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
```

The pd.merge() function recognizes that each DataFrame has an "employee" column, and automatically joins using this column as a key. The result of the merge is a new DataFrame that combines the information from the two inputs. Notice that the order of entries in each column is not necessarily maintained: in this case, the order of the "employee" column differs between df1 and df2, and the pd.merge() function correctly accounts for this. Additionally, keep in mind that the merge in general discards the index, except in the special case of merges by index (see "The left_index and right_index keywords" on page 151).

Many-to-one joins

Many-to-one joins are joins in which one of the two key columns contains duplicate entries. For the many-to-one case, the resulting DataFrame will preserve those duplicate entries as appropriate. Consider the following example of a many-to-one join:

```
In[4]: df4 = pd.DataFrame({'group': ['Accounting', 'Engineering', 'HR'],
                           'supervisor': ['Carly', 'Guido', 'Steve']})
       print(df3); print(df4); print(pd.merge(df3, df4))
```

```
df3                                    df4
  employee        group  hire_date               group supervisor
0      Bob   Accounting       2008     0    Accounting      Carly
1     Jake  Engineering       2012     1   Engineering      Guido
2     Lisa  Engineering       2004     2            HR      Steve
3      Sue           HR       2014

pd.merge(df3, df4)
  employee        group  hire_date supervisor
0      Bob   Accounting       2008      Carly
1     Jake  Engineering       2012      Guido
2     Lisa  Engineering       2004      Guido
3      Sue           HR       2014      Steve
```

The resulting DataFrame has an additional column with the "supervisor" information, where the information is repeated in one or more locations as required by the inputs.

Many-to-many joins

Many-to-many joins are a bit confusing conceptually, but are nevertheless well defined. If the key column in both the left and right array contains duplicates, then the result is a many-to-many merge. This will be perhaps most clear with a concrete example. Consider the following, where we have a DataFrame showing one or more skills associated with a particular group.

By performing a many-to-many join, we can recover the skills associated with any individual person:

```
In[5]: df5 = pd.DataFrame({'group': ['Accounting', 'Accounting',
                                     'Engineering', 'Engineering', 'HR', 'HR'],
                           'skills': ['math', 'spreadsheets', 'coding', 'linux',
                                      'spreadsheets', 'organization']})
print(df1); print(df5); print(pd.merge(df1, df5))
```

```
df1                          df5
  employee        group            group        skills
0      Bob   Accounting      0   Accounting          math
1     Jake  Engineering      1   Accounting  spreadsheets
2     Lisa  Engineering      2  Engineering        coding
3      Sue           HR      3  Engineering         linux
                             4           HR  spreadsheets
                             5           HR  organization


pd.merge(df1, df5)
  employee        group        skills
0      Bob   Accounting          math
1      Bob   Accounting  spreadsheets
2     Jake  Engineering        coding
3     Jake  Engineering         linux
4     Lisa  Engineering        coding
5     Lisa  Engineering         linux
6      Sue           HR  spreadsheets
7      Sue           HR  organization
```

These three types of joins can be used with other Pandas tools to implement a wide array of functionality. But in practice, datasets are rarely as clean as the one we're working with here. In the following section, we'll consider some of the options provided by `pd.merge()` that enable you to tune how the join operations work.

Specification of the Merge Key

We've already seen the default behavior of `pd.merge()`: it looks for one or more matching column names between the two inputs, and uses this as the key. However, often the column names will not match so nicely, and `pd.merge()` provides a variety of options for handling this.

The on keyword

Most simply, you can explicitly specify the name of the key column using the `on` keyword, which takes a column name or a list of column names:

```
In[6]: print(df1); print(df2); print(pd.merge(df1, df2, on='employee'))
```

```
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
```