**CSI 5450 - Database Systems I**

**Winter 2019, Midterm Exam**

**Xuke Yan**

**03/12/2019**

Note:

1. database, tables, python code are posted in my Git repos:
2. Toolbox in this project is SQLite. Programming language is Python3.6
3. Functional Dependencies (8 points)

**Goals:**

Analyze a data set and verify functional dependencies.

**Description:**

During the lecture, you were given the definition of functional dependencies. You were shown two methods to derive them, first from analyzing a snapshot of the data, second, using a domain expert. In this section you will analyze the supplied dataset and verify functional dependencies for the data set. Consider the supplied data set in the file data.txt. Which of the following functional dependencies may hold in the relation? If the dependency does not hold, explain why by specifying some tuples that cause a violation.

**Solution:**

Change data.txt into data.csv.

Import into SQLite as db named midterm and table name data.

Add header A, B, C, D, E, F, G, H in data table.

Find the following functional dependencies analyzed as:

|  |  |  |
| --- | --- | --- |
| a) C → H | No | 0 in C may refer low, med, high in H |
| b) H → G | No | Med in H may refer low, med, high in G |
| c) F → G | No | Big in F may refer high, v-high in G |
| d) G → F | Yes |  |
| e) F → B | Yes |  |
| f) B → F | Yes |  |
| g) A → C | No | Low in A may refer 0, 1, … in C |
| h) B → D | No | Small in B may refer a, b … in D |

1. Normalization (12 points)

**Goals:**

• use the functional dependencies to normalize the data set

• document the normalized data set using formal model

• document the normalized data set using SQL DDL

**Description:**

During the lecture, several normal forms were defined. The functional dependencies for the data set are defined in the file names.txt:

1. Design a schema based on these functional dependencies.

2. Normalize the schema into third normal form (3NF).

3. Document your schema using both a formal schema and SQL DDL.

**Solutions:**

1. From previous database, we know we have table of {A,B,C,D,E,F,G,H}. Each of the attribute is atomic. Applying the FDs giving from name.txt, we can create a 1NF schema as below:

R={C,D,G,A,B,E,F,H}

1. To normalize the schema into 3NF, we firstly need to make it 2NF. First, identify partial dependencies that violate 2NF. These are attributes that are functionally dependent on either parts of the key, {C} or {D}, alone. We can calculate the closures {D}+ to determine partially dependent attributes:

{D}+={D,F,G,H}.So we get 2NF in below:

R1={C,D,B,E};

R2={D,F,G,H};

R3={C,A};

Then We can make it into 3NF to remove the transitive dependencies in R1, R2, R3. The transitive dependency {D}🡪{G}🡪{H}. So we should remove {H} from R2 into R21, and copy the attribute G into R21. The remaining attributes are kept in a relation R22

R21={G,H};

R22={D,F,G};

The final set of relations in 3NF are {R1, R21, R22, R3}.

(Note that R22= {D,F,G}, FD doesn’t hold, but we are informed to ignore it.)

1. create table R1(C int, D text, B text, E text);

create table R22(D text, F text, G text);

create table R21(G text, H text);

create table R3(C int, A text);

(In next section, I will update table and add rowID for each and give you a reason)

1. Indexes (10 points)

**Goals:**

Provide evidence to prove or disprove the following statement:

“Indexes are a more efficient access path to the data set than direct access.”

**Description:**

In the lecture several index structures were presented. The justification for using the index is to enhance query performance. The toolbox you have chosen for this part of the exam should support several index structures.

Consider the following query:

SELECT DISTINCT B, H

FROM {insert your tables here}

WHERE G=low AND (A = med OR A = high)

Import the dataset into your normalized database, then run the above query and time the performance on the following index combinations:

1. no indexes

2. index on A, no index on G

3. index on G, no index on A

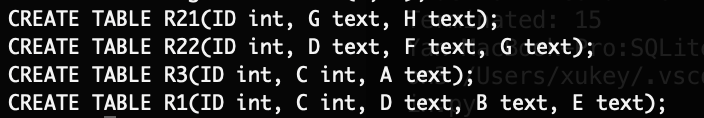
4. separate indexes on A and G

5. composite index on both A and G

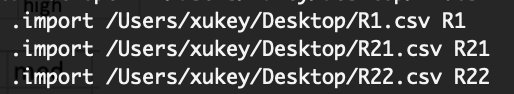
**Solution:**

We need to first made some necessary modification on the data.txt before inserting the dataset into our database. We need to add an incremented ID in the dataset.

