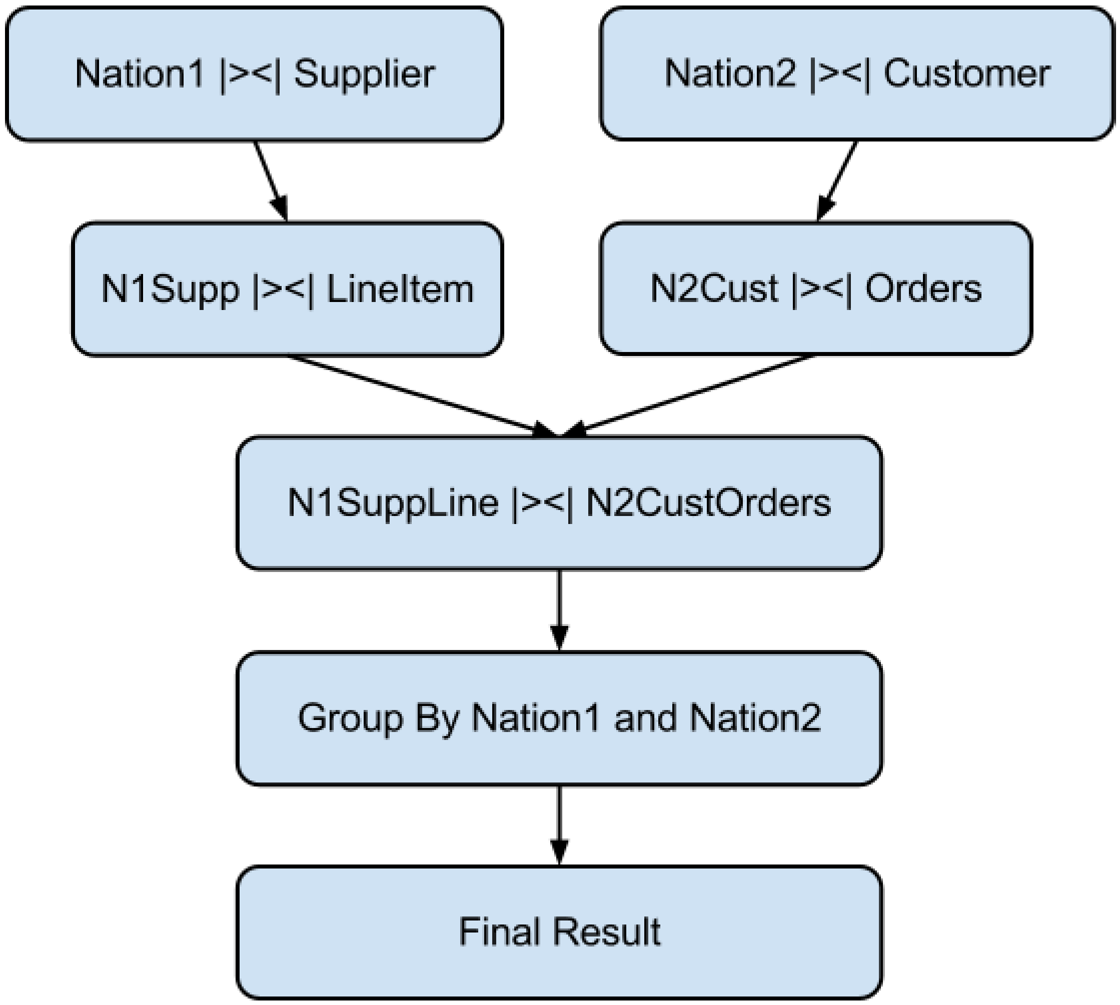
Advanced Databases Project 1 Report

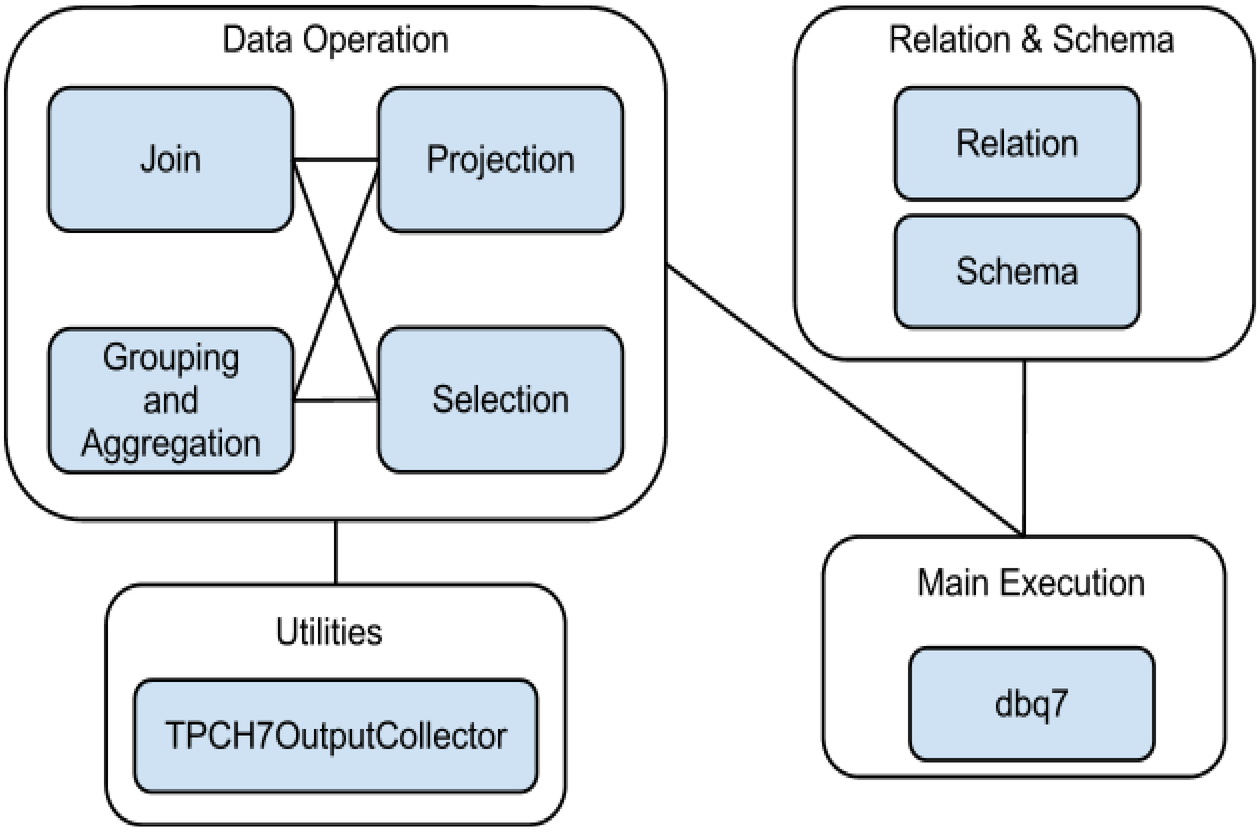
Team 8

