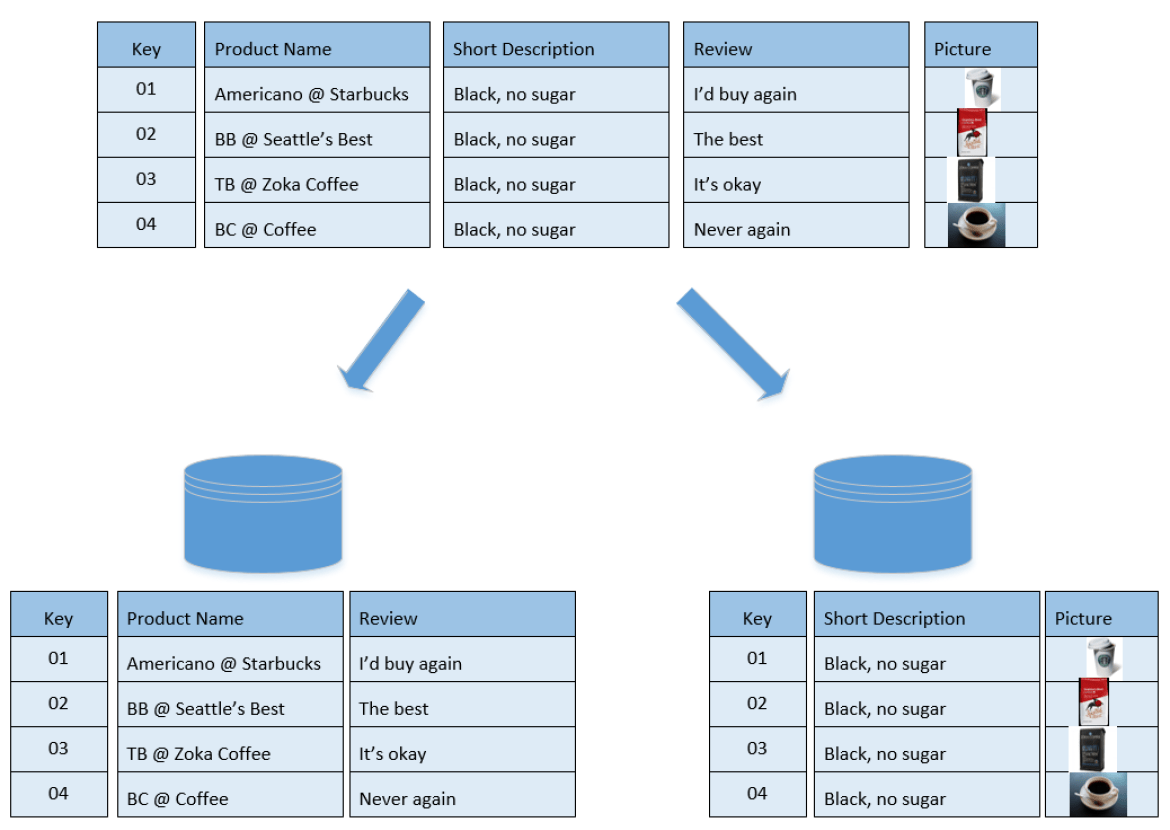
There are mainly three ways to partition your data.

**Vertical partitioning**

This type of partition divides the table vertically, which means that the structure of the main table changes in the new ones. An ideal scenario for this type of partition is when you don’t need all the information about the customer in your query. Let’s say if you only need orders from the current year, you could split it into two databases, one would hold customer information and current purchases, and the other would hold data about purchases from previous years. See Figure 2 for visual representation of a different scenario, where the user doesn’t need to see the short description of a product.

 Figure 2: Example of vertical partitioning

Another example where vertical partitioning is a great option is when you have different types of data in your database, such as names, dates, and pictures. You could keep the string values in SQL DB, and pictures in an Azure Blob. This would save you money because SQL is expensive and Blob storage is one of the cheapest storage options.

Vertical partitioning is straightforward to implement and has a low impact on the application. The main problem with this approach is that if our application experiences additional growth, then it may be necessary to further partition a feature specific DB across various servers (e.g. it would not be possible for a single server to handle all the metadata queries for 10 billion photos by 140 million users).

Vertical partitioning is very domain specific. You draw a logical split within your application data, storing them in different databases. It is almost always implemented at the ****application level**** — a piece of code routing reads and writes to a designated database.

**Horizontal partitioning – also known as sharding**

Here, you partition a table by rows. For example, if you have a large database of customers, you could divide it into four new tables: A – G, H – N, O – U, V – Z. However, this might not be the best optimization for you because, for instance, you have twice as many customers from the range O – U than from A – G. When partitioning your data, you need to assess the number of rows in the new tables, so each table has the same number of customers and will grow by a similar number of new customers in the future.

You might also partition the dataset based on the recent customers, for example, the clients that are not being active at your store are stored in one DB. In addition, the active customer database might be split into more tables, to get the results faster.

The structure of the original table stays the same in the new tables, i.e., we have the same number of columns. See Figure 3 for visual representation of the partition.

 sharding can be implemented at either the application or ****database level****.  A ****logical shard**** is a collection of data sharing the same partition key. A database node, sometimes referred as a ****physical shard****, contains multiple logical shards.

The key problem with this approach is that if the value whose range is used for sharding isn’t chosen carefully, then the partitioning scheme will lead to unbalanced servers. In the previous example, splitting location based on their zip codes assumes that places will be evenly distributed across the different zip codes. This assumption is not valid as there will be a lot of places in a thickly populated area like Manhattan compared to its suburb cities.

