Optimisation strategy:

For the optimisation of the query, the heuristic method shown in the lectures was followed. Firstly, the Query is parsed through the visit(Operator op) functions for all Operators' subclasses. For scans, relations are retrieved in a list, for projections the projected attributes are included in a list that populate the total attributes needed by the query. For selections, the predicate is collected in a predicate list that will form future selections if it is of the form attribute = value or joins if it the predicate is of the form attribute1=attribute2. Products are ignored as they are computationally expensive and would be the last resort of combining relations when a join is not possible. For joins, the predicate is collected. For all predicates collected, their attributes are included in the total attribute list mentioned before. As the query is now parsed, the first step is to push the selections as down the tree as possible. Thus, the source code traverses through the relations and the predicates and if there is a predicate of the form attribute=value that matches the attribute of a relation, a new selection with that predicate and that relation is created and added to the list. If for a relation no selection can be pushed, the relation is added in the list without a selection. Furthermore, the next step is to push possible projections down the tree as I am building the trees from bottom to top. If the number of attributes needed from an operator in the list after pushing selections is smaller than the number of all attributes of the relation, then a projection of these attributes is added to the operator else the operator is included without a projection. The next step is to create joins and determine their order. Therefore, in the optimising function the list consisting of all subtrees with relations and possibly selections and projections is traversed along with the predicate list that holds the predicates of the form attribute=attribute. When a the two attributes of the predicate (left and right) match two different relations (as self joins are not permitted), a new join is created and the two relations that were used for the join, are now removed from the list. If it was not possible to create a join, then combine the two relations (or subtrees that whose output is a relation) with a cross product. After a join or a cross product is created, check if a projection can be done (if fewer attributes are needed than those that populate the join or product). This process is on a loop until the list of subtrees only has one operator which means that there is a finalised query plan. The implementation has no actual heuristic for choosing joins just a brute force approach. If a join is applicable, then make one.