitor the size and growth of individual file groups after they apply compression.

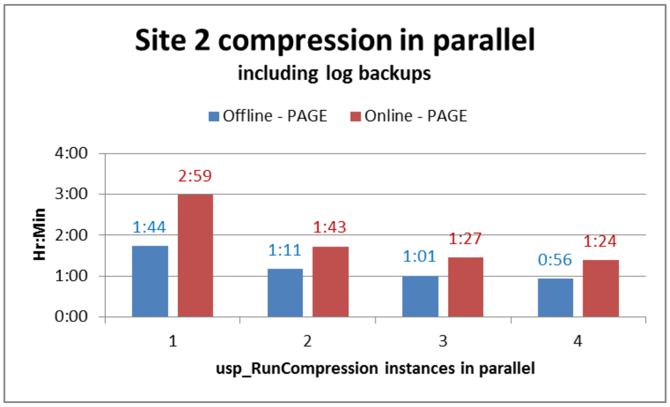


For Site 2, a ***39 percent*** space savings resulted from applying page compression to all objects in the database, as shown in the **Site 2 database size in Performance Lab** chart. Applying row compression reduced the amount of space used by ***31 percent***. Results from applying the hybrid heuristic method were very similar to page compression.



The time required to perform compression doesn't vary much based on compression type, as shown in the **Site 2 time to perform**

**compression** chart. However, using online operations results in a substantial increase in the time required to perform compression, as illustrated in the **Site 2 compression in parallel** chart. When online operations were used on a