IBM Informix® Storage Optimization

A Technical White Paper

Contents

Introduction

Informix Storage Optimization offers an immediate solution to database planners and IT implementers who struggle to supply enough storage to keep up with growing data and enough processing power to run queries responsively. Informix Storage Optimization is breakthrough technology deployed in Informix 11 and Informix 12 that reduces the overall storage footprint of the database and better leverages available CPU and I/O capacity. Traditionally, the database footprint grows much faster than the actual data grows, due to built indexes, backups, and redundant storage, and this operational reality presents an even stronger advantage when using the Informix technology. Transaction logs also benefit from Informix Storage Optimization, and this results in additional performance gains in replication topologies and other scenarios where data is shipped across the network.

IBM Informix® 11 introduced Storage Optimization technology, and it was an immediate success. Informix® 12 expands the storage optimization capabilities by extending the compression functionality to indexes and simple large objects (TEXT/BLOB data types) and by providing automatic compression.

Informix provides the ability to turn on storage optimization and compress existing table data while applications continue to use the table. The automatic compression component of Informix Storage Optimization keeps tables compressed even as significant new data is inserted into the table. System down-time is not required to utilize the Informix Storage Optimization technology. Real-world applications experience a significant reduction in data storage, resulting in cost savings and time reductions in backup and restore operations.

Users of Storage Optimization have experienced significant performance improvements in applications from reduced I/O and improved buffer pool utilization.

What is Storage Optimization?

Informix Storage Optimization employs a set of tightly-integrated compression technologies to store data in memory and on the disk using fewer bytes, while keeping the data intact. It goes beyond simple compression in a number of ways, leveraged by the Informix database engine. The user needs only to do the simple set-up once; the Informix database takes care of the rest. All of this happens behind the scenes, and the advantages realized by way of reduced I/O, better in-memory coverage, better caching and better transaction logging more than make up for the extra work the CPU is doing for you in the background.

Informix Storage Optimization seamlessly handles many aspects of reducing the disk space used by the data:

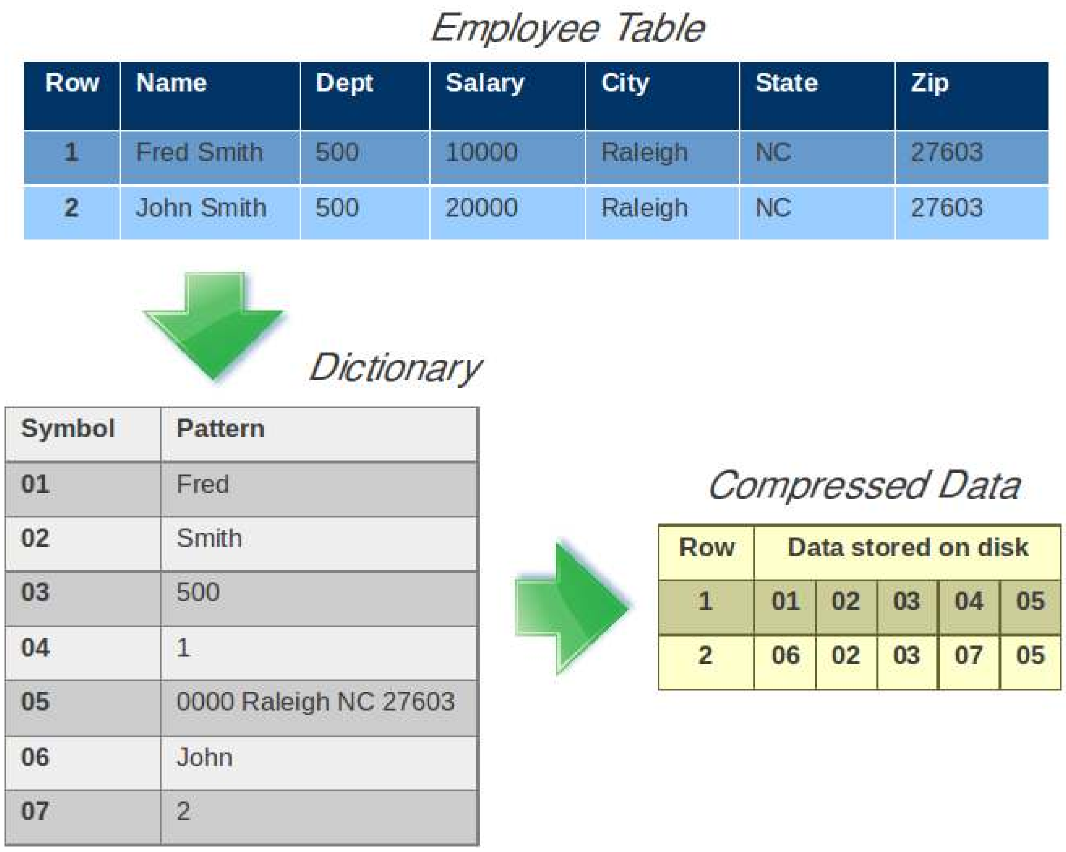
Compression	Compresses data in table or fragment rows, reducing the amount of required disk space
Re-pack data	Consolidates free space in tables and fragments
Shrink space	Returns free space to the dbspace
Defragment table extents	Brings data rows closer together in contiguous, merged extents

How does Informix Storage Optimization work?

