

Similarity Joins

Some of you may remember Homework 5 from a short while ago. This question is about ways we could improve the similarity join algorithm you implemented there.

- Recall from Homework 4 that we ensure that the list of trigrams for a given input string has at maximum 1 instance of each trigram. For example, the trigrams for “geegee” are [“gee”, “eeg”, “ege”] (excluding padding). We decide that we want to weight trigrams based on how frequently they appear in the original string, so our output for the above string would be, conceptually, [“gee”: 2, “eeg”: 1, “ege”: 1].

a) **[1 pt]** Here is the pseudocode for generate_trigrams:

```

1.  DEF GENERATE_TRIGRAMS(INPUT):
2.      TRIGRAMS = LIST()
3.      FOR WORD IN INPUT.SPLIT(" "): # SPLIT INPUT INTO WORDS
4.          # ADD ALL TRIGRAMS FOR THE GIVEN WORD TO OUR LIST.
5.          TRIGRAMS += MAKE_TRIGRAMS(WORD)
6.      SORT(TRIGRAMS) # SORT THE LIST OF TRIGRAMS
7.      UNIQUE(TRIGRAMS) # REMOVE DUPLICATES
8.      RETURN TRIGRAMS

```

Which line should we replace with a call to “AGGREGATE(TRIGRAMS)”, which aggregates adjacent trigrams into the desired weighted list?

- [1 pt]** Recall that our inverted index in HW5 holds entries that contain information about (MinimalTuple, NTrgms in the tuple, and a Trgm in the tuple). We have one index entry for every trigram in every inner tuple. What is the lexicographic sorting order for this index?
 - (MinimalTuple, NTrgms, Trgm)
 - (MinimalTuple, Trgm, NTrgms)
 - (Trgm, MinimalTuple, NTrgms)
 - (Trgm, NTrgms, MinimalTuple)
- [2 pts]** Now we want to add the “Weight” of the given Trgm into the index. We need to decide where we should insert it into the lexicographic ordering from (b). Select *all* valid positions for the “Weight” field in the lexicographic ordering (*there may be more than one valid answer—select all of them!*)
 - Index 0 – at the beginning
 - Index 1 – after the first key
 - Index 2 – after the second key
 - Index 3 – at the end

2. Let's say that instead of storing the MinimalTuple in the index, we decide to store the RID ("record id") which is a field in the MinimalTuple that can uniquely identify every tuple in a given table (i.e., a primary key).
- a) **[1 pt]** How does storing RID instead of MinimalTuple affect our memory usage in general?
 - A. It decreases our memory usage.
 - B. It increases our memory usage.
 - C. It does not change our memory usage.
 - b) **[1 pt]** Assuming the inverted index fits in memory, how does storing RID instead of MinimalTuple affect the number of IOs during the join?
 - A. It decreases the number of IOs.
 - B. It increases the number of IOs.
 - C. It does not change the number of IOs.
 - c) **[2 pts]** Assume the necessary changes have already been made to the inverted index. How will replacing MinimalTuple with RID change the similarity join algorithm?

*On the answer sheet, mark **N** if it will "Not require a change", mark **T** if it will require a "Trivial change" (i.e., just altering one line), or **C** if it will require a "Complicated change" (adding actual logic to the code).*

 - i. Trigramming the inner tuples.
 - ii. Storing the entries in the index.
 - iii. Finding the least tuple pointed to by all the outer trigrams.
 - iv. Concatenation of the tuples to be returned.

Logging and Recovery

After mastering the skills of CS186 you manage to implement your own database. But oh no, a bunny chewed through the power cable on your server and everything crashed! When you boot back up you see the following log records and transaction and dirty page table that were created at the last checkpoint.

Log

LSN	Record	prevLSN
10	T1 updates P3	null
20	T1 updates P1	10
30	T2 updates P2	null
40	T3 updates P1	null
50	Begin Checkpoint	-
60	T3 updates P3	40
70	T3 Aborts	60
80	End Checkpoint	-
90	CLR undo T3 LSN: 60	70
100	T1 updates P4	20
110	T1 commits	100
120	T1 Ends	110

Transaction Table

Transaction	Status	lastLSN
T1	running	10
T2	running	30
T3	running	40


Dirty Page Table

Page ID	recLSN
P1	40
P3	10

- [1 pt]** What is the undoNextLSN of the CLR at 90?
- [2 pts]** Fill in the Transaction and Dirty Page Table at the end of the analysis phase.
- [1 pt]** What log record will the redo phase start at?
- [2 pt]** What are the LSNs of log records to be redone in the redo phase?
- [2 pt]** What are the LSNs of log records that are read in the undo phase?
- [4 pts]** Write down the log records that will be written, and label them with the phase in which they're generated (*A* for *Analysis*, *R* for *Redo*, or *U* for *Undo*). For CLR's, make sure to add the undoNextLSN.

