**Caching in distributed applications**

Distributed applications typically implement either or both of the following strategies when caching data:

* Using a private cache, where data is held locally on the computer that's running an instance of an application or service.
* Using a shared cache, serving as a common source which can be accessed by multiple processes and/or machines.

In both cases, caching can be performed client-side and/or server-side. Client-side caching is done by the process that provides the user interface for a system, such as a web browser or desktop application. Server-side caching is done by the process that provides the business services that are running remotely.

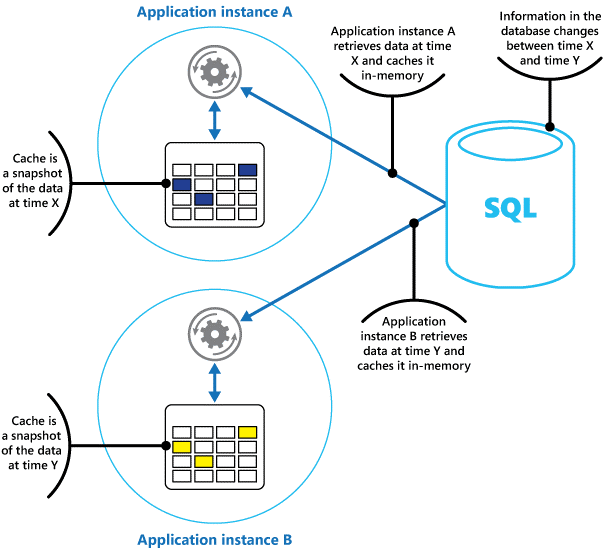
**Private caching**

The most basic type of cache is an in-memory store. It's held in the address space of a single process and accessed directly by the code that runs in that process. This type of cache is very quick to access. It can also provide an extremely effective means for storing modest amounts of static data, since the size of a cache is typically constrained by the volume of memory that's available on the machine hosting the process.

If you need to cache more information than is physically possible in memory, you can write cached data to the local file system. This will be slower to access than data that's held in-memory, but should still be faster and more reliable than retrieving data across a network.

If you have multiple instances of an application that uses this model running concurrently, each application instance has its own independent cache holding its own copy of the data.

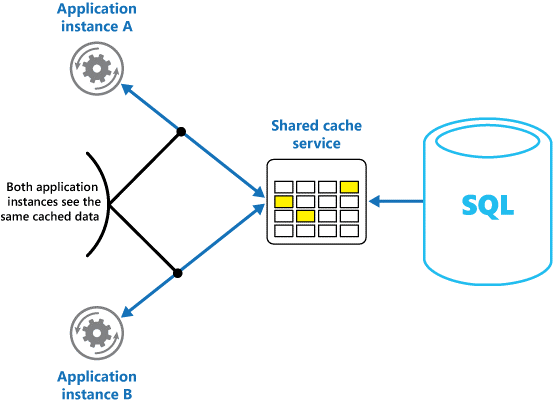
Think of a cache as a snapshot of the original data at some point in the past. If this data is not static, it is likely that different application instances hold different versions of the data in their caches. Therefore, the same query performed by these instances can return different results, as shown in Figure 1.



*Figure 1: Using an in-memory cache in different instances of an application*

**Shared caching**

Using a shared cache can help alleviate concerns that data might differ in each cache, which can occur with in-memory caching. Shared caching ensures that different application instances see the same view of cached data. It does this by locating the cache in a separate location, typically hosted as part of a separate service, as shown in Figure 2.



*Figure 2: Using a shared cache*

An important benefit of the shared caching approach is the scalability it provides. Many shared cache services are implemented by using a cluster of servers, and utilize software that distributes the data across the cluster in a transparent manner. An application instance simply sends a request to the cache service. The underlying infrastructure is responsible for determining the location of the cached data in the cluster. You can easily scale the cache by adding more servers.

There are two main disadvantages of the shared caching approach:

* The cache is slower to access because it is no longer held locally to each application instance.
* The requirement to implement a separate cache service might add complexity to the solution