Setup

The application was developed by using QT Creator 3.0.1; for the QT 5.3.2 - 64bit toolchain it was used gcc 64bit as compiler.
The Qt IDE and toolchain can be downloaded from here:
http://qt-project.org/downloads (Qt 5.3.2 for Linux 64-bit (447 MB) & Qt creator 3.2.1 for linux/ X11 64bit)

No other library is needed to run the application.

If Qt Creator doesn't recognize the new toolchain (5.3.2) automatically you need to manually add it.
In order to insert the toolchain manulally, the following steps are needed:
Select Projects->Build&Run-> manageKits
In "Qt Version" tab - click Add and select the corresponednt qmake location (e.g. /opt/Qt5.3.2/5.3/gcc_64/bin/qmake)
Then go to "Kit" tab and click Add button. the parameter to select are
-Device type "desktop"
-Device "local pc"
-sysroot empty
-compiler "GCC(x86 64bit..)"
-Debugger "system GDB"
-Qt version "Qt 5.3.2"

Additional steps in order to make the application running:

The two files "CSV_Database_of_First_Names.txt" and "CSV_Database_of_Last_Names.txt" must be copied into the working directory.

Test machine

The application was verified on a notebook with a I7-3540M quad core processor with a 8GB RAM.
If the application is run on a machine with a different number of cores, it is possible to optimize the threads management by modifying the row 16 included in the file Mainwindow.cpp. This should be done by changing the value with the correct number of cores of the target machine.
The row 16 of the code where changes should be done is reported below:
#define NUMBER_OF_CORE 4 //number of core of the pc used

Unfortunately my test machine is not equipped with a Linux OS. Most of the development work was done on a Windows version of Qt and then moved to an Ubuntu 14 virtual machine.

Due to the memory size of my computer, I have executed tests with a maximum of 15 million of records. When selecting 10 million records the average size of memory used is 2GB.

Application description

The application developed uses a GUI to help the user during the creation of the inmemory DB and the search.
The workflow is presented below:

- (if needed) change the clusterization threshold parameter. Such parameter is used during the creation of the kd-tree and it represents the maximum number of elements that a single leaf node can have. The standard parameter is 1/10000 of the total number of records.
- Click on “Create Data” to generate the inmemory DB. This procedure does the following things: creates the DB, fills it with randomic data, clusterizes it using a kd-tree, sorts the DB by grouping the elements that are in the same cluster and finally it processes all the name strings (name+last name) by using the q-grams method and filling an inverted table. The complete process is quite longer (best result on windows machine is about 30 minutes).
- Fill the user fields (name, surname, latitude, longitude and age).
- Select the search that you want to execute. The button “search” means a linear or “brute force” search: this is done by calculating the distances between the user data and all the other elements of the DB. At the end a partial quick sort algorithm allows to select the first n (10) nearer elements. The button “Fast Search” uses the kd-tree going top - down to select the best approximate cluster where to find the 10 nearer elements; then it goes bottom - up searching all the tree using the results of the first selection as a threshold to reduce the number of calcs needed to find the exact solution. This threshold is further lowered of a constant (0.95) to speed up the process with a very little change in the results’ precision.

The application allows to do several researches, changing the user data and the parameter weight as you want. It is not possible to cancel and re-create the DB during the same run.

Application analisys and algorithms used

The study of the application started from an analysis of the key elements and of the restrictions:

1) total amount of elements (more than 10 million);
2) how to get the distances from two different strings;
3) time of execution of a single search (max 1 sec);
4) how to weight every parameter in order to have a single distance.

Point 1). I needed a 64bit architecture and toolchain to manage the total amount of memory and speed up all the calculus (32bit is not enough to handle more than 4 million of elements).
Point 2) can be split into two issues:

a) I needed a quick algorithm to calculate distances between two strings. Thinking to the tipology of the task that I need to execute (1 string compared versus N million) I selected the Q-grams algorithm, that allows to pre-process the strings that you have to compare with and create an inverted table that really speeds up the main search process.
b) The second issue is related to the kind of this parameter. with the Q-grams algorithm is possible to determinate the distance between a string and another, but the name parameter is not euclidean. So it's not possible to order N strings, and it's not possible, selecting a string as threshold, say if another string is bigger or smaller than that string. That's the why during the KD-tree creation I used only the age and position parameter to create the tree and grouping the DB in clusters. If the weight of the string will be much more than the other parameters, all the kd-tree creation will be less-useful.

To achieve point 3) I needed to move as much as computation as possible to the pre-processing phase, leaving only the strictly necessary calculus during the search execution. I have evaluated some algorithm of KNN search, then I have selected the KD-tree method -with some modifications. The standard kd-tree allows to organize all the points on a K dimensional space using a tree, where, after selecting the parameter with the greatest variance, every node has two branches: one with a lower value and another with a higher value. All the data are stored as a node (or leaf) in the tree. In my application I have too much elements to fill the tree completely: consequently, I have created a tree that allows to find the nearer index that represents an entire cluster of N elements. So every leaf of my tree is composed by an index and N elements that, if inserted in the tree, will reach that index. This kind of classification has another problem: one of the parameters is not euclidean. So the classification itself cannot research the real nearest element.

Finally the kd-tree is used only to quickly sort a certain number of clusters ordering from the cluster that has more probability to contains the N nearest element to the given point to the cluster that has the lowest probability. Calculating the N (10) elements that are nearer to the given point starting from the cluster that have more probability to contains the N nearest elements I can set a threashold quite near to the real distance value of the 10th nearest element. Using and updating this threshold I can avoid to calculate the complete distance for all the elements where 1 or more parameter's distance is higher than the threshold. The results showed that altought I used a single thread for the fast search and N parallel threads for the linear search I have a time saving of about 1/8 (comparing the calculus only) and more than 1/40 (comparing the calculus and the partial quick sort).

Point 4). I have selected the weight values (1 for name strings, 1.2 for age and 0.004 for geo distances) by giving a personal evaluation of the relevance of the the weight of each parameter, with respect to other parameters used in such a kind of research.

So, probably people that will use this application will expect to find in results other people of a similar age, with a similar name and who live not so far (in terms of kms) from them. I left the possibility to change directly these weight parameters, in order to see what happens if a different weight is given to one parameter or more.