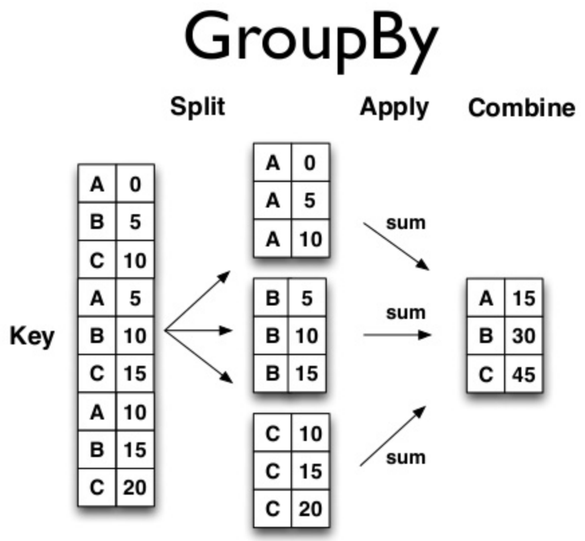
The *GroupBy* Pattern

The *GroupBy* Pattern

We have seen the ***GroupBy*** operator in ***Pandas***, but this is actually a more general ***design pattern*** that can be utilized in many data analyics frameworks and data access interfaces, e.g. in ***SQL***.

GroupBy: general Pattern



GroupBy in SQL:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

GroupBy in SQL:

```
SELECT COUNT(CustomerID), Country
FROM Customers
GROUP BY Country
ORDER BY COUNT(CustomerID) DESC;
```

Pandas SQL-Query example

```
params=tuple([202101, 202102, 202103])

sql = f""" SELECT p.description, SUM(p.price) as price
           FROM product p
           WHERE p.extenal_id in {params}
           GROUP BY p.description """
result = pd.read_sql_query(sql, con=conn)
```

GroupBy in Pandas

In [1]:
```
#setup example
import numpy as np
import pandas as pd
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                   'key2' : ['one', 'two', 'one', 'two', 'one'],
                   'data1' : np.random.randn(5),
                   'data2' : np.random.randn(5)})
df
```

Out[1]:

	key1	key2	data1	data2
0	a	one	1.221437	-1.517520
1	a	two	1.689427	1.246337
2	b	one	-0.953992	-0.657989
3	b	two	-0.470554	-2.072804
4	a	one	0.106501	-1.576961

In [2]:
```
#group by key1
grouped = df.groupby(df['key1'])
grouped #this is now a more complex group object
```

Out[2]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7f1788503c90>
```

In [3]:
```
#and generates a table per group
for name, group in grouped:
    print ("name:", name, "\n",group)
```