Concurrency Control

Consider the following schedule:

TIME 

T1	W(B)						R(E)		
T2		R(C)						W(B)	
T3				W(C)					W(E)
T4			R(E)		R(B)	W(B)			

9. [1 pt] True/False: The above schedule is conflict serializable.
10. [1 pt] True/False: There is an edge from T2 to T1 in the dependency graph.
11. [1 pt] True/False: There is an edge from T1 to T3 in the dependency graph.
12. [1 pt] True/False: There is an edge from T3 to T4 in the dependency graph.
13. [1 pt] True/False: There is an edge from T3 to T2 in the dependency graph.

Consider the sequence of operations in the box to the right:

14. [1 pt] Under two phase locking, what is the earliest step *after which* we could release our lock on C? Your answer should be a number (e.g., after step "12").
15. [1 pt] Under strict two phase locking, what is the earliest step *after which* we could release our lock on C? Your answer should again be a number (e.g., after step "12").
16. [1 pt] True/False: There are conflict serializable schedules that cannot occur under two phase locking.
17. [1 pt] True/False: There are conflict serializable schedules that cannot occur under **strict** two phase locking.
18. [1 pt] In the multi-granularity locking protocol we discussed in class, if I hold an X lock on one page of a table, and an S lock on another page of the same table, what lock (if any) do I have on the table?

- 1) Lock X(C)
- 2) Lock S(A)
- 3) Read(C)
- 4) Write(C)
- 5) Lock S(B)
- 6) Read(B)
- 7) Lock S(E)
- 8) Read(E)
- 9) Read(A)
- <end of transaction>

Query Processing and Optimization

Consider a student database schema as below, with primary keys underlined and foreign keys as listed:

STUDENTS (sid, sname, street, city, age, gender)

REGISTERED (sid, cid, credits)

foreign key sid references STUDENTS

foreign key cid references COURSES

COURSES (cid, cname, profname)

Assume the following statistics:

- The STUDENTS relation has 10,000 tuples spread evenly across 500 pages
- The REGISTERED relation has 50,000 tuples spread evenly across 1,000 pages
- The COURSES relation has 500 tuples; 40 tuples of COURSES fit in each page
- NKeys(city) for STUDENTS is 200 (i.e., there are 200 distinct values for city)
- NKeys(age) for STUDENTS is 80
- NKeys(gender) for STUDENTS is 2
- Low(age) for STUDENTS is 10
- High(age) for STUDENTS is 90
- NKeys(gender) for STUDENTS is 2

19. **[1 pt]** Consider a join of STUDENTS and REGISTERED. How many I/O's will a *Page-Oriented Nested Loop Join* take using STUDENTS as the outer relation? (You should ignore the cost of writing the final join answer out to disk.)

20. **[1 pt]** Again, consider joining STUDENTS and REGISTERED. Now assume that we have $B=52$ memory pages available for the join. Using the *Block (Chunk) Nested Loop Join* with REGISTERED as the outer, how many I/O's will be required? (You should ignore the cost of writing the final join answer out to disk.)

21. **[1 pt]** Consider the following query:

```
SELECT gender, COUNT(*) FROM STUDENTS GROUP BY gender
```

Calculate the total number of I/O's required to answer this query using *Hybrid Hash-based Grouping* using the (<Group Val, Trans Val>) approach described during lecture. Assume that the final output does not need to be written to disk.

Now consider the following indexes:

- **RegSC** is a *clustered* B+ Tree defined on composite key $\langle sid, cid \rangle$ for REGISTERED (NPages = 39,000)
- **StudC** is an *unclustered* B+ Tree index is defined on the *city* attribute for STUDENTS (NPages = 600)
- **StudA** is a *clustered* B+ Tree index is defined on the *age* attribute for STUDENTS (NPages = 600)

Each of the indexes is **Alternative 2** and can be assumed to have a **height of 3** (each root-to-leaf path is 3 pages). Also assume that for each index, 50 index entries fit in one index page

For the next three questions, you need to fill in (a) the expected number of output tuples according to the System R approach, (b) an *efficient method* (file scan, index lookup, etc) for answering this query *and the names of any indexes* you are using, and (c) the number of disk accesses required. If you feel the need, you can write down any assumptions you are making in your answer (but be extremely brief!)

