CptS 415 Big Data
Assignment 1
Jinyang Ruan
011696096

1. Big Data Application: Facebook
- Volume: the amount of data
  Based on searching on Google, Facebook has horrendously large data since it generates 4 petabytes of data per day.
- Velocity: the speed of data entering a solution
  When we send message or post anything on Facebook, it is real-time.
- Variety: different data sources and different types
  Data on Facebook has multiple data types which include numbers, texts, videos, tables, etc. It also includes all kind of data source types such as structured, semi-structured and unstructured.
- Veracity: the degree to which the data is accurate, precise, and trusted
  Since data from Facebook is usually from real-world and data in real-life is often dirty, which means most data on Facebook is inaccurate and misleading.
- Value: the ability of a solution to extract meaningful information from the data.
  I can search the get information from Facebook in several seconds.

If I am required to design a data base system for this application, I would like to use relational databases. Basically, the data on Facebook comes from different users. We live in a world village now and all of us are related in some cases, so the data from different users is also related.

2.
a.

Terms	Explanation
Relation Schema	The description of a relation. In this case, relation schema should be Airport(Airport ID, Name, City, Country, …, Source)
Attribute	Entries in the Airport table. In the Airport table, there are 14 attributes such as Airport ID, Name, etc.
Attribute Domain	Each Attribute has domain which defines its logical definition and datatype or format. For instance, attribute IATA is defined as 3-letter IATA code.
Relation Instance	A relation instance is a tuple in a relation. In this case, a particular combination of airport's attribute value is one relation instance.

Airport Table instance:
Source: https://raw.githubusercontent.com/jpatokal/openflights/master/data/airports.dat

Airport ID	Name	City	Country	IATA	ICAO	Latitude	Longitude	Altitude	Timezone	DST	Tz	Type	Source
1	Goroka Airport	Goroka	Papua New Guinea	GKA	AYGA	-6.081689834590001	145.391998291	5282	10	U	Pacific/Port_Moresby	airport	OurAirports
2	Madang Airport	Madang	Papua New Guinea	MAG	AYMD	-5.20707988739	145.789001465	20	10	U	Pacific/Port_Moresby	airport	OurAirports

3	Mount Hagen Kagamuga Airport	Mount Hagen	Papua New Guinea	HGU	AYMH	-5.826789855957031	144.29600524902344	5388	10	U	Pacific/Port_Moresby	airport	OurAirports
4	Nadzab Airport	Nadzab	Papua New Guinea	LAE	AYNZ	-6.569803	146.725977	239	10	U	Pacific/Port_Moresby	airport	OurAirports

b.
Airport(Airport ID, Name, City, Country, IATA, ICAO, Latitude, Longitude, Altitude, Timezone, DST, Tz database time zone, Type, Source)
Airline(Airline ID, Name, Alias, IATA, ICAO, Callsign, Country, Active)
Route(Airline, Airline ID, Source airport, Source airport ID, Destination airport, Destination airport ID, Codeshare, Stops, Equipment)

Airport:

Airport ID	Name	City	Country	IATA	ICAO	Latitude	Longitude	Altitude	Timezone	DST	Tz	Type	Source

Airline:

Airline ID	Name	Alias	IATA	ICAO	Callsign	Country	Active

Route:

Airline	Airline ID	Source airport	Source airport ID	Destination Airport	Destination Airport ID	Codeshare	Stops	Equipment

Primary key for each schema has been underlined.
Foreign key constraints are displayed as directed arc (arrow) from the FK to the referenced table.
Functional dependencies:
Airport:

Airport ID	Name	City	Country	IATA	ICAO	Latitude	Longitude	Altitude	Timezone	DST	Tz	Type	Source

Airport ID → Name
Airport ID → City
…
Airport ID → Source
Airport ID can determine values of other 13 attributes, hence 13 valid functional dependencies.

c.
If Airport ID → City
Then Airport ID + Name → Name + City

If Airport ID → Country
Then Airport ID + IATA → Country + IATA

d.

R(A_1, A_2, A_3, A_4)

FDs:

A_1	A_2	A_3	A_4

NF3 removes transitive dependencies.

Since A3 is the primary key, and there is one transitive dependence form A2 to A4, we can convert it as following.

$\underline{A_3}$	A_1	A_4

$\underline{A_2}$	A_4

3.

