Carnegie Mellon University
15-826 – Multimedia Databases and Data Mining
Fall 2013, C. Faloutsos
Homework 1 Solutions
Prepared by: Seunghak Lee

Q1 – DBMS and SQL [30pts]

Problem Description: For each question in this part, provide both the SQL statement(s) and the resulting answer(s), unless specified otherwise. You'll be using the *movie reviews from Amazon* dataset [1] [2] [3] [4]
Hint: Please use SQLite3; version 3.6.20 is already on the andrew cluster machines. You may use your own machine and your own sqlite3 installation, as long as your submitted code runs correctly on the andrew cluster machines.

Implementation Details: Write sql code for the following queries.

1. **[0 pt]** *Line number:* Download the movies-short.txt.gz[2] from
   `http://www.cs.cmu.edu/~seunghak/movies-short.txt.gz`
   and `make` file[3] from
   `http://www.cs.cmu.edu/~christos/courses/826.F13/HOMEWORKS/HW1/Q1/makefile-q1.tar.gz`
   and unzip it. Then count the number of lines of movies-mini.csv (try '`make` linenum'). Next, try '`make`', and it will show you what to implement.
2. **[2 pt]** *Parsing:* Extract the following five columns from `movies-short.txt`: productId, userId, helpfulness, score, time. (Run `wc -l movies-short.txt`, to check if the number of lines of the movie data is 47470103.)
   (**Solution**) Write a script program that coverts the original file 'movies-short.txt' to 'movies-short.csv' with the format of 'productId, userId, helpfulness, score, time' for each review. For example, each line represents a review 'B003AI2VGA, A141HP4LYPWMSR, 7/7, 3.0, 1182729600'.
3. **[2 pt]** *Loading:* Create and load the following table, using the parsed data above.
   - `movies(productId, userId, helpfulness,score, time)`

[1] movie data: `https://snap.stanford.edu/data/movies.txt.gz`.
[2] To save bandwidth, short version (`movies-short.txt.gz`) without text comments is available at `http://www.cs.cmu.edu/~seunghak/movies-short.txt.gz`
[3] information about the movie data: `https://snap.stanford.edu/data/web-Movies.html`
[4] make file: `http://www.cs.cmu.edu/~christos/courses/826.F13/HOMEWORKS/HW1/Q1/makefile-q1.tar.gz`

(Hint: check `.help` and `.import` in SQLite3)
(**Solution**) In the previous problem, we generated 'movies-short.cvs' using 'movies-short.txt'. We run the following in sqlite3:

```
sqlite> create table movies (productId TEXT, userId TEXT, helpfulness integer,score
real, time integer);
sqlite> .separator ","
sqlite> .import movies-short.cvs movies
```

4. [**2 pt**] *Movie size:* How many distinct movies does this dataset contain?
(**Solution**) `select count(DISTINCT productId) from movies;`
The number of distinct movies: 253059
5. [**4 pt**] *Popular movie:* What is the most popular movie in this dataset (the movie with the largest number of reviews)? Report both productId and its real movie name by searching 'www.amazon.com/dp/[productId]' in the web.
(**Solution**)

```
select productId from movies group by productId order by count(productId)
DESC LIMIT 1;
```

The most popular movie is:
ProductID: B002QZ1RS6
Movie name: INSANITY DVD Workout
6. [**4 pt**] *Active reviewer:* Who is the most active reviewer? (the reviewer with the largest number of reviews)
(**Solution**)

```
select userId from movies group by userId order by count(userId) DESC LIMIT
1;
```

The most active reviewer is: A16CZRQL23NOIW
7. [**4 pt**] *Top movies:* What are the top 5 movies with the highest average score among the movies with more than 100 reviews? Report productId. For fun, manually find its real movie name by searching 'www.amazon.com/dp/[productId]' in the web.
(**Solution**)

```
select productId from movies group by productId having count(productId)>
100 order by avg(score) DESC LIMIT 5;
```

The top 5 movies:
B007MDB6RO (GCB: The Complete First Season (2012))
0578046725 (Food Production Systems for a Backyard or Small Farm)
B00006B1HI (9/11 - The Filmmakers' Commemorative Edition (2002))
B00006B1HK (9/11 - The Filmmakers' Commemorative Edition [VHS] (2002))
B00006LSE8 (9/11 (2002))
[**2 pt**] *Report* the wall-clock running time of your query, using, e.g., the `time` Linux/Unix command.
(**Solution**)