One of the most important ways Informix Storage Optimization reduces the disk footprint is by compressing the data in the database records. Each record is considered a stream of bytes. Informix compression builds a customized dictionary of tokens for each partition, where each token represents repeating patterns of bytes occurring throughout the partition. By customizing a dictionary for each partition, Informix maximizes the compression rate for each table.

Informix supports compression of text and byte data types (simple large objects) stored in the same partition as the data rows, also known as partition blobs. A customized dictionary is created for each simple large object, tuned to the particular data in that large object. Dictionaries are stored in a repository in the same dbspace as the partition data. Informix provides the flexibility to choose the candidates for compression: in-row data, simple large objects, or both.

Figure 1 illustrates how row data in a table is converted into dictionary symbols.



Row	Name	Dept	Salary	City	State	Zip
1	Fred Smith	500	10000	Raleigh	NC	27603
2	John Smith	500	20000	Raleigh	NC	27603

Symbol	Pattern
01	Fred
02	Smith
03	500
04	1
05	0000 Raleigh NC 27603
06	John
07	2

Row	Data stored on disk				
1	01	02	03	04	05
2	06	02	03	07	05

Figure 1: Conversion of row data into dictionary symbols

Index compression works much in the same way as data compression. Any existing detached B-tree index on a fragmented or non-fragmented table may be compressed, repacked, and shrunk while the index remains available to queries. Any newly-created index may also be created with compression. A minimum of 2000 unique keys is needed for a compression dictionary to be created. Informix ensures efficiency by creating the index without the dictionary if there are too few unique keys – too small a dictionary would lead to compression without sufficient benefit. The quickest way to check the candidates for compression is the graphical display in the OpenAdmin Tool, which shows which indexes will benefit from compression. Also, the unique key statistics reveal whether or not the index is a good candidate for compression: use the *oncheck –pT* command and view information in the *Number of keys* field. To compress an already existing index, run the SQL Admin API task() or admin() function with the index compress argument. The compression operation compresses only the leaves (bottom level) of the index.

Informix Storage Optimization goes beyond just compression allowing for further storage efficiency. Version 11 introduced repack, shrink and extent defragment for tables. In Version 12, those operations have been extended to support index partitions and tables with partition blobs. A compressed table can be repacked at any time, any number of times over the course of its life, all while permitting other concurrent activity. This flexible, low-impact design is combined with an SQL interface, providing flexible and powerful capabilities. The highlights are:

1. Create the compression dictionary.
2. Compress the rows.
3. Coalesce (re-pack) the rows.
4. Reclaim free space.
5. De-fragment the table extents.
6. Automatic compression.

Creating the compression dictionary

A dictionary is created by sampling a set of rows from an existing table or table fragment and then creating a lookup table of symbols that represent byte patterns. The dictionary is stored in the dictionary repository, which is included in the dbspace within the compressed partition. The dictionary also has an in-memory representation, so that active queries and updates can quickly compress and uncompress data with minimal impact to performance.

The size of each dictionary is approximately 75 KB, but each dictionary can grow to be as large as 150 KB. Keep in mind that the dictionaries require some memory: there is one per compressed partition.

Compressing the rows

Starting in Version 11, row compression is applied to existing tables after the data is loaded. This work is done as a background task that compresses each row and leaves the compressed data in the page that contains the row. It is done as small transactions in parallel with normal business transactions to minimize any impact from the background task. The rows being actively compressed are locked only for a short duration. Any new rows that are inserted or updated are compressed automatically. The action in Version 11 is triggered by the sysadmin database task procedure. Version 12 introduces automatic compression, where Informix automatically creates the dictionary and starts compressing the data rows as soon as 2000 rows are inserted into the partition. In addition, if a table is being loaded using light append (via High Performance Loader or RAW tables), compression will buffer rows prior to inserting them into the partition, create the dictionary and then insert, so that all the rows inserted as part of the load are compressed.

Coalescing (repacking) the rows

After a partition is compressed, typically there is a significant amount of unused space (or 'holes') between the rows. The coalesce operation, also known as a repack operation, moves all of the rows to the front of the partition. It uses small transactions and locks only those rows actively being moved. In Version 12, Informix has added the capability to repack tables that contain partition blobs and indexes.

Reclaiming free space

After the rows are repacked, the reclaim operation truncates the unused portion of the partition, and returns the space back to the dbspace where the partition is located. For an index, this returns free space at the end of the index to the dbspace, thus reducing the total size of the index.

Defragmenting table extents

Defragmenting the table extents brings data rows closer together in contiguous, merged extents. A portion of the benefit from this step comes from more efficient data access, while a portion comes from data structure space savings from reducing the total spread of the extents.

Automatic Compression

When the table is created with the COMPRESSED option of the CREATE TABLE statement, Informix performs automatic compression. When the table reaches 2000 rows required for the compression dictionary, any existing and newly-inserted data is automatically compressed. In addition, when there is an attempt to compress a table and it doesn't have a sufficient number of rows to create the compression dictionary, the table will automatically be compressed as soon as it has sufficient rows for compression.

When to use Informix Storage Optimization?

