Building an IT System: Staff Directory

Now that we have learnt a wide range of the principles involved in "building IT systems", in this exercise you and your colleagues will build a non-trivial IT system as a group. This will be done is a way similar to the corresponding lecture demonstration, but taking a few short-cuts to make the process quicker.

Requirement

Accompanying these instructions is a copy of a very large dataset containing details of employees in an organisation. The dataset has been provided in two different formats, to give you a choice of which to work with. Both formats contain exactly the same data. A plain text version has been provided as a comma-separated-values file, `employees.csv`. When opened in a text editor it appears as follows:

```
emp_no,birth_date,first_name,last_name,gender,hire_date
10001,1953-09-02,Georgi,Facello,M,1986-06-26
10002,1964-06-02,Bezalel,Simmel,F,1985-11-21
10003,1959-12-03,Parto,Bamford,M,1986-08-28
10004,1954-05-01,Chirstian,Koblick,M,1986-12-01
10005,1955-01-21,Kyoichi,Maliniak,M,1989-09-12
10006,1953-04-20,Anneke,Preusig,F,1989-06-02
10007,1957-05-23,Tzvetan,Zielinski,F,1989-02-10
10008,1958-02-19,Saniya,Kalloufi,M,1994-09-15
10009,1952-04-19,Sumant,Peac,F,1985-02-18
10010,1963-06-01,Duangkaew,Piveteau,F,1989-08-24
10011,1953-11-07,Mary,Sluis,F,1990-01-22
10012,1960-10-04,Patricio,Bridgland,M,1992-12-18
10013,1963-06-07,Eberhardt,Terkki,M,1985-10-20
10014,1956-02-12,Berni,Genin,M,1987-03-11
10015,1959-08-19,Guoxiang,Nooteboom,M,1987-07-02
10016,1961-05-02,Kazuhito,Cappelletti,M,1995-01-27
10017,1958-07-06,Cristinel,Bouloucos,F,1993-08-03
10018,1954-06-19,Kazuhide,Peha,F,1987-04-03
10019,1953-01-23,Lillian,Haddadi,M,1999-04-30
10020,1952-12-24,Mayuko,Warwick,M,1991-01-26
10021,1960-02-20,Ramzi,Erde,M,1988-02-10
10022,1952-07-08,Shahaf,Famili,M,1995-08-22
```

This version can also be inspected more easily in a spreadsheet program such as Microsoft Excel.

Also included is an SQLite database copy, `employees.db`. (A dump script, `employees_dump.sql`, has also been provided in case you have trouble opening the database.) When viewed in a database application such as the *DB Browser for SQLite* the database version appears as follows:

	emp_no	birth_date	first_name	last_name	gender	hire_date
	Filter	Filter	Filter	Filter	Filter	Filter
1	10001	1953-09-02	Georgi	Facello	M	1986-06-26
2	10002	1964-06-02	Bezalel	Simmel	F	1985-11-21
3	10003	1959-12-03	Parto	Bamford	M	1986-08-28
4	10004	1954-05-01	Chirstian	Koblick	M	1986-12-01
5	10005	1955-01-21	Kyoichi	Maliniak	M	1989-09-12
6	10006	1953-04-20	Anneke	Preusig	F	1989-06-02
7	10007	1957-05-23	Tzvetan	Zielinski	F	1989-02-10
8	10008	1958-02-19	Saniya	Kalloufi	M	1994-09-15
9	10009	1952-04-19	Sumant	Peac	F	1985-02-18
10	10010	1963-06-01	Duangkaew	Piveteau	F	1989-08-24
11	10011	1953-11-07	Mary	Sluis	F	1990-01-22
12	10012	1960-10-04	Patricio	Bridgland	M	1992-12-18
13	10013	1963-06-07	Eberhardt	Terkki	M	1985-10-20
14	10014	1956-02-12	Berni	Genin	M	1987-03-11

|< < 1 - 14 of 300024 > >| Go to: 1

Both versions contain the same data arranged in the same fields: employee number, date of birth, first name, last name, gender and hiring date. Most importantly, both versions are huge, with over 300,000 entries!

The challenge, therefore, is to develop an application which allows the company's human resources department to quickly find employees based on their name. Rather than forcing the user to type in the full name, the application should allow the user to enter a short prefix. The application should then respond by listing details of all employees whose first name or last name begin with this prefix.