different server with less available RAM and a lower-performing disk subsystem, the time required to perform compression skyrocketed much more than the measurements in the chart would suggest.



The same chart also illustrates the impact of running multiple instances of the compression script in parallel. The lab server has a very good disk I/O subsystem, but we still saw the system quickly become I/O saturated when running parallel instances. In the Performance Lab, we did not see much benefit in running more than two instances concurrently. In another situation using a different server with less available RAM and a lower-performing disk subsystem, two instances in parallel overloaded the system much more than the measurements in the chart would suggest.

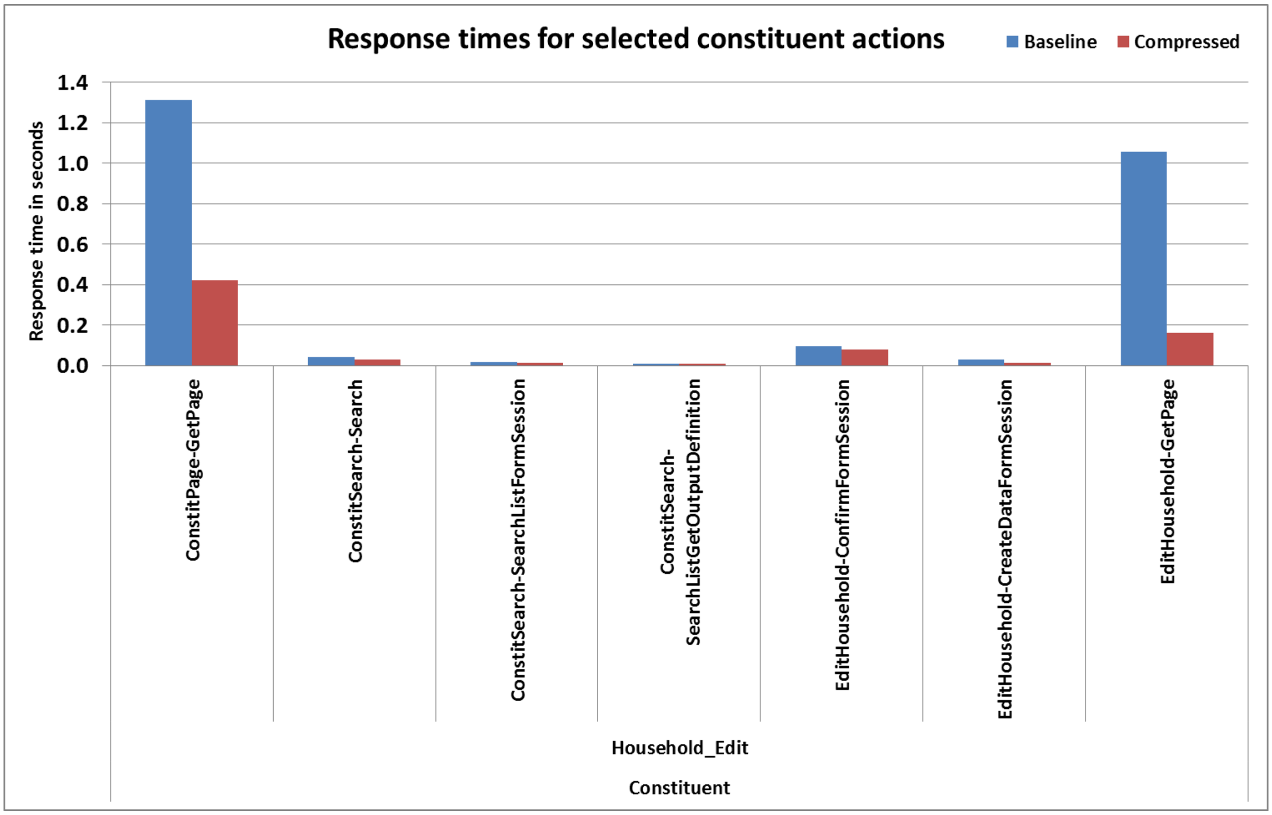
## Performance Gains Measured in the Lab