Operations may need to search through many databases to find the requested data. These are called ****cross-partition operations****and they tend to be inefficient. ****Hotspots**** are another common problem — having uneven distribution of data and operations. Hotspots largely counteract the benefits of sharding.

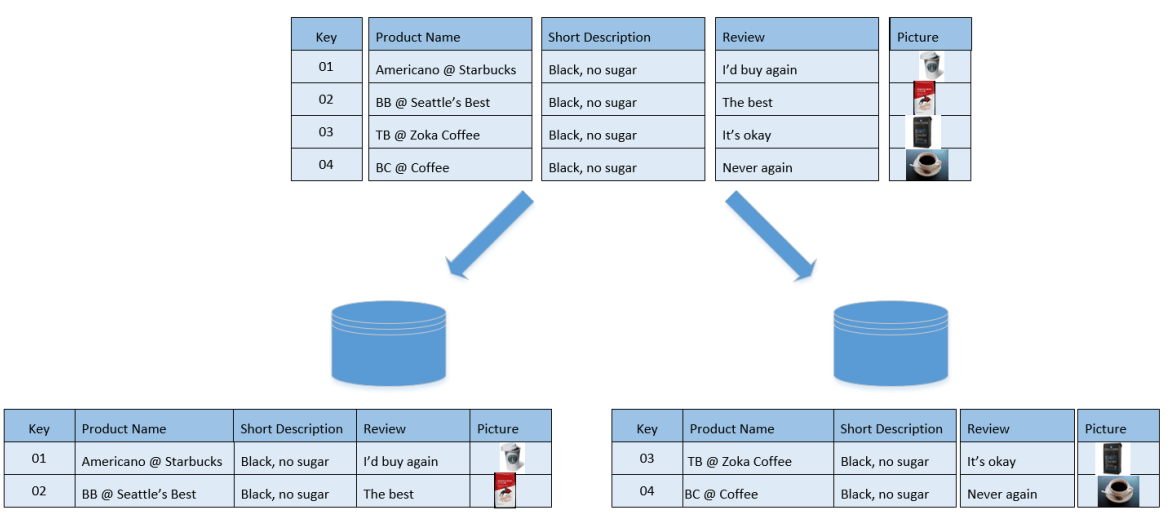


Figure 3: Example of horizontal partitioning

**Hybrid partitioning**

This division combines vertical and horizontal partitioning. If you have a large dataset where you keep different types of data, you could horizontally partition the customer information and vertically divide the database into string values based on your criteria in a SQL DB, and pictures could be stored in Blob storage. See Figure 4 for visual representation of the partition.

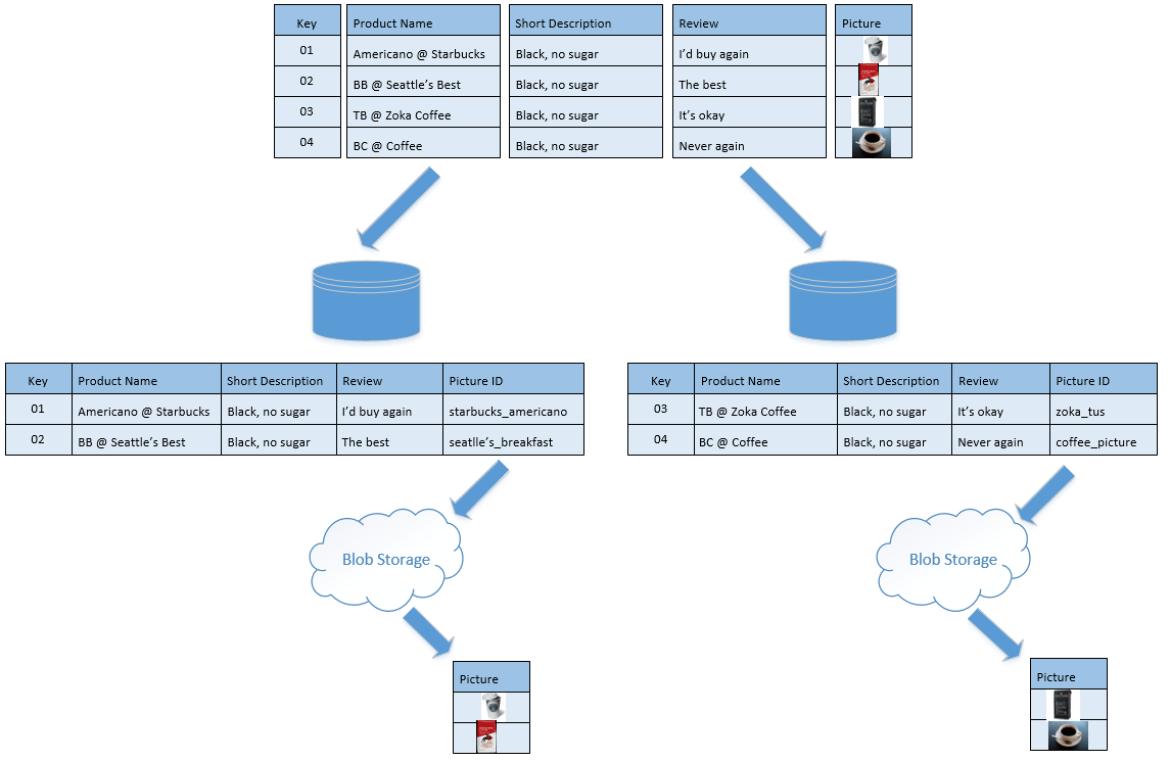


Figure 4: Example of hybrid partitioning

So, to pick the best option for you, you need to analyze the type and the structure of the data you have.

As an alternative to data partitioning, you could use Big Data that would do the sharding for you, but BD has other restrictions that you need to be aware of, but this is out of scope of this article.