(To avoid any privacy issues, the dataset is synthetic and has been generated automatically. As a result most prefixes that match any employee records tend to return a large number of similar results. It would appear that this company hires lots of families!)

For instance, the image below shows one possible version of the intended application and the results returned by a particular search. In this case the user has entered the prefix '`Tei`' and the application has returned all employees whose first name begins with this prefix, such as '`Teiji`', and whose last name begins with this prefix, such as '`Teitelbaum`'. (There are actually many more results than are shown below, which can be revealed by scrolling the text widget in the top right.) Only the following details are displayed for each employee: employee number, first name, last name and date of birth.

Although this exercise can be completed individually, it is best done as a group. Your task is to develop an application with a similar capability to the illustration above. To do so, you will work in two independent teams, one developing the GUI front-end and the other developing the back-end function that looks up the data. Crucially, you will need to agree on the interface between the front-end and the back-end so that you can put both halves together when they are finished.

Step 1 - Form Two Development Teams

In your group decide who will develop the front-end GUI and who will develop the back-end look-up function. Neither involves a large amount of code, but you should aim to partition the work according to your group's skills.

Step 2 - Define the Front-End/Back-End Interface

Before your two teams can begin work you must agree on the interface between the front-end and the back-end code. Clearly this will be a function which accepts the user's 'search term', i.e., the prefix of the employee's name, as a parameter. Less clear is what format the results will be returned in. Some questions your teams should discuss and agree on are:

- Should the results be returned as a list of matching employee details or as a single character string ready to display in the GUI?
- If a list, what should be the format of each item, a list of fields or a single character string?
    - If the former, should all fields for each matching employee be returned or just those that need to be displayed in the GUI? Notice above that our desired application does not use the employees' genders or hiring dates, so do these need to be returned at all?

  - If the latter, should the string contain comma-separated fields or ready-to-display text?
- What should be returned if no matches are found?

Once you have agreed on these points the two teams can begin developing their Python code independently.

Step 3a - Develop the Graphical User Interface

The first team should develop a GUI similar to that shown above, with a "stub" representing the incomplete back-end function. The minimal requirements for the GUI are:

- a text Entry widget to allow the user to enter employee name prefixes;
- a Button widget to allow the user to start the look-up process; and
- a Text or Label widget to display the results.

In our example above we have also included a "Staff Directory" image (actually a Label widget), but you should leave such decorative features to the end, if time permits.

Step 3b - Develop the Back-End Search Function

The second team has the task of developing the back-end function which, given the prefix of an employee's name returns details of all employees whose first or last names begin with this prefix. This can be done using either the SQLite database or the plain text copy of the dataset. At least three solutions are possible:

- Write a Python function which iterates over the lines of text in the CSV file to find matching employees using character string operations. In this case recall that it's easy to iterate over each line in a text file using a `for` loop in Python. Your solution will need to use the built-in file `open` function, the character string `split` method to extract the individual fields from each line (splitting the text on commas), and the string `startswith` predicate to determine if a first-name or last-name begins with the prefix entered by the user.
- Write a Python function which uses the regular expression `findall` function to find and return all employee records in the CSV file with a first-name or last-name that begins with the prefix entered by the user. In this case you will need the built-in `open` function and `read` method to get the contents of the CSV file as a single character string, and the `findall` method to find matching employee data. You will find this quite easy if you use `findall`'s `MULTILINE` option (as a third argument in the call to `findall`) which allows you to use `^` and `$` in your regular expression to match the beginning and end of each line of text in the overall string, respectively. (A variant of this approach would be to iterate over the lines of text in the file using a `for` loop and then use `findall` on each line to determine if it matches.)

- Write a Python function which accesses the SQLite database copy of the employee data. In this case you will need to use the usual SQLite methods for opening and closing the database, executing a query on its contents, and fetching the result set. The query in this case can use the "*C* `like '`*P*`%'`" Boolean expression to find rows in which a text-valued column *C* begins with text pattern *P*.

There is no particular advantage to any of the approaches. By way of illustration we have implemented all three back-ends in this week's workshop solution and all three required about the same amount of code.

If time permits you should also create some unit tests for your function which help document its intended behaviour and aid future maintenance.

Step 4 - Integration Testing and Fine Tuning

Finally, when both teams have completed their separate work they should get back together and integrate their results to produce the final "app". The teams may need to modify the code they've written to make them fit together properly. This is also the opportunity to make final improvements to the complete IT system before delivering it to the appreciative customer!