UCLA Computer Science Department

# CS143 MIDTERM EXAM:
# Closed Book, 110 Minutes

Winter 2019

**Problem A:** On the relation **Taken(StID, CourseID, Grade)** describing the courses taken by each student write the following queries without using more than one level of nested queries.

- A.1 Write an SQL query to find the StID of all students who completed at least three classes with a grade B- or better. (The code for grades is numeric and the grade for B- is 2.7. Finally, you can assume that students only take a course once.)

```
SELECT StID FROM Taken AS S1
WHERE S1.Grade >= 2.7
GROUP BY S1.StID HAVING count(CourseID) >= 3.
```

**Taken(StID, CourseID, Grade)**

A.2 Write an SQL query to find the StID of every student who (i) completed at least three classes, and (ii) took a grade of B- or better in all classes he/she completed:

```
SELECT StID
   FROM Taken AS T1
   WHERE T1.StID NOT IN (SELECT T2.StID
                             FROM Taken AS T2
                              WHERE T2.Grade < 2.7)
  GROUP BY S1.StID HAVING count(S1.CourseID) >= 3
```

**Taken(StID, CourseID, Grade)**

A.3 Express query A.1 in relational algebra, using only select, join, project, and cartesian product (thus no division, and no aggregates). You can assume that CourseID is numeric. To simplify your query you can use assignments:=

*Answer:*

$$Tk1 := Tk0 := \Pi_{StID,CourseID}(\sigma_{Grade \geq 2.7}(Taken))$$

$\bowtie$

$$Tk2 := \Pi_{Tk1.StID,Tk1.CourseID}(Tk1 \bowtie_{Tk1.StID=Tk0.Stid \wedge Tk1.CourseID>Tk0.CourseID} Tk0)$$

$$\textit{Result} := \Pi_{Tk2.StID}(Tk2 \bowtie_{Tk1.StID=Tk2.Stid \wedge Tk2.CourseID>Tk1.CourseID} Tk1)$$

**Problem B** On the relation `Taken(StID,CourseID,Grade)` we have one million tuples and the length of each tuple is 50 bytes. The tuples are stored as fixed-length unspanned records in a file consisting of blocks of size 2048 bytes.

On this file, we build a **sparse** index on `StID`. The index is organized as a B+ tree, where each key takes 18 bytes and each pointer takes 22 bytes (the leaf nodes are chained together as in the textbook). The B+ tree blocks contain 2048 bytes.

B.1 How many blocks does the file use if there remains no empty record slot in the file blocks.

$2048/50 = 40$ *records per block*. $10^6/40 = 25000$.

B.2 Compute the blocks used at each level of the B+ tree, for the best case and the worst case.

*N=51 pointers (50 at the leaf level.) Worst case: 25 at bottom level, 26 at other levels*

B+ tree, best case:

*Leaf level: sparse index. One pointer per block:* 25.000/50 = 500

*Next Level: 10*

*The root:1*

B+ tree, worst Case:

*Leaf nodes:* 25.000/25 = 1000

*Next Level:* 1000/26 = 38.4 *take the FLOOR (not the ceiling). 38!*

*Next Level:* 38/26. *Cannot split in two blocks. This is the root!*

B.3 Is this index a primary index or a secondary one?

*A sparse index can only be built on clustered data. Thus this is a primary index.*

B.3 Is this index a primary index or a secondary one?
*A sparse index can only be built on clustered data. Thus this is a primary index.*

B.4 A query asking for the records of all students whose StID falls in a certain range returns 800 records. Estimate the number of pages accessed to retrieve this query.

*Primary indexes are clustered. The file records must be sorted on the key. 800 take 20–22 blocks. plus 3 blocks of the index, for a total of: 3+22=25.*

**Problem C** Let us use again the table **Taken(StID, CourseID, Grade)** of Problem B.
C.1 Write the SQL query that returns the StIDs for pairs of students who took the same course getting the same grade.

```
SELECT  T1.StID, T2.StID
        FROM Taken AS T1 T2
        WHERE T1.CourseID=T2.CourseID and
        T1.Grade=T2.Grade
```

C.2 To implement this SQL query the system optimizer much select one these two techniques: (i) an index join using the sparse index on **StID**, or (ii) a block nested-loop join. Which technique will the query optimizer select, and how many block-retrievals will occur using your chosen technique if only two blocks of main memory are available for the join (of course, a third block will be used for output)? Show the formulas you use.

***The StID index is of no use in joining on Course.ID and Grade. Thus the optimizer will select a block nested-loop join. The usual formula is n+ n*n. But for self-join is n+ n * (n − 1)= n*n, because for each block of the first file we do not need to reload it from the second file. Thus n × n, i.e. 25000 × 25000 = 625 × $10^{6.}$***

**Problem C.3** The query optimizer must also predict the number of tuples returned by the answer. The optimizer makes its prediction by treating this self-join as any other join and using the information that there are 1000 different courses and 13 different grades and these occur in every possible combination.
The optimizer has no other information available. Compute the number of tuples that, by the optimizer prediction, will be in the answer and also show the formulas you use.

***Answer**: How many combinations (CourseID, Grade): 1000*13= 13000.*
*So each group sharing the same value has size $10^6/13000= 10^3/13$.*
*In the self join each group produces $(10^3/13)*(10^3/13)$ results and*
*There are 13000 of this groups for a total of*
*$13*1000*(10^3/13)*(10^3/13)= 10^6*10^3/13 = 10^9/13$*
*So the estimated size of the join is $10^9/13 = 10^{12}/(13*1000)$*
*Thus if descrease (rsp. Increase) the number of grades the estimate increases (resp. descreases).*

- 4. (4 points)We want to execute the query **SELECT R.A,R.B**
  **FROM R,S**
  **WHERE R.B=S.B"**

Say that we use the B+tree constructed in the previous problems. Since we do not need to return S.C, we decide to use the following algorithm to avoid scanning the S table: For each tuple r in R Lookup S.B index with r.B.If r.B exists in the index, return (R.A, R.B)

1. You must Compute the cost of this join. In computing your answer, please use the main memory buffers in the most efficient way to minimize the number of disk IOs. Do not include the cost for writing the final answer. Assume that neither the index nodes nor the R blocks are cached in main memory in the beginning.

- ANSWER:. The B+tree index is 21 blocks in two levels. With 22 blocks of main memory we can cache the B+

In main memory and then load the R pages one by one and see it the matching value is in the index.

With three blocks we might have to reload an index block for each record in R.