`time sqlite3 movies-short.db <hw1.q1.7.txt`
real 0m32.529s

8. **[2 pt]** *Index impact:* Create indices on the columns `productId`, and `score` (follow the syntax shown here).
(**Solution**)
`CREATE INDEX productId_score_index ON movies (productId,score);`
[2 pt] Run the query from the last question again ("Top movies"), and *report* the wall-clock running time.
(**Solution**)
real 0m4.622s
9. **[2 pt]** *Query optimization:* Show the output of `explain select` for the previous query in the last two questions, i.e., with, and without indices.
(**Solution**)
With index:
0|0|TABLE movies WITH INDEX productId_score_index ORDER BY
Without index:
0|0|TABLE movies
[4 pt] Justify the speed-up with the indices.
Hint: See `http://www.sqlite.org/draft/eqp.html` for a (rough) description of the output of `explain`.
(**Solution**)
With index, sqlite searched the list of productId sorted by productId and score, and thus it could find the relevant entries fast. Without index, sqlite carried out sequential scan which slowed down the query.

What to turn in:

- **Code:** Submit a text file to blackboard in a file name `hw1.q1.tar.gz` (without the data such as movies.db or movies-short.txt), with all the SQL queries and commands you need to answer the queries above.. Make sure it runs: we will grade it using
    `make all`
(See reminder on the front page regarding how to organize your code files for submission to blackboard.)
- **Answers:** Submit a text file to blackboard with your results and responses. Call it `hw1.q1.output.txt`; create it with
    `make all` > hw1.q1.output.txt.
and append query results for "popular movies", "Top movies" , the wall-clock times, and your justification to Q9 to this text file. (Submit answers for Q1 on separate page; hardcopy for each question should be submitted separately.)

Q2 – KD-Trees [40pts]

Problem Description: Consider the k-d-tree package, in C, at KD-Tree Package [5] (tar xvf; make). We want to augment the nearest neighbor search, so that it returns the k=10 nearest neighbors, instead of the 1 nearest neighbor that it returns now. If the tree has fewer than k=10 nodes, then return all the nodes, with a warning: "only <number> nodes found".

1. **[20 pt]** write kdtree with k=10 nearest neighbors (k should be a parameter in your code).

   (**Solution**) We create a priority queue `pq` to keep k-nearest neighbors, and it should have the following functions:
   `pq.largestDist` that retrieves the largest distance within the priority queue, and
   `pq.insert` that inserts a node into the priority queue.

   Pseudocode:

```
pq = knnsearch(root, query, pq){
    if (distance(root,query) < pq.largestDist){
        pq.insert(root)
    }
    h = hypersphere around query with radius pq.largestDist
    if (h intersects the hyperplane of the left-subtree)
        pq = knnsearch(left-subtree-root, query, pq)
    }
    if (h intersects the hyperplane of the right-subtree)
        pq = knnsearch(right-subtree-root, query, pq)
    }
    return pq
}
```

2. **[20 pt]** a hard copy of the output of your code, on the two included input scripts (2d-input.txt,3d-input.txt; they are included in 'kdtree.tar.gz'). In your output, keep only the lines that are the result of the query, to save paper. For your convenience, try 'make hw1'

   (**Solution**) 10-NN of the three points in 2d-input.txt.

   0.500539 0.50159

   0.49842 0.501218

[5] `http://www.cs.cmu.edu/~christos/courses/826.F13/HOMEWORKS/HW1/Q2/kdtree.tar.gz`

0.502002 0.501444

0.50221 0.501296

0.502499 0.499022

0.503631 0.501533

0.496087 0.498

0.497435 0.496108

0.498964 0.495344

0.502271 0.5043

*

0.999869 1.000052

0.999022 0.999642

0.999022 0.999642

0.997992 0.998421

0.997639 0.998586

0.99639 0.99712

0.995784 0.997902

0.996669 0.995438

0.994865 0.997176

0.994852 0.996276

*

0.002425 0.000256

-0.001678 0.002483

0.001384 0.003112

0.002089 0.003123

0.005634 0.004583

0.006286 0.00506

0.0022 0.008047

0.005834 0.007986

0.008229 0.005723

0.008105 0.007026

10-NN of the two points in 3d-input.txt.

