



RainStor Data and Pattern De-duplication

Extreme data compression

This paper describes the data and pattern de-duplication techniques that are utilized by RainStor to deliver extreme data compression. RainStor achieves compression ratios that are typically 40:1, rising to 100:1 with some data sets, through the use of four distinct but complementary techniques:

1. **Field level de-duplication**
2. **Pattern level de-duplication**
3. **Algorithmic compression**
4. **Byte level compression**

Our implementation of field and pattern level compression is fundamental and unique to the RainStor technology, and is subject to several worldwide patents. Let's consider each technique in turn.

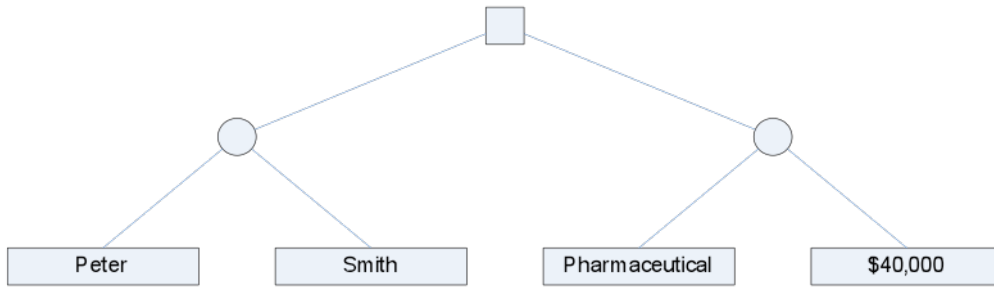
Field level de-duplication

Field level de-duplication involves processing the source data on a column-by-column basis, and reducing the dataset to only a list of the unique values that each column holds, together with a frequency count of the number of times the value appears. For example, in a database of automobile insurance data the manufacturer and model fields are likely to form only a few hundred unique values, however many millions or billions of records the database contains. In this instance the storage space required using field level de-duplication is a fraction of the original data.

This explains only part of the scenario, as simply compressing the data down to a list of the unique values would not enable us to reconstruct the original data from the compressed form. In order to do this, and enable us to store compressed data in a 'loss-less' state, a 'binary tree' is built up with pointers that can be used to reconstitute the data as it was in its original form.

Maintaining the original data structure

How we store compressed information without losing its structure is one of RainStor's key features, so let's consider it separately by taking an example of company records of individuals, the division for which they work, and the value of their last order. 'Peter, Smith, Pharmaceutical, \$40,000' would be held like this:

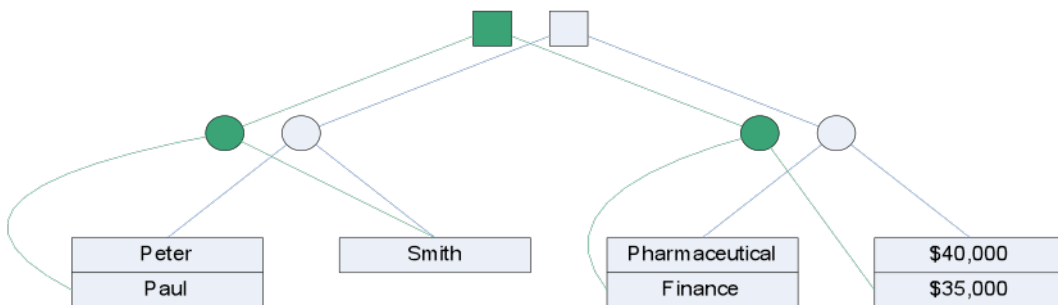


Within this tree the top level node is called the 'root', the subsequent nodes are called the 'branches', and the final elements containing the actual data are called the 'leaves'. The roots and branches are stored as an ordered set of pointers, and the leaves are an ordered set of de-duplicated data.