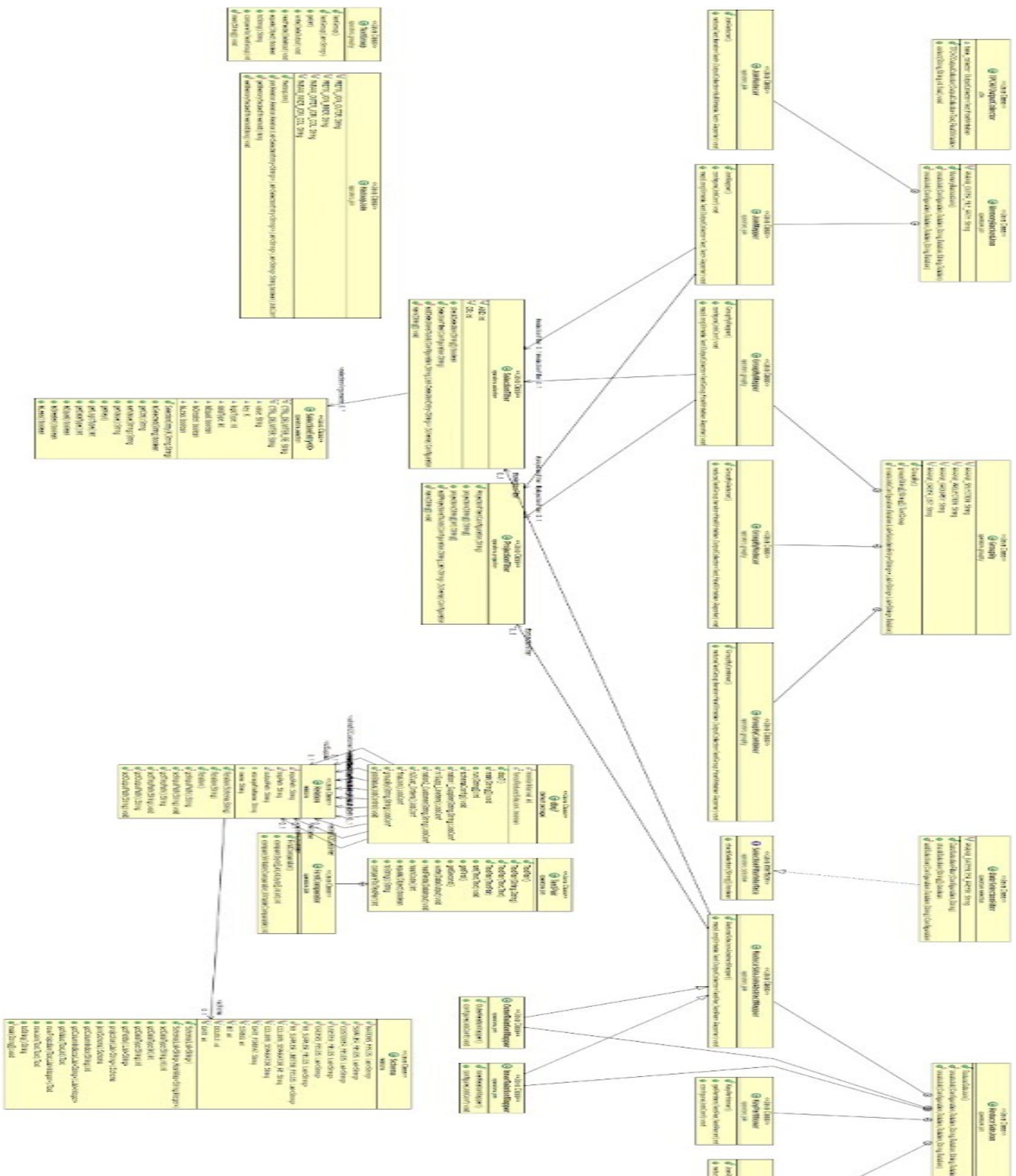
Diggory Hardy
Diwakar Sapan
Elmas Ferhat
Goyal Pulkit
Tran Le Hung
Vidmantas Zemleris

