```
3  NaN  B3  C3
4  NaN  B4  C4
```

The combination of options of the `pd.concat` function allows a wide range of possible behaviors when you are joining two datasets; keep these in mind as you use these tools for your own data.

The append() method

Because direct array concatenation is so common, `Series` and `DataFrame` objects have an `append` method that can accomplish the same thing in fewer keystrokes. For example, rather than calling `pd.concat([df1, df2])`, you can simply call `df1.append(df2)`:

```
In[16]: print(df1); print(df2); print(df1.append(df2))

df1               df2              df1.append(df2)
    A   B             A   B             A   B
1  A1  B1          3  A3  B3         1  A1  B1
2  A2  B2          4  A4  B4         2  A2  B2
                                    3  A3  B3
                                    4  A4  B4
```

Keep in mind that unlike the `append()` and `extend()` methods of Python lists, the `append()` method in Pandas does not modify the original object—instead, it creates a new object with the combined data. It also is not a very efficient method, because it involves creation of a new index *and* data buffer. Thus, if you plan to do multiple `append` operations, it is generally better to build a list of `DataFrames` and pass them all at once to the `concat()` function.

In the next section, we'll look at another more powerful approach to combining data from multiple sources, the database-style merges/joins implemented in `pd.merge`. For more information on `concat()`, `append()`, and related functionality, see the "Merge, Join, and Concatenate" section of the Pandas documentation.

Combining Datasets: Merge and Join

One essential feature offered by Pandas is its high-performance, in-memory join and merge operations. If you have ever worked with databases, you should be familiar with this type of data interaction. The main interface for this is the `pd.merge` function, and we'll see a few examples of how this can work in practice.

Relational Algebra

The behavior implemented in `pd.merge()` is a subset of what is known as *relational algebra*, which is a formal set of rules for manipulating relational data, and forms the conceptual foundation of operations available in most databases. The strength of the

relational algebra approach is that it proposes several primitive operations, which become the building blocks of more complicated operations on any dataset. With this lexicon of fundamental operations implemented efficiently in a database or other program, a wide range of fairly complicated composite operations can be performed.

Pandas implements several of these fundamental building blocks in the `pd.merge()` function and the related `join()` method of `Series` and `DataFrames`. As we will see, these let you efficiently link data from different sources.

Categories of Joins

The `pd.merge()` function implements a number of types of joins: the *one-to-one*, *many-to-one*, and *many-to-many* joins. All three types of joins are accessed via an identical call to the `pd.merge()` interface; the type of join performed depends on the form of the input data. Here we will show simple examples of the three types of merges, and discuss detailed options further below.

One-to-one joins

Perhaps the simplest type of merge expression is the one-to-one join, which is in many ways very similar to the column-wise concatenation seen in "Combining Datasets: Concat and Append" on page 141. As a concrete example, consider the following two `DataFrames`, which contain information on several employees in a company:

```
In[2]:
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})
print(df1); print(df2)

df1                          df2
  employee        group        employee  hire_date
0      Bob   Accounting     0      Lisa       2004
1     Jake  Engineering     1       Bob       2008
2     Lisa  Engineering     2      Jake       2012
3      Sue           HR     3       Sue       2014
```

To combine this information into a single `DataFrame`, we can use the `pd.merge()` function:

```
In[3]: df3 = pd.merge(df1, df2)
       df3

Out[3]:   employee        group  hire_date
       0       Bob   Accounting       2008
       1      Jake  Engineering       2012
       2      Lisa  Engineering       2004
       3       Sue           HR       2014
```