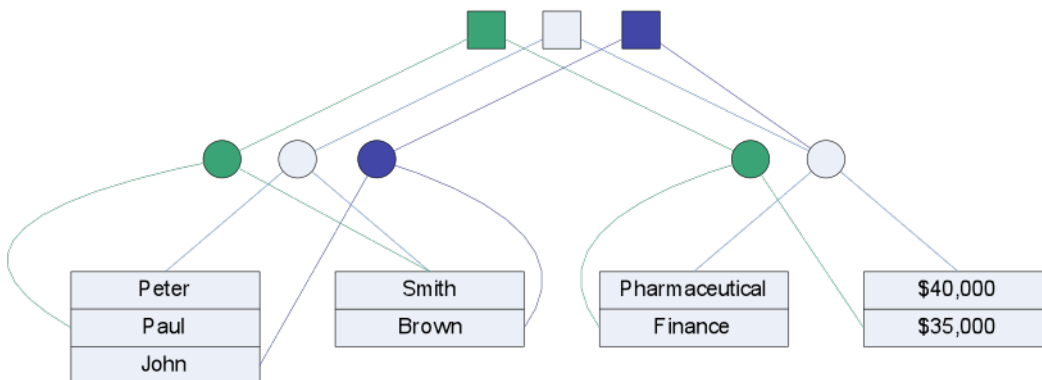
Pattern level de-duplication

Pattern level de-duplication builds on field level de-duplication by further exploiting the ability to store only unique values of the branches, again with a frequency count. This is achieved using exactly the same technique as used at the field level to simply work out the unique combinations.

Let's say we now want to store 'Paul, Smith, Finance, \$35,000'. We can add a record to our data tree, as shown in green. The new node shares 'Smith' as a leaf, but has a new leaf for 'Paul', and a new pair of leaves for 'Finance' and '\$35,000'. This gives us the following:



Next we want to store 'John, Brown, Pharmaceutical, \$40,000'. We add a new record to our data tree, shown in blue. This node has new leaf values for 'John' and 'Brown', but shares the same pattern that pairs 'Pharmaceutical' with '\$40,000'. This gives us:





Note that the only part of the tree structure that is the same size as the original data is the root. As the root is just a node in the binary tree, with a left and right pointer, it uses much less storage space than the original. In addition, the diagram shows the de-duplication beginning to happen at the branch level, with two pointers to the 'Pharmaceutical – \$40,000' pattern, as well as at the leaf level.

In practice, the compression can be further optimized by carefully constructing the design of the tree to place the nodes (be they leaves or branches) with the most frequent combinations next to each other – they are placed on the left-hand side. This placement is optimal for compression purposes, as you get smaller sized branch sets as the tree increases in size. RainStor algorithms ensure that the optimal design is used every time.

Algorithmic compression

Field and pattern compression techniques are equally valuable in saving space on disk as in saving memory. However, RainStor also employs algorithmic compression that involves a series of techniques designed primarily to reduce the amount of disk required for storage. Often an algorithmic compression technique is applicable to more than one area of the data stored on disk. For example, the use of expressions to represent sequences can be applied equally to leaves, branches, and the root.

It should also be noted that the algorithmic compression techniques, while making the footprint on disk smaller, result in little or no degradation in read times. Although there is some additional light processing needed to reverse the compression, this is more than compensated for by the reduced amount of data that needs to be retrieved from the physical storage, which is always an expensive part of the operation.

Byte level compression

The final form of compression is byte level compression. Here components of the tree are aggressively compressed independently using industry-standard byte compression algorithms tuned to obtain the maximum on-disk compression of the data.

Optimizing for query or space

There is a choice (the only configuration choice when deploying RainStor) that must be made when combining these compression techniques before building the data store. If you wish to have a data store that's optimized for data retrieval, and hence optimized for fast data querying, you choose QOPT (Query Optimization). If you wish to have a data store that's optimized for maximum saving in storage space, you choose COPT (Compression Optimization).

Field and pattern level compression are used for both QOPT and COPT compression to give the basic level of compression. If QOPT is chosen then only the algorithmic compression is applied before the information is stored on disk. If COPT is chosen then the most appropriate byte level compression is also applied. When queried, RainStor reverses the algorithmic and byte level compression (if applied) so that the query is made on the compressed field and pattern level information. The application of field and pattern compression in memory is a benefit of RainStor, allowing less data to be stored during the query process and offering optimal processing.



How data is held on disk

During data import, no data files or trees are constructed in memory – everything is written to disk, first temporarily during a streaming process, and then committed transactionally when data import is completed successfully. The data is written to disk in partitions, each with its own compression strategy, and stored in what is in effect a logical table-based file structure within each partition.

It should be noted that the optimal design will depend entirely on the original dataset, and that multiple imports of data into a table within the data store will almost certainly result in RainStor partitions with different designs. This difference in tree design from partition to partition to obtain optimum compression is handled transparently by RainStor – the user does not need to concern themselves with how the tree is designed either at import or query time.