**属性图**

属性图（Property Graph）是一张有向多边图（directed multigraph），顶点和边都可以有自定义的属性。顶点有编号和顶点属性，边只有属性，不过边可以连接源顶点和目标顶点。

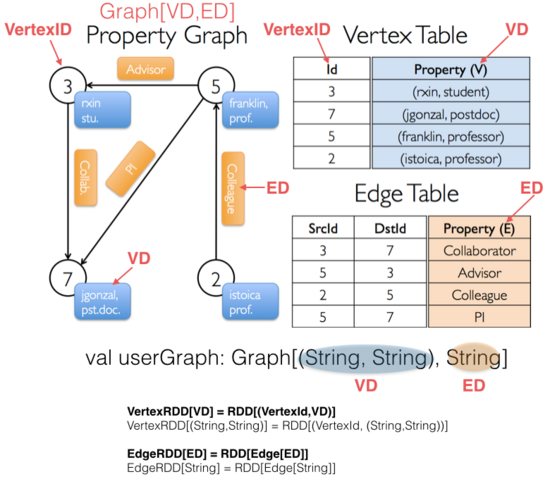
一张图当然会有很多顶点和很多边组成，而Spark的RDD就类似于一个集合，所以图可以用

* **RDD[(VertexID, VD)]** 表示一系列的顶点
* **RDD[Edge[ED]]** 表示一系列的边

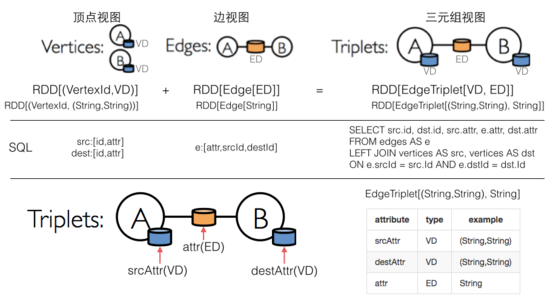
VertexRDD和EdgeRDD是对上面两个数据结构更高级的封装。VD和ED作为Graph的类型参数表示的是顶点和边的数据，   
为什么在图中没有VertexID呢，因为顶点编号肯定是必须的！但是顶点数据和边数据则是自定义的。泛型就是为了自定义。   
所以我们可以肯定VertexRDD一定包含了VertexID，而VD是不会包含VertexID的。

**class** **Graph**[**VD**, ED] {  
 val vertices: VertexRDD[VD] // RDD[(VertexID, VD)] 顶点编号，顶点数据（属性）  
 **val** edges: EdgeRDD[ED] // RDD[Edge[ED]] 边数据（属性）  
}

属性图可以用顶点表和边表来表示，边表有两个外键引用了顶点表的主键，分别表示边的源顶点和目标顶点。顶点和边都可以有自己的属性。



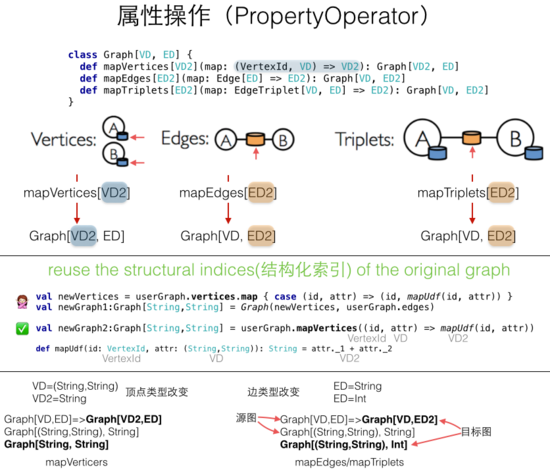
通过图可以获取顶点视图和边视图，除此之外还提供了triplet三元组，这样通过triplet也可以获得顶点（源顶点和目标顶点）的属性



属性图，提供的操作就是属性操作（Property Operator），可以操作顶点的属性和边的属性。但是顶点的编号，以及边的源编号和目标编号都是不能操作的，因为更改了编号，图就完全变样了。

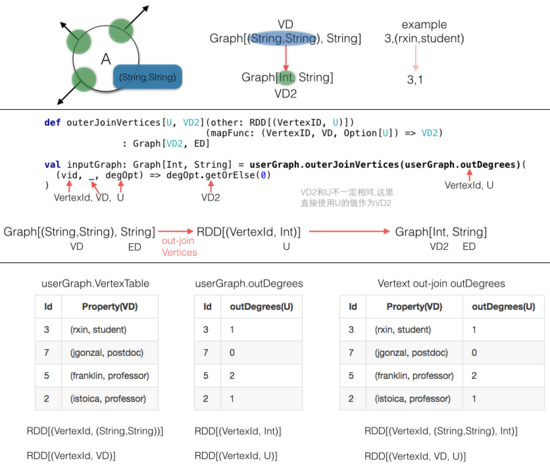
Triplet实际上是继承了Edge，原先的边只有srcId和destId，现在加上了srtAttr和destAttr，信息更加丰富。

* mapVertices：对图的Vertices做操作，原图的VD类型通过map函数转换为新图的VD2类型
* mapEdges/mapTriplets：对图的顶点做操作，原图的ED类型通过map函数转换为新图的ED2类型



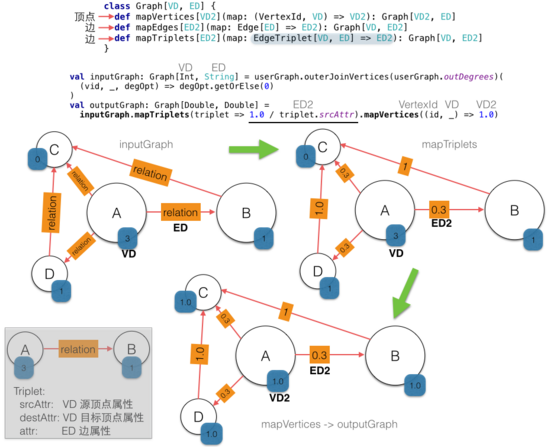
outerJoinVertices：顶点本身是一张表，主键是顶点编号，其他属性作为普通的列比如名字和职位组成VD：(String,String)，   
可以对顶点边转换成新的表，不过新的表的主键也一定仍然是顶点编号，其他列比如只有一个：统计顶点的入度，VD=Int   
将原图（的顶点表）和新的顶点表进行out-join，相同顶点编号作为一行，剩余一起join的是原表的列，和新表的列。

**SELECT** v.id, v1.attr, v2.outDegree **from** vertices v1  
OUT **JOIN** (  
 **select** **id**, **count**(outDegrees) **as** outDegree **from** vertices  
) v2 **ON** v1.id = v2.id



下面的示例是构造一张PageRank图的初始步骤，每条边包含权重（入度的倒数），每个顶点初始值为1.0。

| **Triplet** | **Type** | **A-[relation]-B** |  | **mapTriplets** | **example** |  | **mapVertices** | **example** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| srcAttr | VD | 3 |  | VD | 3 |  | VD2 | 1.0 |
| destAttr | VD | 1 |  | VD | 1 |  | VD2 | 1.0 |
| attr | ED | professor |  | ED2 | 1/3=0.3 |  | ED2 | 0.3 |



joinVertices详解：



outJoinVertices详解：