Knowing from question#2, we have four normalized schemas documented named as R1, R21, R22, R3. I apply this incremented ID for each table as primary key column. That will also make it happen when we want to join multiple tables using the ID as foreign key.

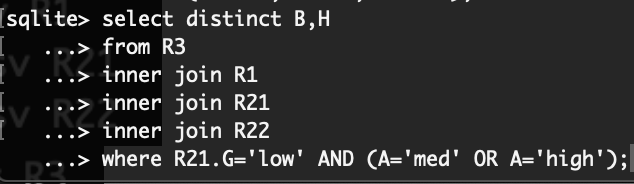


1. Import R1, R21, R22, R3 dataset into the table R1, R21, R22, R3:





1. Run supplied sql command:



The Result is big, med. That confirms that our imported dataset works fine.

1. Write python script to time the different index conditions.
2. Import sqlite3.py and sqlite3.connect the database
3. Define five test cases by add index A, G, A and G separately and combined.
4. (Ignore the execution time to Add index/ Drop index)
5. Use timeit.py to measure the execution time of different cases
6. Source code displays as below:
7. Result:

|  |  |
| --- | --- |
| test cases | time |
| No index condition: | 0.018 |
| Index on A condition: | 0.009 |
| Index on G condition: | 0.007 |
| Separate index on A and G condition: | 0.005 |
| Composite index on A and G condition: | 0.004 |

import sqlite3

import timeit

def indexNo():

conn = sqlite3.connect('midterm')

curs = conn.cursor()

curs.execute('select distinct B,H \

from R1 inner join R21 inner join R22 inner join R3\

where R21.G= "low" AND (A="med" OR A="high")')

conn.commit()

def indexA():

conn = sqlite3.connect('midterm')

curs = conn.cursor()

curs.execute('create index tag\_A on data (A)')

curs.execute('select distinct B,H\

from R1 inner join R21 inner join R22 inner join R3\

where R21.G= "low" AND (A="med" OR A="high")')

conn.execute('drop index tag\_A')

conn.commit()

def indexG():

conn = sqlite3.connect('midterm')

curs = conn.cursor()

curs.execute('create index tag\_G on data (G)')

conn.commit()

curs.execute('select distinct B,H\

from R1 inner join R21 inner join R22 inner join R3\

where R21.G= "low" AND (A="med" OR A="high")')

conn.execute('drop index tag\_G')

conn.commit()

def indexAandG():

conn = sqlite3.connect('midterm')

curs = conn.cursor()

curs.execute('create index tag\_G on data (G)')

conn.commit()

curs.execute('create index tag\_A on data (A)')

conn.commit()

curs.execute('select distinct B,H\

from R1 inner join R21 inner join R22 inner join R3\

where R21.G= "low" AND (A="med" OR A="high")')

conn.commit()

conn.execute('drop index tag\_A')

conn.execute('drop index tag\_G')

conn.commit()

def indexAG():

conn = sqlite3.connect('midterm')

curs = conn.cursor()

curs.execute('create index tag\_AG on data (A, G)')

conn.commit()

curs.execute('select distinct B,H\

from R1 inner join R21 inner join R22 inner join R3\

where R21.G= "low" AND (A="med" OR A="high")')

conn.commit()

conn.execute('drop index tag\_AG')

conn.commit()

if \_\_name\_\_ == '\_\_main\_\_':

print('No index condition:')

print(timeit.timeit(stmt='indexNo()', number =1, setup='from \_\_main\_\_ import indexNo'))

print('Index on A condition:')

print(timeit.timeit(stmt='indexA()', number =1, setup='from \_\_main\_\_ import indexA'))

print('Index on G condition:')

print(timeit.timeit(stmt='indexG()', number =1, setup='from \_\_main\_\_ import indexG'))

print('Seperate index on A and G condition:')

print(timeit.timeit(stmt='indexAandG()', number =1, setup='from \_\_main\_\_ import indexAandG'))

print('Composite index on both A and G condition:')

print(timeit.timeit(stmt='indexAG()', number =1, setup='from \_\_main\_\_ import indexAG'))