The Performance Team constructed a workload to run using the Heifer database. This is used for a variety of engineering purposes, and one application was to test compression. The workload represents the activities that a number of users perform while interacting with the system. It allows us to create a realistic but repeatable load on the system and to take performance measurements in that context. (However, the workload does not include business processes at this time.)

With this workload, we measured the performance of the system before compression and after compression. Based on these tests, we found several overall trends.

- **On average, actions are more than 2 times faster with compressed data**. In other words, an average action takes less than half as long when using compressed data.
- Changes in response time varied widely.
    - A number of actions are several times faster with compressed data.
    - Many actions are not significantly affected one way or the other by compression.
    - A few actions are somewhat slower with compressed data.
- The choice of page compression, row compression, or hybrid compression made an insignificant difference, but we recommend page compression because it saves the most storage space and has excellent performance characteristics.

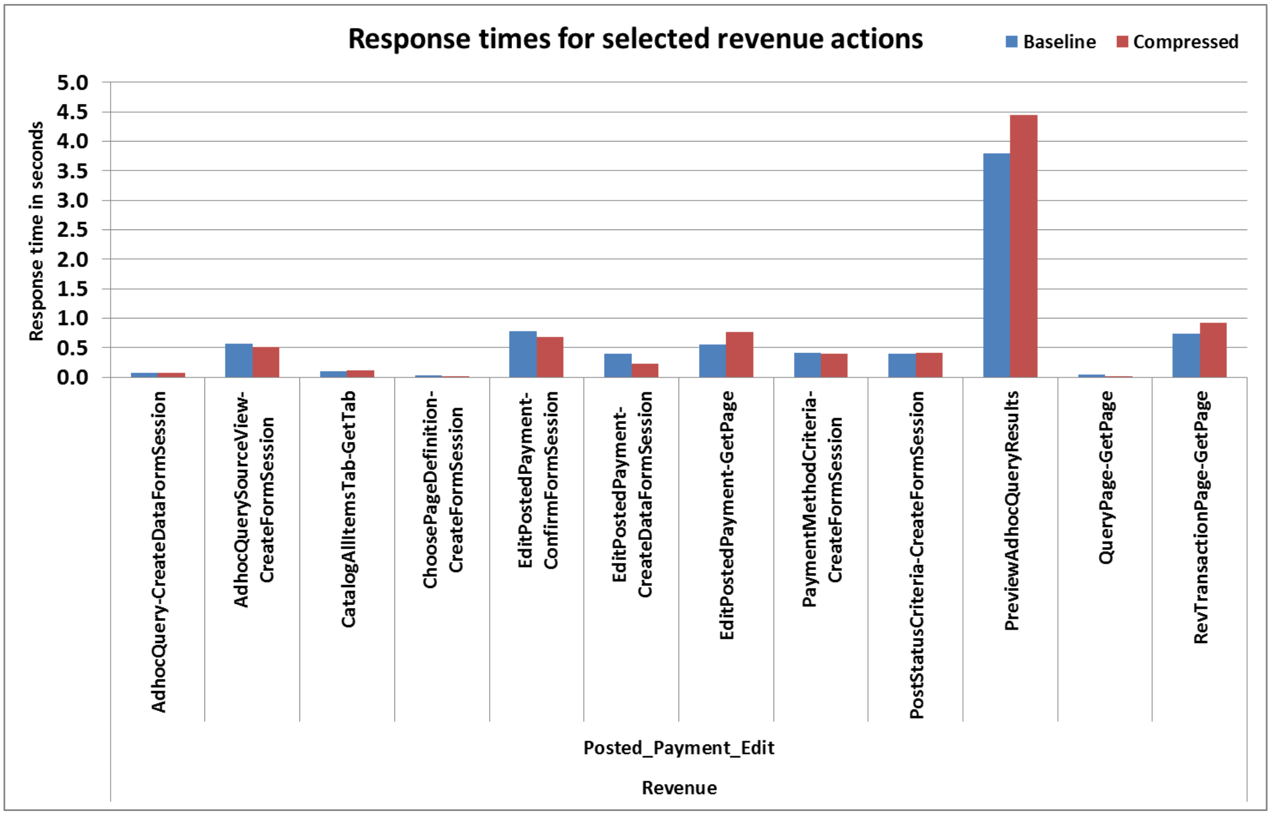
We included 128 different types of user actions in the workload, and we ran each one many times during the course of a workload run. We performed tests using all three compression methods. It's too much data to report fully in this paper, but the charts in this section show average response times for a few of the action types after page compression.



The **Response times for selected constituent actions** chart is very typical of the results from our testing. Many actions are not significantly altered, while some experience significant performance gains. Futhermore, we observed a tendency for longer-running actions to experience the larger performance gains. With more space, we could include many charts similar to this one. The **Response times for selected revenue actions** chart shows that we observed some negative changes. They are not nearly as big or as common as the gains, but negative changes did occur.

Because of the possibility of negative changes, we advise customers to test compression in their staging systems before they deploy it in production. That said, we expect substantially positive performance benefits from compression. On average, actions completed in half the time using compressed data.

While the performance of the system improved on average after compression regardless of compression type, the resource usage decreased. Disk I/O decreased by nearly 20 percent, and CPU usage by more than 20 percent.