0.501403 0.501639 0.500283

0.500965 0.499862 0.497529

0.499576 0.498828 0.497649

0.501154 0.499437 0.502467

0.497303 0.498551 0.497515

0.502697 0.502087 0.502288

0.50224 0.503525 0.499414

0.496252 0.497806 0.498141

0.496741 0.500506 0.496479

0.503554 0.504731 0.503079

*

0.799253 0.800578 0.800628

0.799543 0.801168 0.799171

0.798736 0.798995 0.799473

0.798242 0.797204 0.799336

0.802433 0.801048 0.802293

0.800675 0.803042 0.801632

0.7999 0.796638 0.797813

0.796724 0.797453 0.801043

0.799279 0.796064 0.797378

0.80257 0.802304 0.803562

What to turn in:

- **Code:** Submit your code to blackboard in a file `hw1.q2.tar.gz`. (Your `make` should print your responses to the two input files, i.e., 2d-input.txt and 3d-input.txt.) (See reminder on the front page regarding how to organize your code files for submission to blackboard.)
- **Answers:** Submit a hard copy for Q2.2. (Submit answers for Q2 on separate page; hardcopy for each question should be submitted separately.)

Q3 – Hilbert and z-order [30pts]

Problem Description: Write the code to compute the z-value and the hilbert-value of a 2-d point, as well as the inverses. The command-line syntax should be, e.g. for the 'zorder' version: zorder -n <order-of-curve> <xvalue> <yvalue>
Thus:

- zorder -n 2 0 0 # should return '0'
- zorder -n 3 0 1 # should return '1'
- horder -n 2 0 0 # should return '0'

Write four programs to do these computations; hand in your source code on hard copy; and submit your source code to blackboard.

1. **[5 pt]** zorder should return the z-value of the given (x,y) point.

   (**Solution**)

   Pseudocode:

```
z_value = zorder(x, y, n){
    x_bits = make_binary(x)
    y_bits = make_binary(y)
    z_bits = ()
    if (length(x_bits) > n or length(y_bits) > n)
        exit("input out of range")
    for bit_counter in 1:n{
        z_bits = cat(z_bits, x_bits[bit_counter])
        z_bits = cat(z_bits, y_bits[bit_counter])
    }
    return(make_decimal(z_bits))
}
```

Explanation:

Make sure to check that input is in range of $-n$ parameter. That is, x and y must be $< 2^n$ in order to fit in the $2^n \times 2^n$ grid. If so, make a new binary number by concatenating the first (leftmost) bit of the xcoordinate, then the first bit of the y-coordinate, then the second bit of the x-coordinate, and so on. Translate this number to base 10 and output.

2. **[5 pt]** izorder should give the inverse:

   - izorder -n 5 0 # should return the 'x' and 'y' values, ie, 0 0
   - izorder -n 2 15 # should return '3 3'

   (**Solution**)

   Pseudocode:

```
(x_value, y_value) = izorder(z, n){
    x_bits = ()
    y_bits = ()
    z_bits = make_binary(z)
    z_bit_counter = 1
    for xy_bit_counter in 1:n{
        x_bits = cat(x_bits, z_bits[z_bit_counter])
        z_bit_counter++
        y_bits = cat(y_bits, z_bits[z_bit_counter])
        z_bit_counter++
    }
    return(make_decimal(x_bits), make_decimal(y_bits))
}
```
(1)

Explanation:

Reverse the process above. Translate the input to binary, then put the first bit as the first bit of the x-coordinate, the second bit as the first bit of the y-coordinate, the third bit as the second bit of the x-coordinate, and so on. Translate the final x and y-coordinates to decimal and output.

3. **[5 pt]** horder should compute the hilbert order of the given point - specifically

   - horder -n 2 0 0 # should return '0'
   - horder -n 1 0 1 # should return '1'
   - horder -n 2 0 1 # should return '3'

   (**Solution**) Please refer to `http://en.wikipedia.org/wiki/Hilbert_curve` and Appendix in "Linear Clustering of Objects with Multiple Attributes" Jagadish [SIGMOD 90].

4. **[5 pt]** ihorder should do the inverse - for example

   - ihorder -n 2 0 # should return '0 0'
   - ihorder -n 1 1 # should return '0 1'
   - ihorder -n 2 3 # should return '0 1'

   (**Solution**) Please refer to `http://en.wikipedia.org/wiki/Hilbert_curve` and Appendix in "Linear Clustering of Objects with Multiple Attributes" Jagadish [SIGMOD 90].

5. **[0 pt]** Download a file from
[`http://www.cs.cmu.edu/~christos/courses/826.F13/HOMEWORKS/HW1/Q3/format-q3.tar.gz`], in which there are two files: hw1-q3.output.txt, and hw1-q3.input.txt.

