Time spent on the project: 9hours.
Group task division: coding and report done 50/50 by the 2 group members.

JoinOptimizer Class

- EstimateJoinCost: we built the second condition where the logical join node is false (so no subquery). In this case the join cost correspond to the estimated cost of the scan on the left-hand side of the query "cost1", added with the product of the left-side cardinality "card1" and cost of the scan of the right-hand side of the query "cost2", added again to the product of the left-side cardinality "card1" and the right-side cardinality "card2".
- EstimateTableJoinCardinality: To estimate the join cardinality of two tables we first have to check if the join operator use the predicate operator of equality.
  - If not the result is simply the product of the cardinalities of the tables.
  - If yes we must check if one of the 2 tables is a primary key table and return the cardinality of the table that is not primary. If none of them is primary table we must take the maximum cardinality between the 2 tables.
- OrderJoins: We compute the cost and cardinality of each subplan in each join node using multiple nested loops. First we loop on the index of the join nodes list. A second nested loop on the set inside a subset of join nodes (selected by the previous index loop). And finally a last loop that go throught each node of the selected set and compute the cost and cardinality and select the best plan, cost and cardinality.

IntHistogram Class

- AddValue: add the value to a particular bucket. To find the correct index of the bucket related to that value we substract the minvalue from the value given in input and divide it by the number of unit value of buckets.
- EstimateSelectivity: Main component of the project**.** The structure is a switch case that select the right operator from the input given. For each operator we have different ways of estimating the selectivity, we use to compute the value 2 supplementary method called "estimateSelectivityEquals" and "estimateSelectivityInequality" (added from the original skeleton). Also a "findBucket" method has been added to optimize the code of the 2 additional methods to estimate the selectivity introduced before.(Code comment for more details)

TableStats Class (most part already done)

- EstimateScanCost: the cost is the number of pages times the cost per page IO.
- EstimateTableCardinality: the cardinality is the number of tuples times the selectity factor given as input.
- EstimateSelectivity: check if the tuple field type is an integer.
  - If yes I take the histogram element where the index is equal to the field value and return the estimate of the selectivity of the integer histogram obtained (simple call to the estimate selectivity method of the histogram class).
  - If not, that means that I have to istantiate a string histogram and call the relative method to estimate the selectivity and return.

Sources of informations: Javadoc, stackoverflow, github