6.830 Quiz 1 Review Session

Rebecca Taft

10/16/14

Logistics

- Exam is in class on Monday ☺
- 75 minutes
- Open notes; laptops allowed but no Internet

Topics on the Quiz

- Database history: IMS, Codasyl
- Schema design: BCNF, Entity-Relationship diagrams
- Query optimization: Access methods, join algorithms, join ordering, cost analysis
- Database internals: Buffer pool, iterator model for operators, index structures
- Column stores: Database layout for analytic workloads
- Physical database design: Choice of best indexes and materialized views

Tips for Studying

- Read through the lecture notes (your own and the ones posted on the website)
- Do the practice exams and check your answers. Make sure you understand the solution.
- Review the problem sets and solutions (PS2 solution to be posted soon!)
- Review the high-level concepts of each lab
- Review the readings
- Ask questions on Piazza

Review: Access methods

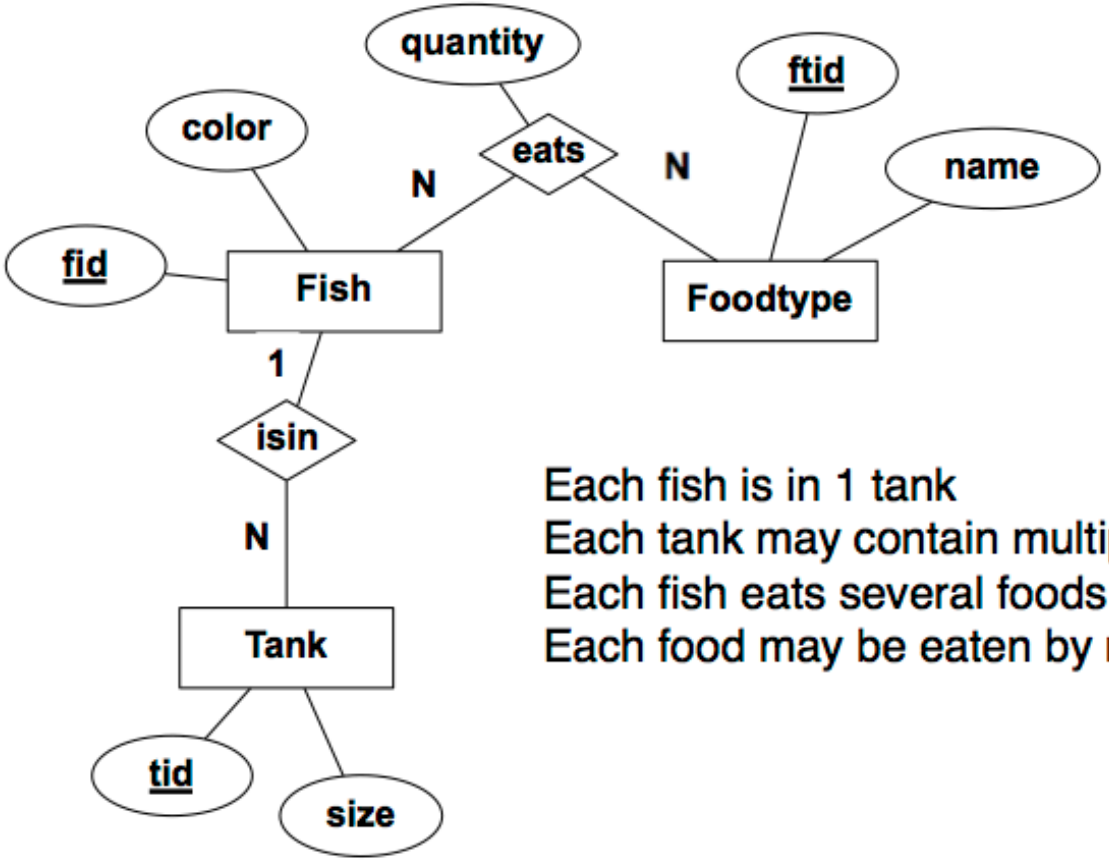
Access Method	Key Features
Heap file	• Records are unsorted • Search for records by sequentially scanning the entire file • Use if there are no available indexes on your search key or you expect to return a large number of records
Hash index	• Typically points to an unsorted underlying heap file • Constant time search for records • Useful for finding a set of specific keys, *not* searching for ranges of keys • May not be worth using if you have to perform random I/O to access a large number of records in the underlying heap file
B+ tree index	• Typically points to an unsorted underlying heap file • Logarithmic time search for records ($\log_B n$) • Useful for finding a set of specific keys or scanning a range of keys • May not be worth using if you have to perform random I/O to access a large number of records in the underlying heap file
Clustered index	• Records in underlying file are sorted, eliminating need for random I/O • Constant or logarithmic search • Useful for finding a set of specific keys or scanning a range of keys • Could be used as input to sort-merge join, to avoid sort step • Can have multiple indexes per table but only one clustered index!

