Project

Task 2: Due Dec 21, 2014 (Prepare to present in person on Sunday)

Implement all tasks without using a database and compare performance and describe reasons behind the discrepancies.

Task 1: Due Sunday Nov 23, 2014.

Teams of at most two each and Individual submissions are allowed.

Recent database research has looked into "Just-in-time" or "Invisible Loading" techniques to overcome the daunting cost of loading data before processing. Loading consists of:
1. Constructing the schema of a database
2. Copying the data into the database
3. Building the necessary indices to speed up future query workloads.

While invisible loading manages this cost through on-the-fly loading i.e. allowing users to write code directly on the raw files using special operators that move data from the raw file system into a backend database incrementally and gradually. JIT databases do not use a database at all and instead build on the fly position maps to at least speed up access into the raw file and minimize the overhead of parsing.

Your first project milestone is therefore to measure the cost of loading. You are provided with the following synthetic datasets that model real-life social networks (https://github.com/ldbc/ldbc_snb_datagen). The data is provided as comma-separated values in plain text. Your code should work with these three datasets:

Small: Social NT data for 1000 people ~ 40 MB (available on course drive)
Medium: Social NT data for 10,000 people ~ 1.1 GB (available on course drive)
Large: Social NT data for 100,000 people (You can build it using the generator … we will use this for testing)

1. The data set schema properties are explained here:
https://github.com/ldbc/ldbc_snb_datagen/wiki/Data-Schema
2. The csv output format is explained here:
https://github.com/ldbc/ldbc_snb_datagen/wiki/Data-Output

Part A

Create a script (in any language that you want) that

1. Sets up the database.
2. Creates the necessary schema tables.
3. Copies the data from each file into the database.

For each data set, and table log the time to copy the data. At the end of your script, create a table with at least the following details (this is just a sample table):

File	table	create-time (ms)	copy-time (ms)
Person.csv	person	100000ms	100000000ms
Comment.csv	comment	100000ms	100000000ms

Explain how you ensure that your timings are not tainted by external factors not related to the experiment? Think about caches and other processes running while you run your tests.

Part B

Consider a workload of the following two queries:

Query Type 1 (Shortest Distance Over Frequent Communication Paths)

Given two integer person ids *p1* and *p2*, and another integer *x*, find the minimum number of hops between *p1* and *p2* in the graph induced by persons who

1. have made more than *x* comments in reply to each others' comments (see comment_hasCreator_person and comment_replyOf_comment), and
2. know each other (see person_knows_person, which presents undirected friendships between persons; a friendship relationship between persons x and y is represented by pairs x|y and y|x).

API query1(*p1*, *p2*, *x*)
Output One integer (hop count) per line.

Sample input/output

Input	Output
query1(576, 400, -1)	3 % path 576-618-951-400 (other shortest paths may exist)
query1(58, 402, 0)	3 % path 58-935-808-402 (other shortest paths may exist)
query1(266, 106, -1)	3 % path 266-23-592-106 (other shortest paths may exist)
query1(313, 523, -1)	-1 % path none
query1(858, 587, 1)	4 % path 858-46-31-162-587 (other shortest paths may exist)
query1(155, 355, -1)	3 % path 155-21-0-355 (other shortest paths may exist)
query1(947, 771, -1)	2 % path 947-625-771 (other shortest paths may exist)

Query Type 2 (Interests with Large Communities)

Given an integer *k* and a birthday *d*, find the *k* interest tags with the largest range, where the range of an interest tag is defined as the size of the largest connected component in the graph induced by persons who

a. have that interest (see tag, person_hasInterest_tag),
b. were born on *d* or later, and
c. know each other (see person_knows_person, which presents undirected friendships between persons; a friendship relationship between persons x and y is represented by pairs x|y and y|x).

API query2(*k*, *d*)

Output Exactly *k* strings (separated by a space) per line. These *k* strings represent interest tag names, ordered by range from largest to smallest, with ties broken by lexicographical ordering.

Sample Input/Output

Input	**Output**
`query2(3, 1980-02-01)`	`Chiang_Kai-shek Augustine_of_Hippo Napoleon % component sizes 22 16 16`
`query2(4, 1981-03-10)`	`Chiang_Kai-shek Napoleon Mohandas_Karamchand_Gandhi Sukarno % component sizes 17 13 11 11`
`query2(3, 1982-03-29)`	`Chiang_Kai-shek Mohandas_Karamchand_Gandhi Napoleon % component sizes 13 11 10`
`query2(3, 1983-05-09)`	`Chiang_Kai-shek Mohandas_Karamchand_Gandhi Augustine_of_Hippo % component sizes 12 10 8`
`query2(5, 1984-07-02)`	`Chiang_Kai-shek Aristotle Mohandas_Karamchand_Gandhi Augustine_of_Hippo Fidel_Castro % component sizes 10 7 6 5 5`
`query2(3, 1985-05-31)`	`Chiang_Kai-shek Mohandas_Karamchand_Gandhi Joseph_Stalin % component sizes 6 6 5`
`query2(3, 1986-06-14)`	`Chiang_Kai-shek Mohandas_Karamchand_Gandhi Joseph_Stalin % component sizes 6 6 5`

Write the SQL queries/scripts for the above queries. Make sure your scripts can take input from a text file that looks as follows:

```
query1(576, 400, -1)
query2(3, 1983-05-09)
query1(266, 106, -1)
```

and produce output that looks as follows:

```
3
Chiang_Kai-shek Mohandas_Karamchand_Gandhi Napoleon
3
```

You can use any database trick you have learned, temporary tables, indices.

Compete:
Who produces the fastest query? You will be allowed to run your scripts on the same machine once done to demonstrate the better DB person.

Now revise your loading script

Include the time required to create temporary tables and any indices you need for these queries. Explain how you eliminated any external factors from messing up the values you reported. Think about caches and other processes running while you run your tests.

Produce a table that looks like this:

table(s)	Index/view	create-time (ms)
person	name	1000000000ms
person	Person_old	10000000000ms