Query Execution



Architecture





Database Operations

Joins

Reduce Side Join

A robust and our default join method which is useful when the relations are quite big and don't fit in the main memory.

The idea behind this is using a composite key that contains join attribute and relation name. A custom comparator is implemented for the composite key. The comparator ensures tuples will be sorted by join attribute first, then those of the outer relation are always ordered before those of inner relation. In order to make sure that all tuples with the same join attribute are passed to the same reducer, we have to implement a custom partitioner and a custom comparator that checks only the join attribute to identify the equality of keys. The actual join is then implemented in the reducer.

Memory Backed Join

When one of two relations fits completely into main memory, memory-backed join can be used to overcome the overhead of shuffling the tuples over the network. In this case, we can load the smaller dataset into the main memory of all "mapper processes" (during the configuration phase) and have the mappers perform the join by simply looping over all the tuples in the larger relation. This is the same concept as simple-hash join in a non-distributed environment.

To improve the efficiency of the query, we implemented a generalized join operator that can look at the size of the relations to be joined and then applies the appropriate method (i.e. memory-backed or reduce-side) on them to obtain the result. We also make sure that the smaller relation (in term of data size) will always be the outer relation in join operation.

Projection and Selection

Projection

Projection takes column indexes to be projected and accordingly puts relevant columns into an array and returns it.

Selection

Selection is flexible by a general implementation. Boolean operands and operators are created for all columns' selection conditions. The selection process checks each tuple of the relation against all the selection conditions and creates a boolean expression. Then, the boolean expression is computed by a simple function regarding to the priority of boolean operators. If the result is false, the tuple is discarded, otherwise the tuple is put into the result set. The selection also considers the data type of the column when comparing the value.

Selections and projections is executed in the map and reduce phases of other operations in order to apply these operations as early as possible to save the cost of transmitting unrequired data over the network.

Group By, Aggregation and Order By

We have implemented the grouping, aggregation and ordering operators in one map-reduce job. The group-by columns are specified as configuration options before running the job.

Since the reshuffle phase of Hadoop automatically groups the keys that the mappers produce and hands them as input to the same reduce function, the group by operation is implicit between the maps and reduce phase. Moreover, the intermediate data arrive at the reducer in order, sorted by the key, thereby making the order by implicit. Therefore, in order to implement these operations, we use a composite key which consists of all the *group-by* attributes. A custom comparator is implemented to sort the tuples according to the *order-by* attributes.

The aggregation can then be easily loaded into the reduce job as it deals only with tuples which have the same key for all the *group-by* attributes and therefore would be passed to the same reduce function. Here, we don't face any problems with available memory as because of associativity of SUM() aggregate used we only need to store one tuple at a time in main memory.

Relation and Schema

In addition to the basic operators for query, we have also implemented the classes for handling the schema and relational metadata. These classes help us to decouple the implementation of map-reduce jobs from the database schema and therefore allows our program to easily adapt to new schemas. Using these classes, we can easily manage the input, output and intermediate schemas. The schemas of intermediate and output results are defined before projection and join operations and sent to job configuration. These classes also keep track of the relations in HDFS.