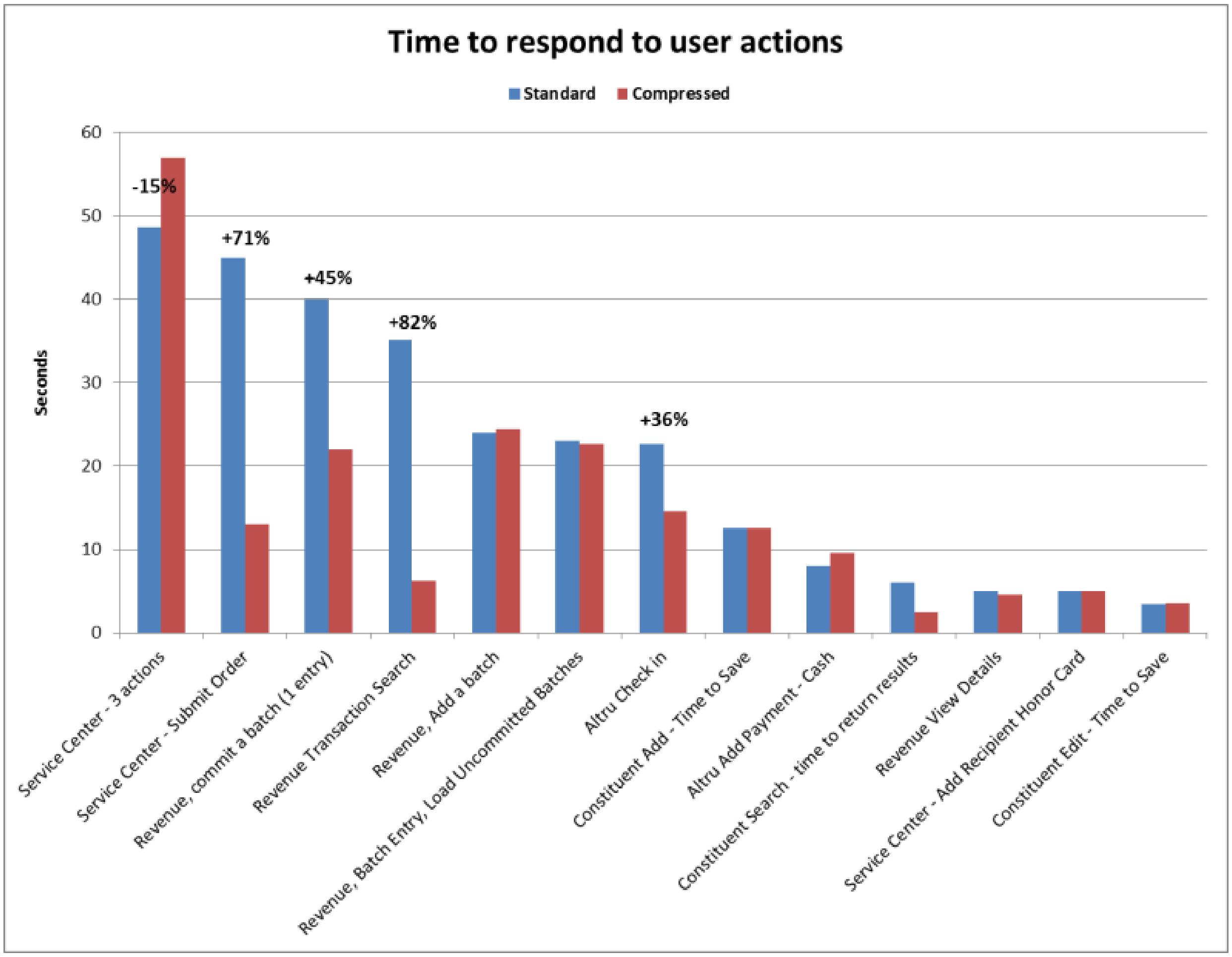

Response times for selected revenue actions
Baseline
Compressed
Response time in seconds
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
AdhocQuery-CreateDataFormSession
AdhocQuerySourceView-CreateFormSession
CatalogAllItemsTab-GetTab
ChoosePageDefinition-CreateFormSession
EditPostedPayment-ConfirmFormSession
EditPostedPayment-CreateDataFormSession
EditPostedPayment-GetPage
PaymentMethodCriteria-CreateFormSession
PostStatusCriteria-CreateFormSession
PreviewAdhocQueryResults
QueryPage-GetPage
RevTransactionPage-GetPage
Posted_Payment_Edit
Revenue