Informix Storage Optimization improves performance and reduces the memory and disk footprint in a very wide range of real-world scenarios. Saving storage and memory significantly reduces storage cost. There are a variety of scenarios where compression can provide significant benefit, both large and small. The amount of benefit depends primarily on the size of the data and the nature of the data, and even small savings may significantly boost an application where space is a premium or extra performance is valued.

How much will you save? To see how your data might compress, use the Storage Compression Estimation tool at ibm.com/informix/compression. This tool takes information from your existing infrastructure and estimates the compression ratio for your data so you can determine the potential savings in your environment.

But let's look a little deeper, beyond just disk space savings. Here are some other motivators for using Storage Optimization:

Optimizing space

- To reduce storage costs. Smaller databases mean less primary storage and smaller redundant storage (HDR, backups, fewer log backups), and the custom dictionaries achieve high compression ratios.
- The shrink operation improves utilization by reclaiming space for new tables to use.

Performance

- Sequential scans of tables require fewer I/Os
- Leaf scans across the index require fewer I/Os
- There is better use of the buffer cache since more data effectively fits in. Every page found in the cache reduces I/O and is retrieved faster (can be microseconds for the cache hit instead of milliseconds for the I/O)
- Smaller b-trees mean fewer levels of the tree to traverse. Example, demonstrated by the use of oncheck -pT:
  - Before:

```
Index Usage Report for index ix1_fact on inventory:acct.fact

                     Average    Average  Average
    Level    Total No. Keys Free Bytes Del Keys
    ----- -------- -------- ---------- --------
        1        1      493       2252
        2      493      309       3475      222
    ----- -------- -------- ---------- --------
    Total      494      309       3472   109610
```

  - After index compress repack:

```
Index Usage Report for index ix1_fact on inventory:acct.fact

                     Average    Average  Average
    Level    Total No. Keys Free Bytes Del Keys
    ----- -------- -------- ---------- --------
        1        1      101       6956
        2      101      425        939        0
    ----- -------- -------- ---------- --------
    Total      102      421        998        0
```

- ○ Smaller backups mean faster backups to create and restore
- ○ Less data to ship across the network in replication environments

A typical database requires more storage than one might perceive. Consider the following scenario:

The database is a 2 TB system. Within that system, 1 TB is a candidate for compression and it has a compression ratio of 50 percent. The initial storage savings for that system starts at 500 GB. The system also generates 20 GB of log data each day, and the policy is to keep 30 days worth of logs. Because Informix compresses the log data for compressed tables, typically 30 percent of that log storage can be saved. This results in 6 GB per day, or around 180 GB per month. If 3 complete backups of the data are kept, another 1.5 TB can be saved, because the compressed data takes less backup space.

It adds up to more than the 1 TB that was initially considered:

```
 500  GB less storage for the database
 180  GB less storage for log backups for a month
1500  GB less storage for archives
=====================================
2180  GB total storage savings
```

This is the immediate storage space saved in the first month!

Here is an actual scenario from an Informix customer:

The customer: a very large network infrastructure provider who handles multiple terabytes of data with thousands of concurrent connections. This customer was experiencing high expense due to the heavy storage needs. Informix Storage Optimization helped this customer save money by reducing the total disk footprint. The savings occurred while the system remained active and tables were continuously accessed and modified.

Informix Storage Optimization achieved this goal with ease, saving space via compression, repacking, shrinking, and defragmenting, while improving performance with high throughput.

- ✔ Space saving: The free space increased by 68% while the average log size decreased by 22%

- ✔ I/O improvement: Depending on the activity rate and the time-of-use, the cache rate improved by 23% to 98%. The I/O operations dropped by 4,400/second, which resulted in an average of 17 million operations per hour

- ✔ Throughput: Overall transactions rate increased by 29.5%. The CPU usages increased by 11.8% while the CPU usage per transaction decreased by 8%. The number of logical logs that were created increased by 27.6% along with the increase of log space usage by 7.8%

Here is another success story from an Informix customer:

> *The customer: a financial services company. This customer reported a reduction of an 8 TB database to just under 2 TB. The customer was thrilled because not only did they save storage, but their backups were significantly faster because of the reduction in the number of I/Os because the database size had been reduced significantly.* *Figure 2 shows the savings and final compressed size in this story.*