Review: Join algorithms

Join algorithm	Key Features
Nested loops	• O(nm), where n is tuples in outer, m inner • Only useful if the inner relation is very small, and therefore the overhead of building a hash table is not worth it • *Block nested loops*: Can operate on blocks of tuples of inner relation, to make more efficient; complexity is then (nB), where B is number of blocks
Index nested loops	• Only possible if you have an index on the inner relation • Efficient if the number of lookups you need to do on the index is small
In-memory hash	• If one of the tables can fit in memory, can create an hash table on it on the fly • Pipeline lookups from other table (which may not fit in memory) • Good choice for equality joins when there is no index
Simple hash	• Good choice if one of the tables almost fits in memory • I/O cost is P(|R| + |S|), where P is the number of partitions you split each relation into. Each partition P must fit in memory • |R| and |S| are the number of pages in relations R and S • Always better to use Grace hash if P > 2
Grace hash	• Usually the best choice if neither relation can fit in memory • I/O cost is 3(|R| + |S|)
Sort-merge hash	• Same I/O cost as Grace hash, but less efficient due to cost of sorting • Could be a good choice if the relations are already sorted or you will need the output to be sorted on the join attribute later in the query plan (e.g., ORDER BY)

Example Problem 1

You are given the following ER-diagram for a database that stores information about a fish store. Here the labels 1 and N on edges of the diagram indicate whether there is 1 entity or many (N) entities participating in a relationship.



Each fish is in 1 tank
Each tank may contain multiple fish
Each fish eats several foods
Each food may be eaten by many fish

4. [8 points]: In addition to the functional dependencies suggested by the statements in the lower right of the diagram, assume that every attribute is functionally dependent on the key of the entity in which it appears. Write a collection of BCNF relations corresponding to this diagram.

Example Problem 2

You are given the following schema and SQL query:

```
dept (did int primary key, bldg int, campus int)  // 12 bytes per record
hobby (hid int primary key, hname char(17), cost int) //25 bytes per record
emp (eid int primary key, ename char(17), d int references dept.did) //25 bytes per record
hobbies (e int references emp.eid, h int references hobby.hid) //8 bytes per record

SELECT ename,bldg,SUM(cost)
FROM emp,dept,hobbies,hobby
WHERE emp.d = did
AND hobbies.e = eid
AND hobbies.h = hid
AND ename LIKE '%Sam%'
GROUP BY ename,bldg
```

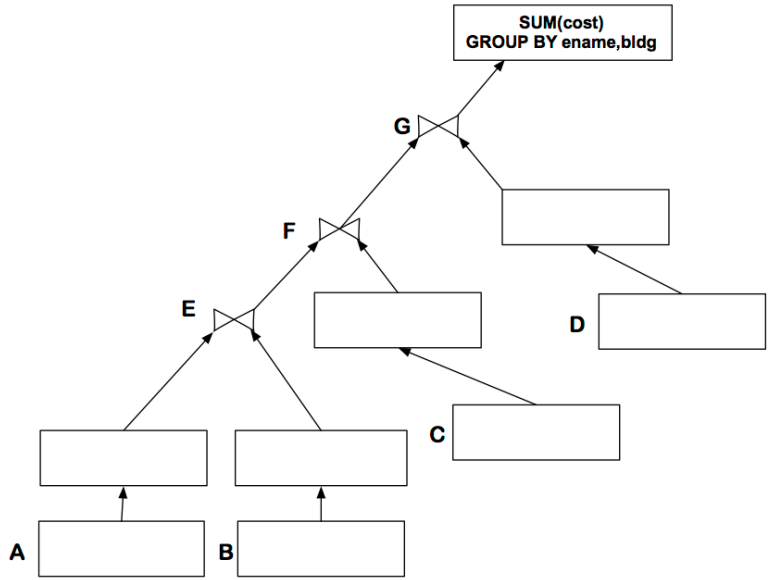
You are given the following statistics (here, $|A|$ denotes the number of tuples in relation A, and $S(e)$ denotes the selectivity of expression e).