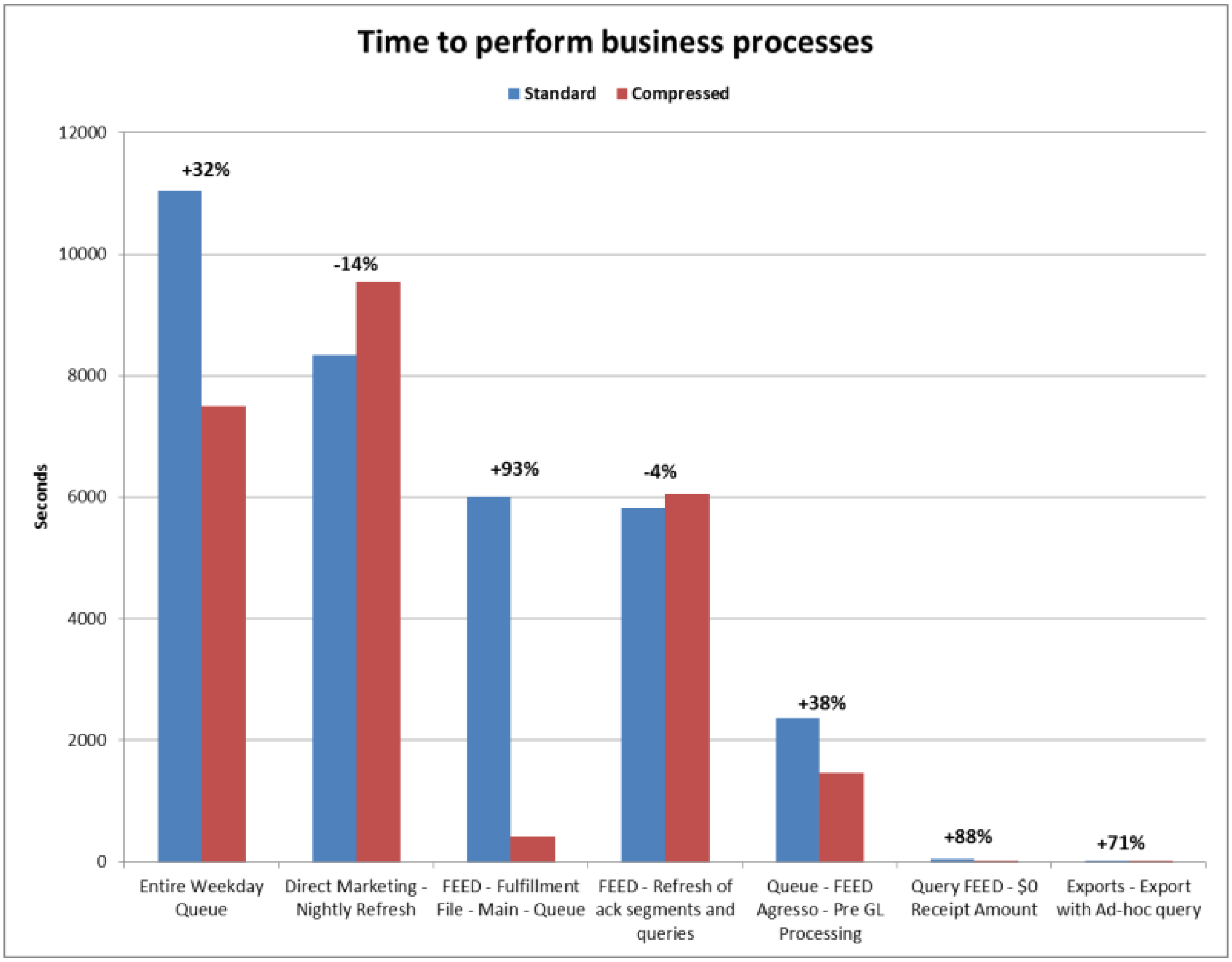
## Performance Gains Measured by Heifer

Heifer worked with SDO to test page compression on their staging system. They outlined a set of acceptance tests and performed them before and after compression. The tests were performed on a system that also hosts other databases. They were performed manually and timed using stopwatches, and some variability is to be expected due to those factors. In addition, the staging server is a much smaller machine than the production server. Still, the overall trend is similar to the observations from the lab tests described earlier. Some things were a lot faster, many things did not change significantly, and a few things were a little slower.

The **Time to perform business processes** chart shows the results of running business processes, while the **Time to respond to user actions** chart shows actions that interactive users on the system might perform.

Based on these results, Heifer is currently working with SDO to plan a deployment of compression in their production system.




Time to perform business processes
Standard
Compressed
Seconds
12000
10000
8000
6000
4000
2000
0
+32%
-14%
+93%
-4%
+38%
+88%
+71%
Entire Weekday Queue
Direct Marketing - Nightly Refresh
FEED - Fulfillment File - Main - Queue
FEED - Refresh of ack segments and queries
Queue - FEED Agresso - Pre GL Processing
Query FEED - $0 Receipt Amount
Exports - Export with Ad-hoc query