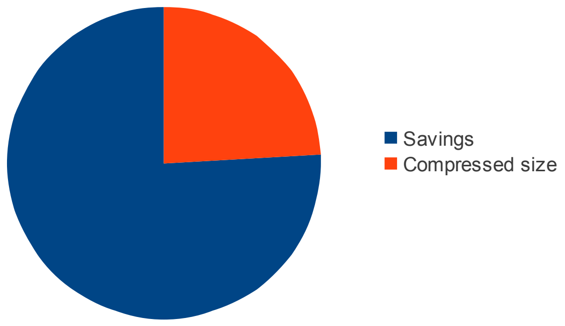


Figure 2: Disk Space Savings

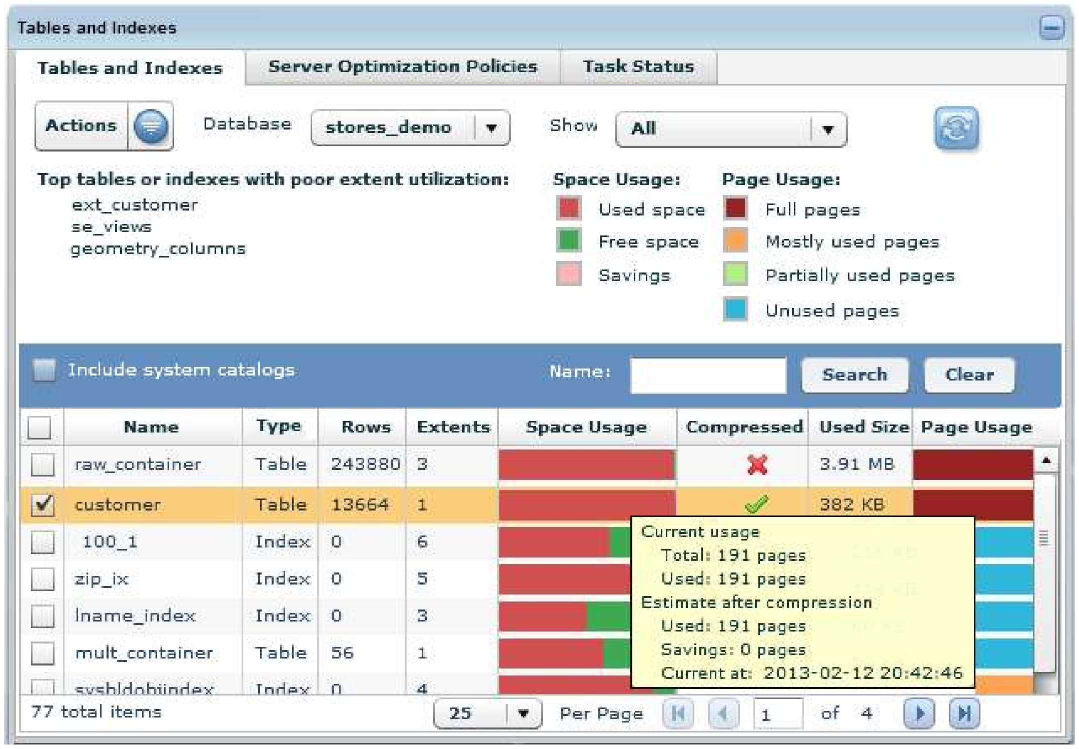
Informix Storage Optimization is not only about saving disk space. It also improves I/O, reduces back-up time, improves cache hit ratios, and reduces row splitting across pages.

Informix Storage Optimization via OAT

The IBM OpenAdmin Tool (OAT) provides a simple way to interact with Storage Optimization. OAT provides an easy interface to identify compression candidates and perform a variety of administration tasks, including setting up Storage Optimization.

OAT shows you the tables within a selected database. After the database is selected, OAT selects the tables within that database that might benefit from being compressed and calculates the approximate space savings. If you hold the cursor over the ***Space Usage*** column for a given table, OAT displays a compression estimate. You can use OAT to keep up with ever-changing data and configure a task to periodically search all the tables and fragments contained within the Informix instance for good compression candidates. The task then automatically submits a job to compress, repack, and shrink the partition.

Figure 3 illustrates the database view with a compression estimate for the *customer* table. The green check-mark indicates that the table is already compressed. Although compression has been enabled, the figure does not show compression estimates for other user tables or indexes because the tables and indexes do not satisfy the minimum compression requirement (2000 rows for table or 2000 unique keys for an index).



Tables and Indexes

Tables and Indexes | Server Optimization Policies | Task Status

Actions · Database: stores_demo · Show: All

Top tables or indexes with poor extent utilization:
- ext_customer
- se_views
- geometry_columns

Space Usage:	Page Usage:
🟥 Used space	🟥 Full pages
🟩 Free space	🟧 Mostly used pages
▨ Savings	🟩 Partially used pages
	🟦 Unused pages