Statistic	Value
$\|emp\|$	10^5
$\|dept\|$	10^3
$\|hobby\|$	10^3
$\|hobbies\|$	2×10^5
S(ename LIKE '%Sam%')	.1

Assume an integer is 4 bytes and a character is 1 byte, and that a disk page is 1000 bytes. Suppose you are running in a system with 100 pages of memory. Assume that each employee has about the same number of hobbies, and that hobbies and departments are assigned to employees uniformly and at random.

Example Problem 2 Cont

5. [10 points]: For now, assume that each join is a nested loops join and that there are no indices and no projection operations. In the diagram below, in the boxes labeled A, B, C, and D, write the names of the relations in the optimal left-deep join order for this plan. You should place the relation in the outer loop of the join in the leftmost box. Also indicate where the filters from the query should be placed. Some boxes may be empty. You should assume that the 100 pages of memory are optimally allocated between the joins to minimize the amount of I/O required by the plan.



6. [6 points]: Estimate the total number of disk pages read by the plan you drew above (do not worry about seeks for this problem).

7. [6 points]: Estimate the number of tuples produced by the plan you drew above.

Example Problem 2 Cont

8. [6 points]: Suppose that the system runs for awhile and the database grows to have 10 times as much data in every table.

Which of the following would likely improve the runtime of the above query on this larger database by more than a factor of 2, assuming its total runtime is dominated by I/O, that seeks take about 100× longer than the time to read a page from disk, and that the optimizer makes optimal decisions about join ordering and index selection.

(Circle ALL that apply.)

A. Create an unclustered B+Tree on `emp.ename`

B. Use a Grace Hash Join for the join between employees and departments

C. Use a Grace Hash Join for the join between employees and hobbies

D. Create an unclustered hash index on `emp.ename`

Example Problem 1 Answer

4. [8 points]: In addition to the functional dependencies suggested by the statements in the lower right of the diagram, assume that every attribute is functionally dependent on the key of the entity in which it appears. Write a collection of BCNF relations corresponding to this diagram.

(Write your answer in the space below.)

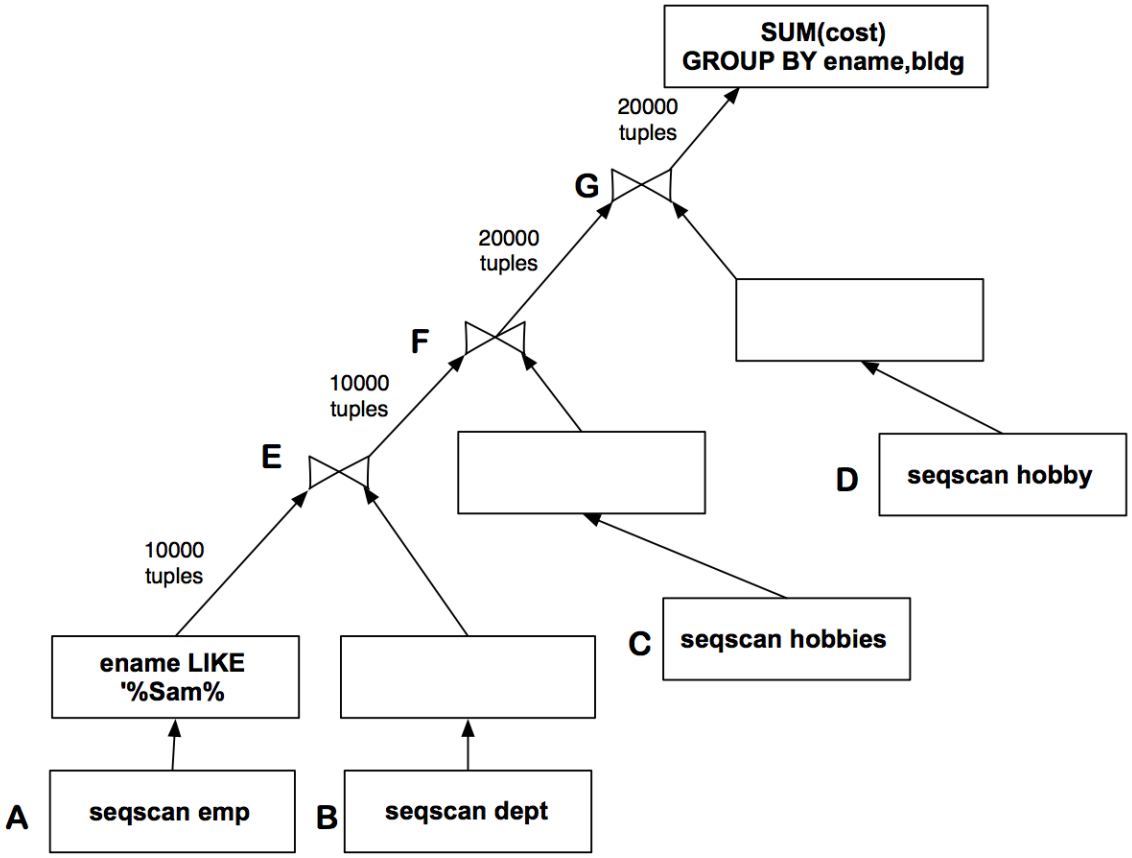
Answer:

fish: (fid, color, tank refs tank.tid)
fish_eats: (fid, ftid, quantity)
foodtype: (ftid, name)
tank: (tid, size)

Note that fish-tid is a many to one dependency, therefore we don't need an extra table for it.

Example Problem 2 Answer

5. [10 points]: For now, assume that each join is a nested loops join and that there are no indices and no projection operations. In the diagram below, in the boxes labeled A, B, C, and D, write the names of the relations in the optimal left-deep join order for this plan. You should place the relation in the outer loop of the join in the leftmost box. Also indicate where the filters from the query should be placed. Some boxes may be empty. You should assume that the 100 pages of memory are optimally allocated between the joins to minimize the amount of I/O required by the plan.



Example Problem 2 Answer Cont.

Answer:

The hobby and dept tables can both fit into RAM. The emp and hobbies do not. By doing the emp-dept join first, with emp as the outer, we are able to scan emp just once. We only scan dept once because it fits into memory. We have to do the join with hobbies next (because the join with hobby would be a cross product). Due to the constraints of left-deep plans, we must put the hobbies table on the inner, requiring us to scan it once for each tuple output by the emp/dept join (10^4 times). Finally, we can do the join with hobby; since hobby fits into memory, we only scan it once as well.

The following table summarizes the pages used by each of the tables:

Table	Formula	No. pages
emp	$10^5/(1000/25)$	2500
dept	$10^3/(1000/12)$	12
hobby	$10^3/(1000/25)$	25
hobbies	$2 \times 10^5/(1000/8)$	1600

Example Problem 2 Answer Cont.

6. [6 points]: Estimate the total number of disk pages read by the plan you drew above (do not worry about seeks for this problem).

(Write your answer in the space below.)

Answer: The total I/O cost is one scan of dept + one scan of hobby + one scan of emp + 10^4 scans of hobbies. Emp is 2500 pages, dept is 12 pages, and hobbies is 25 pages. Summing these numbers, we get: $1.6x10^7 + 2500 + 12 + 25 \approx 1.6x10^7$ pages

7. [6 points]: Estimate the number of tuples produced by the plan you drew above.

(Write your answer in the space below.)

The top-most join produces 20,000 records, but this represents only 10,000 distinct employees, so the GROUP BY will output 10,000 values (assuming employee name are unique.)

Example Problem 2 Answer Cont.

8. [6 points]: ~~Suppose that the system runs for awhile and the database grows to have 10 times as much data in every table.~~

Which of the following would likely improve the runtime of the above query on this larger database by more than a factor of 2, assuming its total runtime is dominated by I/O, that seeks take about 100× longer than the time to read a page from disk, and that the optimizer makes optimal decisions about join ordering and index selection.

(Circle ALL that apply.)

A. Create an unclustered B+Tree on `emp.ename`

B. Use a Grace Hash Join for the join between employees and departments

C. Use a Grace Hash Join for the join between employees and hobbies

D. Create an unclustered hash index on `emp.ename`

Answer: Indices won't help because the predicates are not very selective, and so indices will result in a lot of random I/O. Grace hash on employees and hobbies is a win because this join dominates the overall I/O cost as a nested loops, and grace hash will do far less I/O. Grace hash between employees and departments won't help because the departments table fits into RAM, so a nested loops join only scans both tables once, whereas a grace hash scans them multiple times.