22. [2 pts] `SELECT * FROM REGISTERED WHERE cid = "CS186";`

- Estimated # Tuples in Answer:
- Method and access path used:
- Estimated # I/O's required:
- Assumptions:

23. [2 pts] `SELECT * FROM REGISTERED WHERE sid = "01234567";`

- Estimated # Tuples in Answer:
- Method and access path used:
- Estimated # I/O's required:
- Assumptions:

24. [2 pts] `SELECT * FROM STUDENTS WHERE city = 'Berkeley' AND age > 50;`

- Estimated # Tuples in Answer:
- Method and access path used:
- Estimated # I/O's required:
- Assumptions:

Parallelism

Imagine that the CS186 staff—5 TAs and 1 Professor—has to grade 300 exams with 6 questions per exam. Each question was written by a different Member Of Staff (MOS), and must be graded by that MOS: no MOS can grade a question written by another MOS!

Circle Grading. The staff sits in a circle to grade. Exams are partitioned into 6 random piles of 50 exams each, and each MOS gets one pile. The grading process works as follows: each staff member *removes* an exam from the top of their pile, works on it for exactly 1 minute and writes down a score for their question. They then instantly *slide* the exam to the bottom of the pile belonging to the MOS on their right. (Assume that *remove* and *slide* can happen concurrently and instantaneously). When a MOS sees an exam with their own question already scored, they finish.

25. [1 pt] How many minutes pass in the Circle Grading Scheme before the last MOS finishes?

FedEx Grading. Imagine the following grading scheme. The professor leaves town to participate in a 2-day "research retreat" at Lake Tahoe. The 5 TAs go home to 5 different cities (none of them Berkeley) for summer break. The Professor tells each TA to take home 60 exams, use FedEx to pass the exams among them as needed, and then send the exams back to him in Berkeley. Assume that FedEx accepts packages until 5PM, and delivers the next day by 9AM for a fixed cost of \$10 per package, regardless of size. Assume also that TAs have nothing better to do on a summer day than grade exams, at the rate of exactly 1 question per exam per minute. Finally, recall that the final results have to arrive in Berkeley at the professor's office so he can grade his question at the end.

26. [2 pts] Consider optimizing the FedEx Grading Scheme to minimize time to completion. How many days does it take from the start until the Prof. gets the exams?

27. [2 pts] Now consider optimizing the Distributed Grading Scheme to minimize FedEx cost. How much money would be required to complete the grading process at minimum?

28. [1 pt] Is it possible to optimize both time-to-completion and FedEx cost?

- A. Yes, because the time-optimal algorithm minimizes communication as well.
- B. Yes, in the average case analysis, but No in the worst case analysis.
- C. No, because of inherent tradeoffs between latency and bandwidth.
- D. No, because minimizing FedEx cost requires time to form bigger batches.

Sorting. After grading, the exams must be sorted by (Lastname, Firstname). This will take time. Assuming the MOS are all in one place, there are two proposals for how to proceed:

TED's PROPOSAL: "Let's range-partition by LastName. We create 6 "bins" of exams: one bin each for LastNames beginning with [A-D], [E-H], [I-L], [M-P], [Q-T], and [W-Z]. In the first phase we *shuffle*—go through our subpiles and drop exams into the appropriate bins. In the second phase, each of us sorts one bin independently, and then we just *concatenate* the results".

JIM's PROPOSAL: "Let's each just sort the subpile we have. We can then *merge* our sorted runs".

Assume that *shuffle* and *merge* take the same amount of time, and *concatenate* takes zero time. Assume also that each MOS sorts at the same speed, $O(n \log n)$ in the number of exams.

29. [1 pt] Which proposal is more efficient: Ted's or Jim's? Explain your answer in *five words or less*. (Over-long answers may be penalized!)

SQL

It is almost time for the Summer Olympics, during which stressed-out athletes get tested and scored while we sit on the couch. That will be nice.

In this question we assume a very simple database for scoring athletes in their Olympic events, with primary keys underlined, and integrity constraints as noted:

Athletes(sid integer, name text, country text)

Events(eid integer, sid integer, score float, ename text)

sid foreign key to Athletes.

score ranges from 0 to 10

score NOT NULL

30. [6 pts] For each query (a)-(f), write down the letter of the corresponding English description on the bottom (A)-(F). Each query matches only one description, but it is possible for each description to correspond to multiple queries (or none!)