Include system catalogs · Name: · Search · Clear

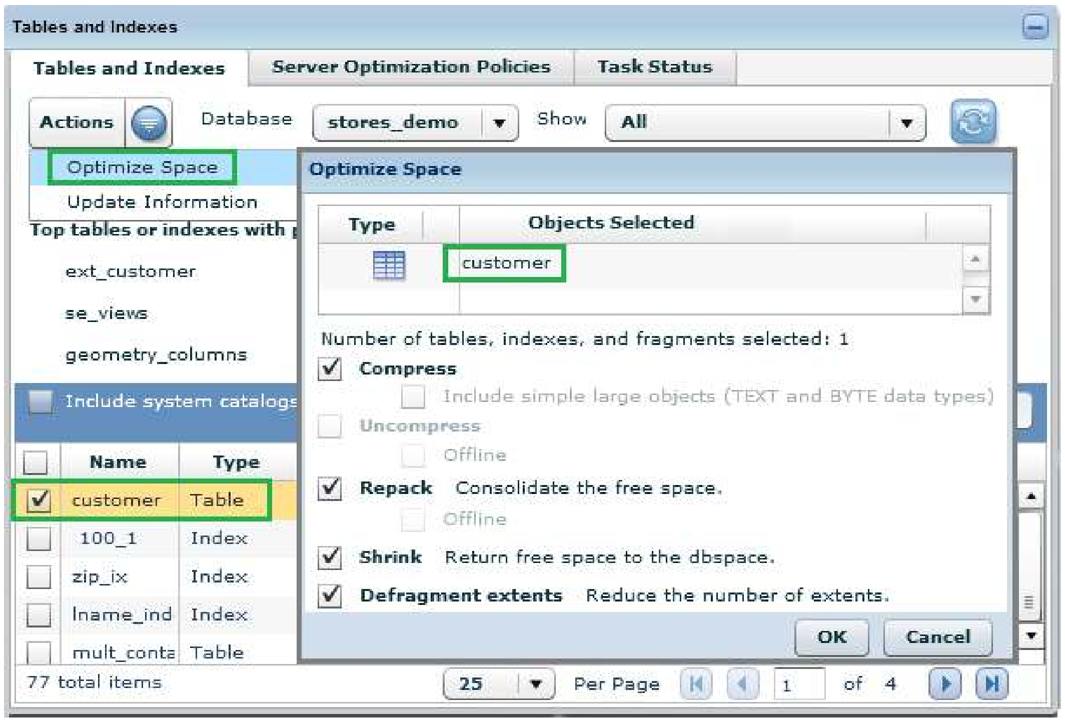
	Name	Type	Rows	Extents	Space Usage	Compressed	Used Size	Page Usage
☐	raw_container	Table	243880	3		✗	3.91 MB	
✓	customer	Table	13664	1		✓	382 KB	
☐	100_1	Index	0	6				
☐	zip_ix	Index	0	5				
☐	lname_index	Index	0	3				
☐	mult_container	Table	56	1				
☐	sysbldobjindex	Index	0	4				

```
Current usage
   Total: 191 pages
   Used: 191 pages
Estimate after compression
   Used: 191 pages
   Savings: 0 pages
   Current at:  2013-02-12 20:42:46
```

77 total items · 25 Per Page · 1 of 4

Figure 3: Database: Table view with a compression estimate

After you choose to compress a table, index, or fragment, by selecting the candidate and initiating *Optimize Space* from the *Action menu*, a new screen appears. You can use this screen, as illustrated in Figure 4, to choose the actions to perform.   This is where you can compress, repack, shrink, and defragment.



Tables and Indexes

Tables and Indexes | Server Optimization Policies | Task Status

Actions · Database: stores_demo · Show: All

Actions menu: Optimize Space · Update Information

Top tables or indexes with p…
- ext_customer
- se_views
- geometry_columns

Include system catalogs

	Name	Type
✓	customer	Table
☐	100_1	Index
☐	zip_ix	Index
☐	lname_ind	Index
☐	mult_conta	Table

77 total items · 25 Per Page · 1 of 4

Optimize Space

Type	Objects Selected
▦	customer

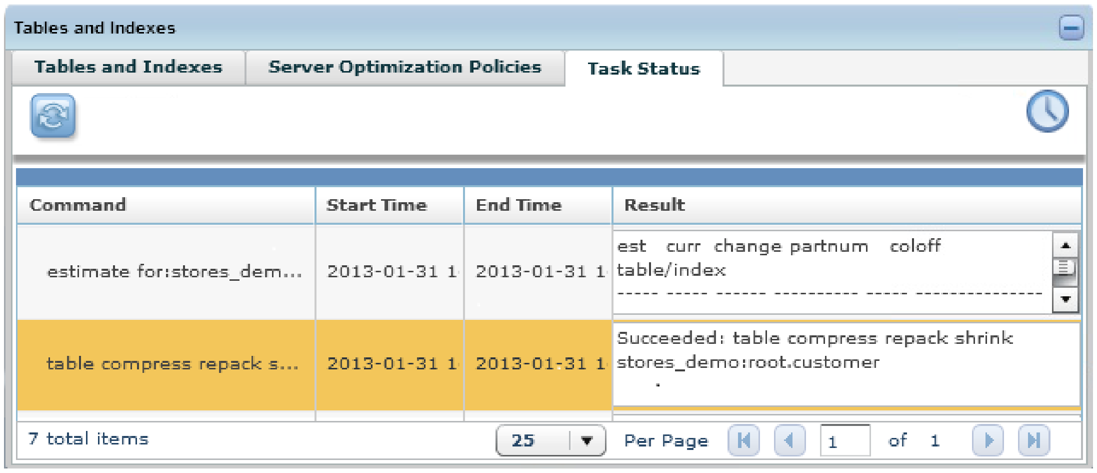
Number of tables, indexes, and fragments selected: 1

- ✓ **Compress**
  - ☐ Include simple large objects (TEXT and BYTE data types)
- ☐ Uncompress
  - ☐ Offline
- ✓ **Repack** Consolidate the free space.
  - ☐ Offline
- ✓ **Shrink** Return free space to the dbspace.
- ✓ **Defragment extents** Reduce the number of extents.

OK · Cancel

Figure 4: Choosing action for compression

After you choose which compression operations to perform, Informix runs the job. You can immediately go to the *Task Status* tab to see the status of running tasks. Figure 5 shows the status of a compression job that was initiated on the table.