```
name: a
   key1 key2     data1     data2
0    a  one  1.221437 -1.517520
1    a  two  1.689427  1.246337
4    a  one  0.106501 -1.576961
name: b
   key1 key2     data1     data2
2    b  one -0.953992 -0.657989
3    b  two -0.470554 -2.072804
```

In [4]:
```
#access group table
grouped.get_group('b')
```

Out[4]:

	key1	key2	data1	data2
2	b	one	-0.953992	-0.657989
3	b	two	-0.470554	-2.072804

In [5]:
```
#get numper of entries (rows) per group
grouped.size()
```

Out[5]:
```
key1
a    3
b    2
dtype: int64
```

In [6]:
```
#get number of group entries by columns
grouped.count()
```

Out[6]:

	key2	data1	data2
key1			
a	3	3	3
b	2	2	2

Think of grouped DataFrames as 3d objects:

In [7]:
```
#accessing the "3d" group tables
grouped['data2'].get_group('a')
```

Out[7]:
```
0   -1.517520
1    1.246337
4   -1.576961
Name: data2, dtype: float64
```

In [8]:
```
grouped.get_group('a')['data2']
```

Out[8]:
```
0   -1.517520
1    1.246337
4   -1.576961
Name: data2, dtype: float64
```

Group by external keys

Group by external keys

In [9]:
```
#define external key years as numpy array
years = np.array([2005, 2005, 2006, 2005, 2006])
df['data1'].groupby([years]).mean()
```

Out[9]:
```
2005    0.813437
2006   -0.423746
Name: data1, dtype: float64
```

Group by functions

Group by functions

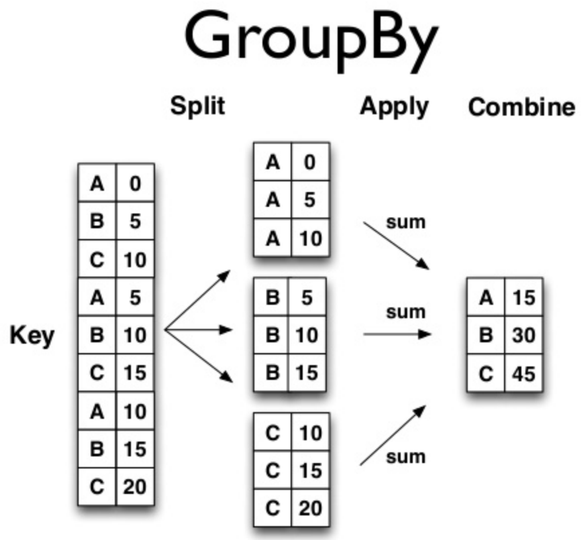
In [10]:
```
#sort by column and retun top n
def top(df, n=5, column='data1'):
    return df.sort_values(by=column)[-n:]

df.groupby(df['key1']).apply(top,n=5)
```

Out[10]:

		key1	key2	data1	data2
key1					
a	4	a	one	0.106501	-1.576961
	0	a	one	1.221437	-1.517520
	1	a	two	1.689427	1.246337
b	2	b	one	-0.953992	-0.657989
	3	b	two	-0.470554	-2.072804

Group-wise aggregation (apply)



Typical build in aggregation functions:

- sum
- mean
- max / min
- quantile
- ...

Typical build in aggregation functions:

- sum
- mean
- max / min
- quantile
- ...

In [11]:
```
#aggregate over the groups
grouped.count()
```

Out[11]:

	key2	data1	data2
key1			
a	3	3	3
b	2	2	2

In [12]:
```
grouped.sum()
```

Out[12]:

key1	data1	data2
a	3.017365	-1.848144
b	-1.424545	-2.730793

Custom Aggregation Functions

In [13]:
```
def peak_to_peak(arr):
    return arr.max() - arr.min()
grouped.agg(peak_to_peak)
```

Out[13]:

	data1	data2
key1		
a	1.582926	2.823297
b	0.483438	1.414815

Multiple aggregations

In [14]:
```
#just call a list of function
grouped.agg([peak_to_peak, 'mean', 'median'])
```

Out[14]:

	data1			data2		
	peak_to_peak	mean	median	peak_to_peak	mean	median
key1						
a	1.582926	1.005788	1.221437	2.823297	-0.616048	-1.517520
b	0.483438	-0.712273	-0.712273	1.414815	-1.365397	-1.365397

Suppressing the Group Keys

In [15]:
```
df.groupby(df['key1']).apply(top,n=2)
```

Out[15]:

		key1	key2	data1	data2
key1					
a	**0**	a	one	1.221437	-1.517520
	1	a	two	1.689427	1.246337
b	**2**	b	one	-0.953992	-0.657989
	3	b	two	-0.470554	-2.072804

In [16]:
```
df.groupby(df['key1'], group_keys=False).apply(top,n=2)
```

Out[16]:

	key1	key2	data1	data2
0	a	one	1.221437	-1.517520
1	a	two	1.689427	1.246337
2	b	one	-0.953992	-0.657989
3	b	two	-0.470554	-2.072804

More Exercises in the Lab session...

```
In [ ]:
```