a)

```
SELECT DISTINCT A.sid
  FROM Athletes A, Events E
 WHERE E.sid = A.sid
      AND E.score < 10;
```

d)

```
SELECT A.sid
  FROM Athletes A
 WHERE 1 >=
      (SELECT COUNT(*)
       FROM Events E
       WHERE E.sid = A.sid
           AND E.score = 10);
```

b)

```
SELECT A.sid
  FROM Athletes A
 WHERE A.sid NOT IN (
   SELECT E.sid
     FROM Events E
    WHERE E.score < 10);
```

e)

```
SELECT DISTINCT A.sid
  FROM Athletes A
 WHERE A.sid NOT IN (
   SELECT E.sid
     FROM Events E
    WHERE E.score = 10);
```

c)

```
SELECT DISTINCT A.sid
  FROM Athletes A, Events E
 WHERE E.score = 10;
```

f)

```
SELECT A.sid
  FROM Athletes A, Events E
 WHERE A.sid = E.sid
 GROUP BY A.sid
 HAVING AVG(E.score) = 10;
```

- A. Athletes who got no 10 scores
- B. Athletes who got at least one 10 score
- C. Athletes who got at least one score less than 10
- D. Athletes who got "straight 10's": i.e., a 10 score in every event they competed in
- E. Athletes who got at most one 10 score
- F. Athletes who competed in every event.

Database Design

GSL (GOMTV Global Starcraft II League) three-time champion Lim Jae Duk has contacted you to ask for help in normalizing the GSL's database of 1v1 Starcraft II matches. He provides you the following CREATE TABLE statement, as well as the following additional functional dependencies.

SQL CREATE TABLE Statement:

```
CREATE TABLE GSL
(match_datetime timestamp, -- D
 match_id, int PRIMARY KEY, -- M
 player1_name text, -- N1
 player2_name text, -- N2
 player1_gamertag text, -- G1
 player2_gamertag text, -- G2
 player1_charid int, -- C1
 player2_charid int, -- C2
 commentator_name text, -- E
 commentator_rating int); -- R
```

Additional Functional Dependencies:

$M \rightarrow D$
 $E \rightarrow R$
 $N_1 \rightarrow G_1, C_1$
 $N_2 \rightarrow G_2, C_2$
 $G_1, C_1 \rightarrow N_1$
 $G_2, C_2 \rightarrow N_2$

31. [2 pts] Decompose the above table into BCNF using the functional dependencies in order, considering the PRIMARY KEY as the first functional dependency. In your answer show only the tables that result from the decomposition.

32. [1 pt] True/False: This decomposition is lossless join.

Lim Jae Duk has also requested an ER diagram of the decomposed table. He has translated the above CREATE TABLE declaration and functional dependencies into ER-diagram-speak. He provides you with the following translation:

- Each match participates in the GSL relationship at most once, with 2 Players and 1 Commentator.
- *id* determines *datetime* for Match.
- A Commentator's *name* determines his or her *rating*.
- A Player's *name* determines his or her Character's *charid* and *gamertag*, if that player has a Character.
- A Character's *gamertag* and *charid* determines the name of the Player, if that Character has a Player.

33. [4 pts] Complete the ER diagram on the answer sheet to reflect the translation above. The only things you need to do are (i) add lines & arrows and (ii) add underlining to indicate primary keys.

Name: _____

Class Account:
cs186-_____

Similarity Joins

1a. Line # _____ 1b. (A-D) _____ 1c. (A-D) _____

2a. (A-C) _____ 2b. (A-C) _____ 2c.) i. _____ ii. _____ iii. _____ iv. _____

Logging and Recovery

3. undoNextLSN= _____

4.

Transaction	Status	lastLSN

Page ID	recLSN

5. LSN: _____

6. LSNS: _____

7. LSNS: _____

8.

Phase	LSN	Record	prevLSN	undoNextLSN

Name: _____

Class Account:

cs186-_____

Concurrency Control

9.

T F

10.

T F

11.

T F

12.

T F

13.

T F

14. After step _____

15. After step _____

16.

T F

17.

T F

18. Lock: _____

Query Processing and Optimization

19. _____ I/Os

20. _____ I/Os

21. _____ I/Os

22a. _____ tuples

22b. _____

22c. _____ I/Os

23a. _____ tuples

23b. _____

23c. _____ I/Os

24a. _____ tuples

24b. _____

24c. _____ I/Os

Name: _____

Class Account:

cs186-_____

Parallelism

25. _____ minutes

26. _____ days

27. \$ _____

28. _____

29. _____

SQL

30a. (A-F) _____

30b. (A-F) _____

30c. (A-F) _____

30d. (A-F) _____

30e. (A-F) _____

30f. (A-F) _____

Database Design

31. _____

32.

T F

Name:

Class Account:
cs186-_____

33.

