Project-9

Akanksha Bansal ([akankshabansal90@gmail.com](mailto:akankshabansal90@gmail.com))

Gaurav Nanda ([gaurav324@gmail.com](mailto:gaurav324@gmail.com))

Prateek Aggarawal ([prat0318@gmail.com](mailto:prat0318@gmail.com))

* *And you should have a substantial write-up of what you have done, how you have proceeded, and your organization.*

We started by implementing ReadRelation component that reads a text file and produces a stream of tuples. Instead of using Java's **StringTokenizer,** we are using regular expression to tokenize the tuples.

Next we proceeded to implement PrintTuple class, which prints a stream of tuples to standard output.

After having a basic Connecter/Reading/Writing infrastructure, we spent some time in understanding the BMap class. It was not very intuitive in the earlier phases that we could split a BMap and ensures the correctness of the joins. A re-examination of the code led us to understand that each row in BMap has data for separate thread. Also, we are using hash keys to split data among threads using same criteria as BMap, correctness would always be ensured.

Bloom :

Hsplit: Splitting based of the hash of the Key allows us to slip the data into streams which are distinct and the streams can then be merged based of that field as well.

Basically splitting based of the hash of key allows us to join the data from

Then we started implementing all the components (inheriting from Thread), present in the final design (as discussed in the class). After developing each and every component, we wrote its sample test case in **“MainTest.java”**. This test code is not simile to regular java test cases, like we don’t assert output against a fixed output. This was primarily written by developers to ensure the integrity of their component.

In this first version of the code, we were not using Relations class properly and were ignoring the headers present in the database files. We later introduced Relations in the final version of code. There were so many things going on initially, this approach just made the development simpler.

Once we had enough confidence in our Thread components, we moved on to implement the pipe-and-filter graphs, which implement the functionality of primitive boxes. Though these classes are supposedly inheriting from ArrayConnectors class, we did find much to inherit from ArrayConnectors class. Instead, we are using ArrayConnectors as an attribute in these classes. Still, we are not using any fancy functions of this class and just using it to represent a collection of Connectors.

Also classes like MapReduceBFilter should ideally be able to replace BFilter seamlessly. However, BFilter is a thread and MapReduceBFilter is not. But it also seemed correct to not make MapReduceBFilter as a thread, because all its components were already threads. In the final solution, we wrote “start()” method inside MapReduceBFilter, which starts all its components and exists. In this way, we can use MapReduceBFilter and BFilter interchangeably.

Testing infrastructure.