CompSci 316 Fall 2016: Homework #4

100 points (8.75% of course grade) + 20 points extra credit
Assigned: Tuesday, November 15
Due: Thursday, December 1

This homework should be done in parts as soon as relevant topics are covered in lectures. If you wait until the last minute, you might be overwhelmed.

For Problems 1 and 4, you will need to use Gradiance. Access Gradiance via the "Gradiance" link on the course website. There is no need to turn in anything else for these problems; your scores will be tracked automatically.

For other problems, you will need to turn in the required files electronically. Please read the "Help → Submitting Non-Gradiance Work" section of the course website for instructions. When submitting your work, make sure you select the correct course and homework. Multiple submissions are okay, but please upload ***all*** required files in each resubmission.

The non-Gradiance problems in this homework does not require the course VM. You may prepare your answers electronically or on paper (handwritten). In the latter case, scan or photograph the pages and submit the resulting PDF (preferred) or JPG files. Please name your files informatively, e.g., as `n.pdf`, where n is the problem number.

Problem 1 (15 points)

Complete the Gradiance homework titled "Homework 4.1 (Indexes)."

Important note: These problems use a definition of "fan-out" that is different from what we discussed in class. When they say "fan-out $n = 3$" they mean that the maximum number of keys per node is 3, and the maximum number of pointers per node is 4 (i.e., "max fan-out is 4" in our terminology).

Problem 2 (20 points)

Consider a table R occupying 1,000,000 disk blocks. There is no index, and 501 memory blocks are available for query processing. Suppose you have the following two options to improve query performance:

1. Buy more memory to increase the number of available memory blocks to 1001.
2. Buy a faster disk to increase the speed of I/O by 20%.

(a) Suppose the objective is to speed up the query "`SELECT * FROM R ORDER BY A;`". Which option is more effective? Briefly justify your answer.
(b) Suppose the objective is to speed up the query "`SELECT * FROM R WHERE R.A > 100;`". Which option is more effective now? Briefly justify your answer.

Problem 3 (45 points)

Consider the following schema for an online bookstore:

Cust (*CustID*, *Name*, *Address*, *State*, *Zip*)
Book (*BookID*, *Title*, *Author*, *Price*, *Category*)
Order (*OrderID*, *CustID*, *BookID*, *ShipDate*)
Inventory (*BookID*, *Quantity*, *WarehouseID*, *ShelfLocation*)
Warehouse (*WarehouseID*, *State*)

Cust and *Book* represent customers and books, respectively. When a customer buys a book, a tuple is entered into *Order*. *Inventory* records the quantity and shelf location of each book for every warehouse. *Warehouse* records the state where each warehouse is located in. *Price* is numeric. *ShipDate* is an integer representation of a date. In the following, `:today` is a constant denoting the integer representation of today's date.

(a) Transform the following query into an equivalent query that 1) contains no cross products, and 2) performs projections and selections as early as possible. Represent your result as a relational algebra expression tree.

$\pi_{Title,Author}$
$\sigma_{(State="NC")\text{ and }(Author\ LIKE\ "\%Kondo")\text{ and }(ShipDate>:today-60)}$
$\sigma_{(Cust.CustID=Order.CustID)\text{ and }(Book.BookID=Order.BookID)}$
$(Cust \times Order \times Book)$.

Suppose we have the following statistics:

- $|Cust| = 3{,}000;\ |\pi_{State}Cust| = 50;$
- $|Book| = 1{,}000; |\pi_{Category}Book| = 10;$
- $|Order| = 60{,}000; |\pi_{BookID}Order| = 1{,}000; |\pi_{CustID}Order| = 3{,}000; |\pi_{ShipDate}Order| = 1{,}000;$
- $|Inventory| = 40{,}000; |\pi_{BookID}Inventory| = 1{,}000; |\pi_{WarehouseID}Inventory| = 50;$
- $|Warehouse| = 50; |\pi_{State}Warehouse| = 50.$

For each of the following relational algebra expressions, estimate the number of tuples it produces. Note that each estimation may build on the previous ones. You may make the same assumptions as in the lecture on query optimization. If you make different or additional assumptions, please state them explicitly.

(b) $\sigma_{ShipDate>:today-60}Order$.
(c) $\pi_{CustID}(\sigma_{ShipDate>:today-60}Order)$.
(d) $\sigma_{State="NC"}Customer$.
(e) $(\sigma_{State="NC"}Customer) \bowtie (\sigma_{ShipDate>:today-60}Order)$.

For the following questions, further suppose that:

- Each disk/memory block can hold up to 10 rows (from any table);
- All tables are stored compactly on disk (10 rows per block);
- 8 memory blocks are available for query processing.

(f) Suppose that there are no indexes available at all, and records are stored in no particular order. What is the best execution plan (in terms of number of I/O's performed) you can come up with for the query $\sigma_{ShipDate=:today}(Order \bowtie Inventory)$? Describe your plan and show the calculation of its I/O cost.

(g) Suppose there is a B+-tree primary index on *Order*(*OrderID*) and a B+-tree primary index on *Inventory*(*BookID*, *WarehouseID*), but no other indexes are available. Furthermore, assume that both B+-trees have a maximum fan-out of 100 for non-leaf nodes; each leaf stores 10 rows; and all nodes in both B+-trees are at maximum capacity except the two roots. What is the best plan for the same query in (f)? Again, describe your plan and show the calculation of its I/O cost.

Problem 4 (20 points)

Complete the Gradiance homework titled "Homework 4.4 (Concurrency Control and Recovery)."

Extra Credit Problem X1 (20 points)

Suppose a table *Visit*(*uid*, *timestamp*) tracks all user visits to your website. Assume that *uid* takes 4 bytes, and *timestamp* takes 4 bytes. All rows are stored unordered and compactly (with no gaps in between) on disk with 8k-byte blocks (i.e., 8192 bytes per block). There are 25.6 million rows in *Visit*, but you only have 10 blocks of memory available for data processing.

(a) Design the best algorithm you can come up with to find the most frequent visitor(s) to your website (as measured by the total number of visits). Note that there might be multiple such users (who tie for the top spot). Compute the number of I/O's your algorithm uses.

(b) Suppose that you care about the most frequent visitors only if they are really, really addicted to your site. Design the best algorithm you can come up with to find the most frequent visitors only if they each have at least 3,000 visits. If there are no such visitors, the algorithm can just say so. Compute the number of I/O's your algorithm uses.