Our Process:

We believe the best way to approach this task, given the amount of data we had to process, was to work with each dataframe individually and perform as many of the necessary transformations as possible on that dataframe before the join.

We started off by getting rid of unnecessary columns.
Then there were some values that needed to be fixed (e.g., the letters in the income column of the demographic_data dataframe) so we dealt with it.
Afterwards we started working on new columns that will be helpful to check the conditions.
We worked on each dataframe individually for as long as possible to take advantage of the spark caching.

After joining the tables and completing the last conditions (those who require data from a few separate dataframes), we were left off with a dataframe containing all program-codes, and for each program code, which conditions he satisfies. These were the columns of our combined post-transformations dataframe:

-['prog_code', 'cond_1', 'cond_2', 'cond_3', 'cond_4', 'cond_5', 'cond_6']

for each prog-code and condition, the respected cell in the dataframe contained the value 1 if the condition of satisfied by the program, and 0 otherwise.

Lastly (2.2) we got the program-codes of all the columns that answer 4 or more of these conditions and displayed the results as requested.

All the transformations are clearly documented in the code files.

Answer to question 1.2.1:

Another solutino to the problem:

- start of by working with each scheme individually, performing as much of the necessary operations on the separate tables.
- establishing clear logical connections between the different datasets and join them when necessary.

Pros & Cons

Pro 1: if we combine all the data into one table early on, It will be very easy to work with because all the qualities are in one table, so you can access all the necessary properties of each record without alternating between different tables and joining.
Pro 2: combining all the data into one big table means we will only use that table throughout the entire process, which means we can cache/persist it early on, and we won't have to worry about memory management.

Con 1: Joining all the tables when they still contain all the records (before filtering anything out) can be very heavy computationally. Addtionally, naturally this table will have alot more records, meaning with each operation such as filtering, we will need to go through all the records in the big table, even if using one of the sub-tables would have been sufficient.
Con 2: Increased complexity: Joining all the data into one table requires mapping and integrating different data structures, formats, aliases and naming conventions. Needless to say this can be very complex when dealing with datasets that encompass a vast array of diverse properties.