Q1: Which theaters feature "Zootopia"?

$$\pi_{Theaeter}(\sigma_{Title="Zootopia"}(Schedule))$$

Q2: List the names and address of theaters featuring a film directed by Steven Spielberg.

$$\pi_{Theaeter,Adress}(\sigma_{Director="Steven\ Spielberg"}(\text{Movies} \bowtie_{Title} Schedule \bowtie_{Theater} Location))$$

Q3: What are the address and phone number of the Le Champo theater?

$$\pi_{Address,Phone\ number}(\sigma_{Theater="Le\ Champo"}(Location))$$

Q4: List pairs of actors that acted in the same movie.

$$\rho(C(3 \rightarrow actor'), Movie')$$

$$\pi_{Actor,Actor'}(\sigma_{Actor!=Actor'}(Movies \bowtie_{Title} Movie'))$$

4.

a. Block Nested Loops Join.

For each block B_R of R
    For each block B_S of S
        For each tuple t_R in B_R
            For each tuple t_S in B_S
                Test if pair (t_R, t_S) satisfy the join condition θ:
                If $R.A = S.B$, then
                        add $t_R * t_S$ to the result.
            end for
        end for
    end for
end for

Total tuples in R = 100,000
B_R =10,000
Total tuples in S = 20,000
B_S =2,000

Worst case block transfer = $B_R + B_R * B_S = 10{,}000 + 10{,}000 * 2{,}000 = 20{,}010{,}000$
Worst case seeks =$2B_R = 20{,}000$

Best case transfer = $B_R + B_S = 10{,}000 + 2{,}000 = 12{,}000$
Best case seeks = 2

Not best not worst: memory can hold 52 blocks
Transfer = $B_R + B_R * (B_S - 51) + 51 = 10{,}000 + 10{,}000 * 1{,}949 + 51 = 19{,}500{,}051$
Seeks =$2B_R = 20{,}000$

b. Sort-merge Join

Memory size $M = 52$
Create sorted runs:

- Let i be 0 initially.
- Repeatedly do the following till the end of the relation:
    - Read M blocks of relation into memory
    - Sort all in-memory tuples
    - Write sorted data to temporary run file R_i; increment i.
- Let the final value of i be N
- $N = (B_R/M)$

Merge the N runs.

- Use N blocks of memory to buffer input runs, and 1 block to buffer output. Read the first block of each of the M runs into its buffer page (N block in total)
- Repeat
    - Select the first record (in sort order) in each block (N in total) among all buffer pages.
    - Write the record to the output buffer. If the output buffer is full write it to disk.

- Delete the record from its input buffer page. If the buffer page becomes empty then read the next block (if any) of the run into the buffer.

- Until all input buffer pages are empty.

Block transfers:
$B_R\left(2log_{M-1}\left(\frac{B_R}{M}\right)+2\right) \approx 10{,}000 * (2 * log_{51}192 + 2) \approx 10{,}000 * (2 * 1.34 + 2) = 46{,}800$
Seeks:
$2* \left\lceil\frac{B_R}{M}\right\rceil + \left\lceil\frac{B_S}{B_b}\right\rceil \left(2log_{M-1}\left(\frac{B_R}{M}\right) - 1\right) \approx 386 + 39 * 1.68 \approx 452$

c. Hash Join

Partition the relation *R* and *S* using hashing function *h*.
h maps *JoinAttrs* values to {0, 1, …, n} partitions where *JoinAttrs* denotes the common attributes of R and S

- $r_0, r_1, r_2, \dots, r_n$ denote partitions of *R* tuples
  - Each tuple $T_R \in R$ is put in partition r_i where $i = h(t_R[JoinAttrs])$
- $s, s_1, s_2, \dots, s_n$ denote partitions of *S* tuples
  - Each tuple $T_S \in S$ is put in partition s_i where $i = h(t_S[JoinAttrs])$

Tuples in r_i need only to be compared with tuples in s_i

We assume that $s_i < mem\ buffer$, cost of hash join is:
Transfers = $3(B_R + B_S) = 3 * (10{,}000 + 2{,}000) = 36{,}000$
Seeks = $2\left(\left\lceil\frac{B_R}{B_b}\right\rceil + \left\lceil\frac{B_S}{B_b}\right\rceil\right) = 2 * \left(\frac{10{,}000}{50} + \frac{2{,}000}{50}\right) = 480$