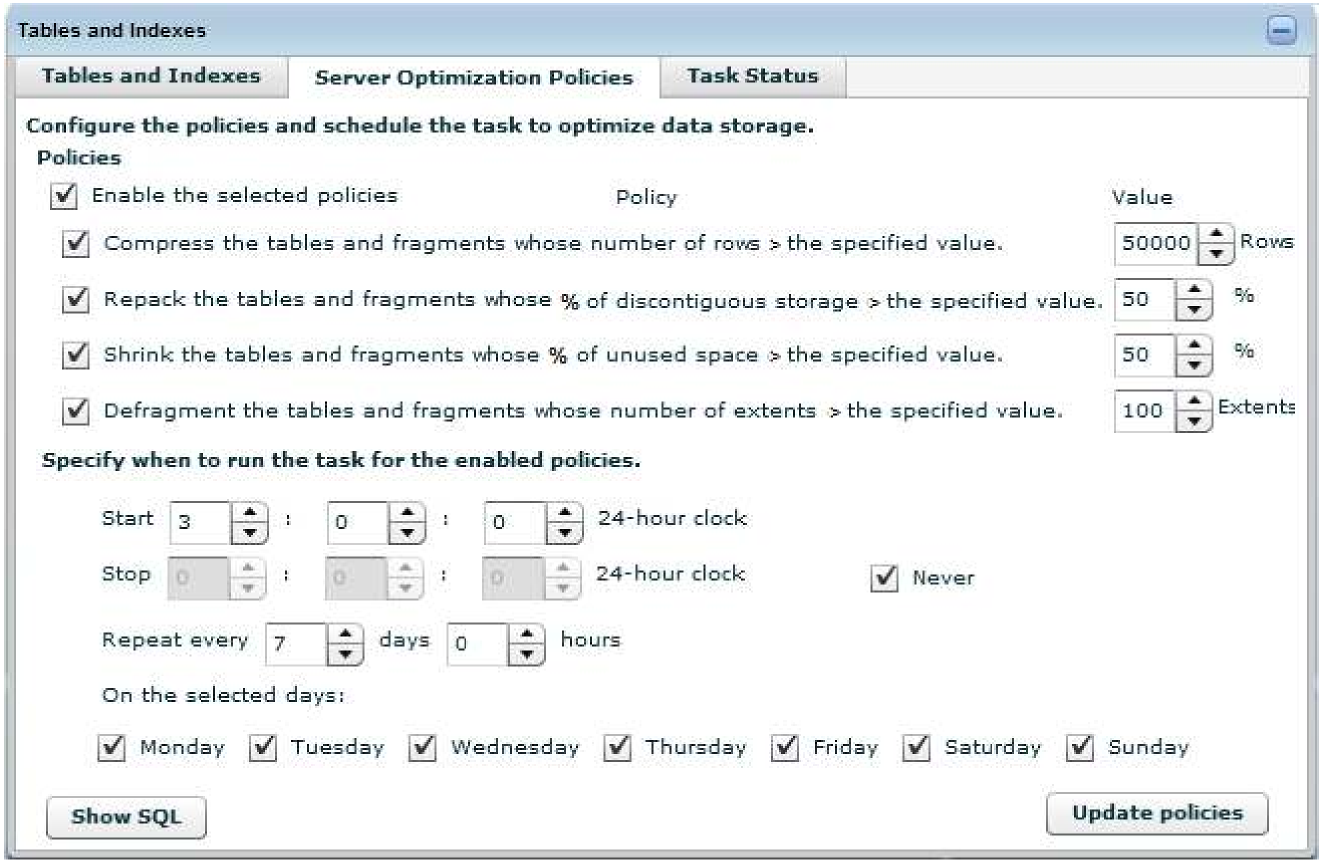


Command	Start Time	End Time	Result
estimate for:stores_dem...	2013-01-31 1	2013-01-31 1	est curr change partnum coloff table/index ----- ----- ------ ---------- ----- ---------------
table compress repack s...	2013-01-31 1	2013-01-31 1	Succeeded: table compress repack shrink stores_demo:root.customer

7 total items 25 Per Page 1 of 1

Figure 5: Status of compression job

It is equally easy to set up automatic compression. Under the Server Optimization Policies, select the defaults for compression, repacking, shrinking and defragmenting. Then set up a schedule for the automated task. Figure 6 illustrates how this looks in OAT.



Configure the policies and schedule the task to optimize data storage.

Policies

✓ Enable the selected policies

	Policy	Value	
✓	Compress the tables and fragments whose number of rows > the specified value.	50000	Rows
✓	Repack the tables and fragments whose % of discontiguous storage > the specified value.	50	%
✓	Shrink the tables and fragments whose % of unused space > the specified value.	50	%
✓	Defragment the tables and fragments whose number of extents > the specified value.	100	Extents

Specify when to run the task for the enabled policies.

Start 3 : 0 : 0 24-hour clock

Stop 0 : 0 : 0 24-hour clock ✓ Never

Repeat every 7 days 0 hours

On the selected days:

✓ Monday ✓ Tuesday ✓ Wednesday ✓ Thursday ✓ Friday ✓ Saturday ✓ Sunday

Figure 6: Enabling automatic compression

Another way to set up compression is outside of OAT, in the CREATE TABLE and CREATE INDEX statement. For example, initialize a compressed index by specifying the COMPRESSED option of the CREATE INDEX statement during the index creation time. There is no automatic compression for indexes – only when there are at least 2000 unique keys will the index be created to leverage compression. However, once a compressed index has been created, every subsequent index entry is compressed.