   Make sure your results match 'hw1-q3.output.txt' when applied on 'hw1-q3.input.txt'. (Hint: check `diff`) If there is difference, change your code so that it gives outputs that match 'hw1.q3.output.txt', as we will use this format for grading.

6. **[5 pt]** Give the results of your programs on the input file:

   [`http://www.cs.cmu.edu/~christos/courses/826.F13/HOMEWORKS/HW1/Q3/inp.txt`]

   Make sure you echo the input, so that it is clear which answer refers to which question

   (**Solution**)

   2

   3

   14

   0 0

   2 2

   0

   2

   2

   0 0

   1 0

   0 2

   207

   216

   10279

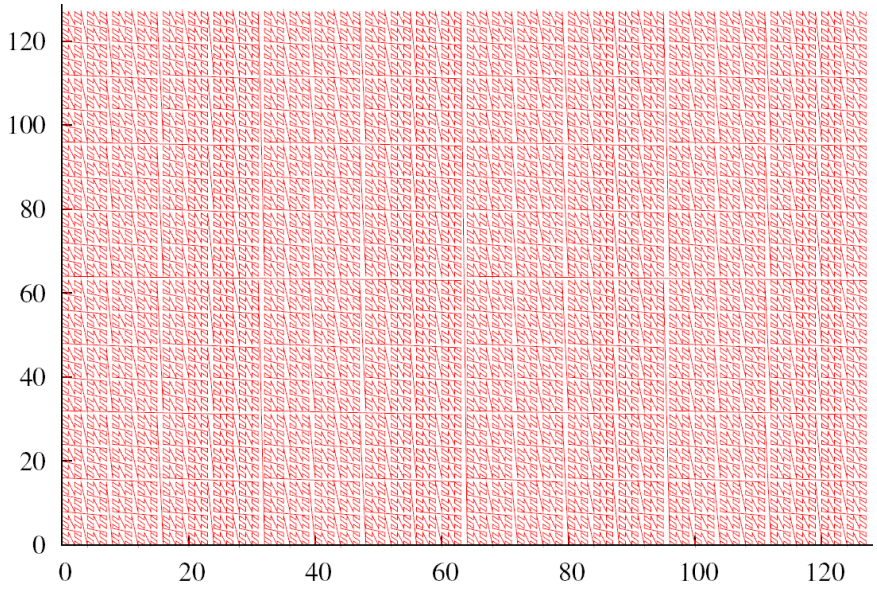
   10285

   57 33

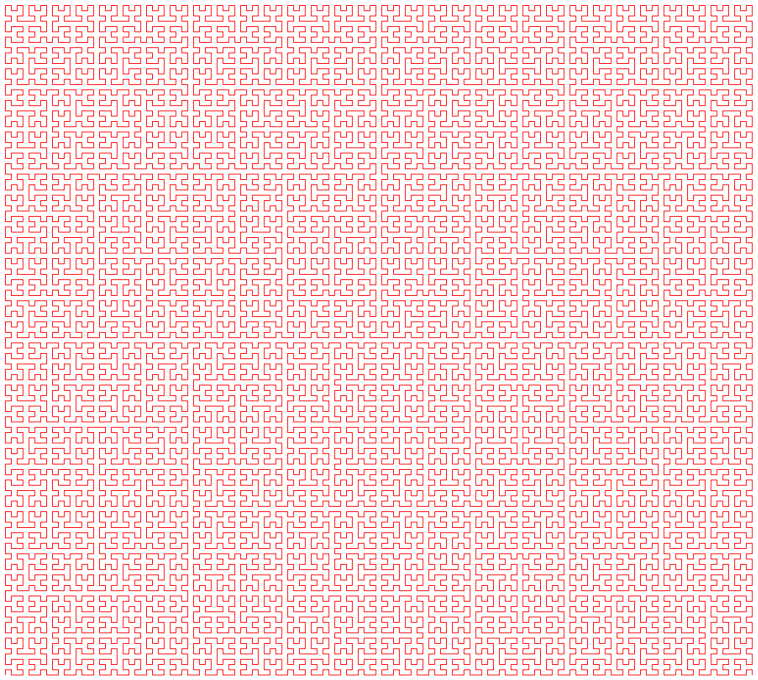
   24 36

7. **[2 pt]** Using your izorder, plot a z-curve of order 7 (128×128 grid) and hand in the hardcopy of the plot.

8. **[3 pt]** Using your ihorder, plot a Hilbert curve of order 7, and hand in the hardcopy of the plot.



(a)



(b)

Figure 1: (a): A z-curve of order 7, and (b): A Hilbert curve of order 7.