(1)
a) Describe the algorithms basic idea.
The algorithm is an implementation of Tane.
It uses position lists indidexes for functional depency checking. The initial plis for each column are generated at the start.
The algoriothm iterates through the lattice from bottom to top and uses pruning to reduce the number of functional dependency checks in the next lattice level.
To efficiently generate the next lattice level it uses the apriori idea.

b) If you used an algorithm from literature, provide a reference to the according publication.

We implemented the TANE algorithm from the slides.

c) Provide one or two arguments why it is or could be better than related algorithms.

The algorithm uses pruning to reduce the number of checked functional dependencies so it is better than a naive brute force approach.

d) If your algorithm implements an adption of optimization of existing approaches, describe these briefly.

We stripp the position lists indixes to reduce storage of plis and runtime of insersect of two plis.

(2)
a) How many unique column combinations did your algorithm find on the provided datasets?

Input data set	Super FD	Fy FD
WDC Satellites	35	35
WDC Planets	66	66
NC Voter 1k	759	759
FD Reduced 30	89571	89571

b) How long did it take?

Used hardware: 2,6 Ghz Intel Core i7, 16GB DDR 3, Mac OS 10.12

Input data set	Super FD	Fy FD
WDC Satellites	50ms	49ms
WDC Planets	30ms	36ms
NC Voter 1k	677ms	213ms
FD Reduced 30	312ms	179ms

c) Did you discover any limitations of your approach?

Our approach is limited by memory for datasets with many columns as we store all subsets of R with size n at the nth iteration. It is also limited by time as the generation of all functional

dependencies of the nc voter set would have taken many days as can be seen in the slides.