Additional Storage Optimization Considerations

Database backup performance improvement

Database backups can also experience significant performance improvements, because the size of the database is significantly smaller. The reduced database size for archiving means fewer I/O operations leading to faster backups. It is important to note that for large tables, the repack and shrink options must be executed for the data to coalesce. This removes unused extents from the end of the table.

Improving cache hit ratio

Each application has a *working set*. The working set is the data that the application queries over a particular interval. The more of the working set that is contained in memory, the better the application will perform, as this means it performs less I/O. When the working set fits into the bufferpool, Informix Storage Optimization will not improve performance. But when the working set does *not* fit into the bufferpool, compressing the data may significantly improve performance: well-compressed data is smaller and more likely to fit, and thus there is less I/O.

Here is an example derived from the TPCC benchmark where compression improved performance due to an a better cache hit ratio. It's not important that it's TPCC data, just that we use it as an example where all data is equal, and therefore all data is part of the working set. Growth of the working set is simulated by increasing the number of warehouses and the number of users. This increases the size of all of the tables, and it is similar to a business that is growing with more application sessions and more data to manage. Here, the amount of memory and CPU processing power is constant. As the working set grows, the number of I/Os increase, which decreases overall throughput.

Figure 7 shows the result. To the left of the chart, the workload started with 10 warehouses and 5 users. This small size is totally cached, and no I/O occurs other than transaction logging, so the overall throughput good even without compression. Compression fares slightly worse because of the overhead to materialize the row with each row access. As I/O is introduced into the workload, it quickly crosses over to a point where compression improves performance due to the reduced I/O. As the processes become more and more I/O bound, both results taper off.

None of the experiments were done with the system being fully CPU-bound in the cached case or fully I/O- bound in the non cached cases.

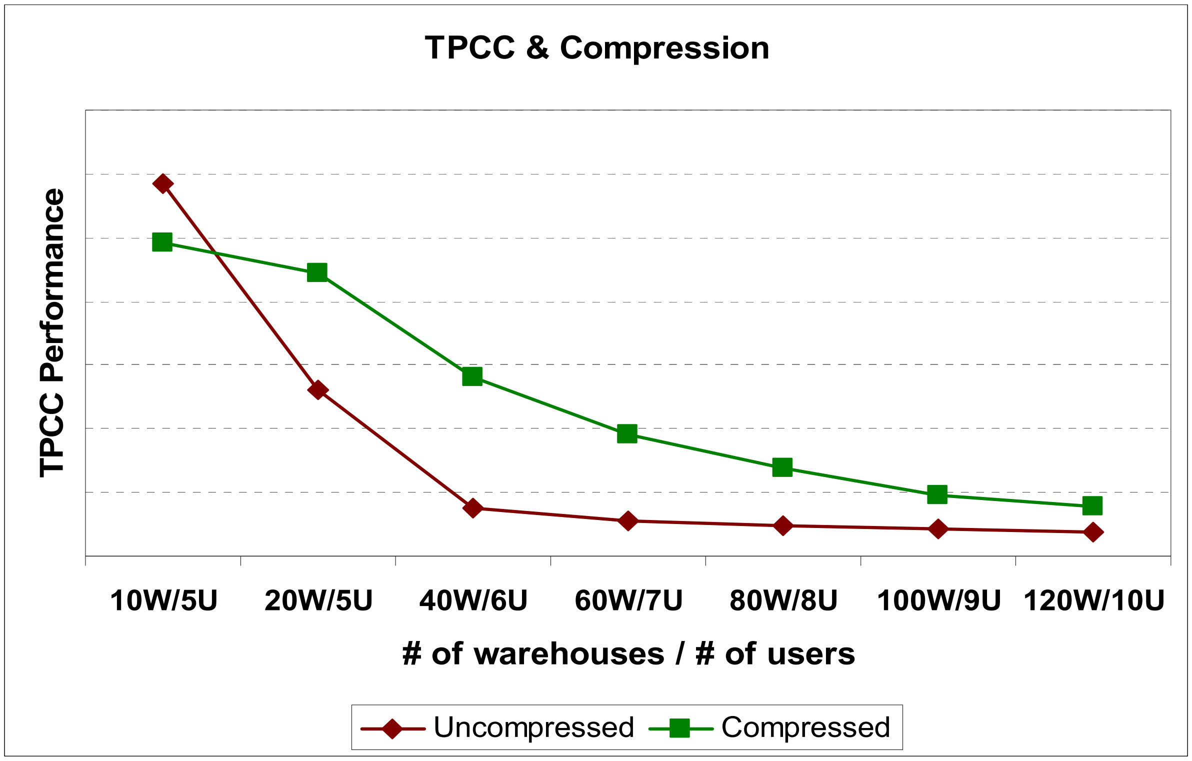


Figure 7: Workload data comparison for compression

Improving Response Times

Another way that compression improves performance is by helping applications maintain good response times. Often, an application must adhere to strict service level agreements, where a nominal response time must be maintained for every transaction. As a database grows and the working set cannot be maintained within the buffer cache, there is often a dramatic degradation in response time because the speed of I/O is considerably slower than a memory access.

Figure 8 shows the results of an experiment to demonstrate response times as the size of the working set grows, and it shows how Storage Optimization makes it possible for many applications to meet a particular required response time. The figure also shows database size, total I/Os, I/Os per transaction, and response time, with and without compression. Each successive color in the bar chart represents an increase in the size of the database, and in the uncompressed case, the response time increases rapidly, while in the compressed case, the impact is quite moderated.

