**KEY-VALUE PAIR**

A key-value pair (KVP) is a set of two linked data items: a key, which is a unique identifier for some item of data, and the value, which is either the data that is identified or a pointer to the location of that data.

**KEY-VALUE STORES**

Key-value stores are probably the **simplest** form of [database management systems](http://db-engines.com/en/article/Database+Management+System). They can **only store pairs of keys and values**, as well as retrieve values when a key is known.

These simple systems are normally not adequate for complex applications. On the other hand, it is exactly this simplicity, that makes such systems attractive in certain circumstances. For example resource-efficient key-value stores are often applied in embedded systems or as high performance in-process databases.

The way it works is demonstrated in this little sample chart:

Color Red

Age 18

Size Large

Name Smith

Title The Brown Dog

Where you have a key (left) and a value (right) ... notice it can be a string, int, or the like. Most KVP objects allow you to store any object on the right, because it's just a value.

Since you'll always have a unique key for a particular object that you want to return, you can just query the database for that unique key and get the results back from whichever node has the object (this is why it's good for distributed systems, since there's other things involved like polling for the first n nodes to return a value that match other nodes returns).

Now my example above is very simple, so here's a slightly better version of the KVP

user1923\_color Red

user1923\_age 18

user3371\_color Blue

user4344\_color Brackish

user1923\_height 6' 0"

user3371\_age 34

So as you can see the simple key generation is to put "user" the userunique number, an underscore and the object. Again, this is a simple variation, but I think we begin to understand that so long as we can define the part on the left and have it be consistently formatted, that we can pull out the value.

Notice that there's no restriction on the key value (ok, there can be some limitations, such as text-only) or on the value property (there may be a size restriction) but so far I've not had really complex systems. Let's try and go a little further:

app\_setting\_width 450

user1923\_color Red

user1923\_age 18

user3371\_color Blue

user4344\_color Brackish

user1923\_height 6' 0"

user3371\_age 34

error\_msg\_457 There is no file %1 here

error\_message\_1 There is no user with %1 name

1923\_name Jim

user1923\_name Jim Smith

user1923\_lname Smith

Application\_Installed true

log\_errors 1

install\_path C:\Windows\System32\Restricted

ServerName localhost

test test

test1 test

test123 Brackish

devonly

wonderwoman

value key

You get the idea... all those would be stored in one massive "table" on the distributed nodes (there's math behind it all) and you would just ask the distributed system for the value you need by name.

**EMBEDDED SYSTEM**

An **embedded system** is a [computer](http://en.wikipedia.org/wiki/Computer) [system](http://en.wikipedia.org/wiki/System) with a dedicated function within a larger mechanical or electrical system, often with [real-time computing](http://en.wikipedia.org/wiki/Real-time_computing) constraints.[[1]](http://en.wikipedia.org/wiki/Embedded_system#cite_note-Barr-glossary-1)[[2]](http://en.wikipedia.org/wiki/Embedded_system#cite_note-2) It is *embedded* as part of a complete device often including hardware and mechanical parts. Embedded systems control many devices in common use today.

**IN-DATABASE PROCESSING**

**in-database processing**, sometimes referred to as in-database analytics, refers to the integration of data [analytics](http://en.wikipedia.org/wiki/Analytics) into [data warehousing](http://en.wikipedia.org/wiki/Data_warehousing) functionality. Today, many large databases, such as those used for credit card fraud detection and investment bank risk management, use this technology because it provides significant performance improvements over traditional methods.