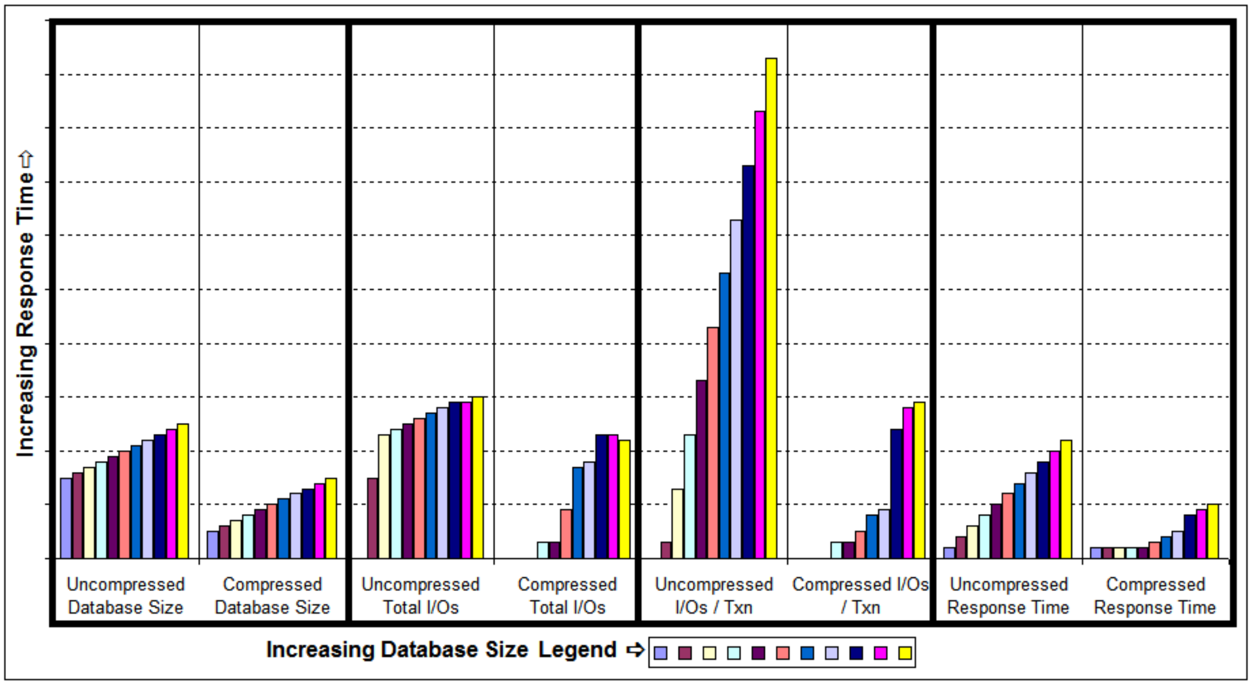


Figure 8: Transaction response time versus data growth

Large rows that are split across pages

When a row is too large to fit on a page, you need multiple I/Os to retrieve all the pieces of the row. Compression can often reduce the size of the row so that the entire row fits within a single page.

You can use the **oncheck –pT** command to display the number of remainder pages a table is using before and after data is compressed. The information displayed in the “Compressed Data Summary” section of the output shows the number of any compressed rows in a table or table fragment and the percentage of rows that are compressed. If rows are not compressed, the "Compressed Data Summary" section does not appear in the output.

Cached tables

In some situations, you should not compress data. When a table is already cached, compressing the table can degrade performance. If a cached table is compressed,   additional overhead is needed to materialize the uncompressed version of the row with each row access. This adds an additional CPU cost.

Random access across a huge data set with no clustering

In some other situations, a performance improvement will not be visible even when there are significant storage savings. These situations can be evaluated to determine the overall payoff for implementing compression technology. For example, when performing OLTP style queries across a huge data set, where the data is not clustered, Informix Storage Optimization might not improve performance. Consider the following example:

- Table T1 has 100 M rows across 1 M pages (100 rows per page)
- The application randomly accesses 100 rows / sec from the table

In this example, there is one chance in a million that a given row operation will access a particular page. Since all accesses have an equal chance, it is unlikely that there will be enough accesses to the table that they will hit the same page while the page is in the bufferpool. Informix Storage Optimization is still a great mechanism to reduce storage costs, as noted above, but in this case, where the working set is huge, it simply does not help improve performance.

Summary

The Informix storage optimization feature saves disk space and memory footprint, and therefore, saves you operating cost. Other benefits include performance improvements due to such things as fewer I/Os and more likelihood of data fitting inside memory. In fact, it is for the performance benefits that many of our customers use Storage Optimization. It's simply easy to use, both from the command interface when creating tables and indexes and by using the OpenAdmin Tool. It's simply automatic when you want it to be, and the work is done behind the scenes while the normal business transactions happen concurrently. There is minimal impact to existing applications, and it can be simply deployed in production environments.

Index of Figures

For more information

To learn more about the Informix features, contact your IBM representative or IBM Business Partner, or visit ibm.com/software/data/Informix

IBM Informix Storage Optimization
A Technical White Paper

March 2013
Scott Lashley, Anup Nair, and Jef Treece

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.