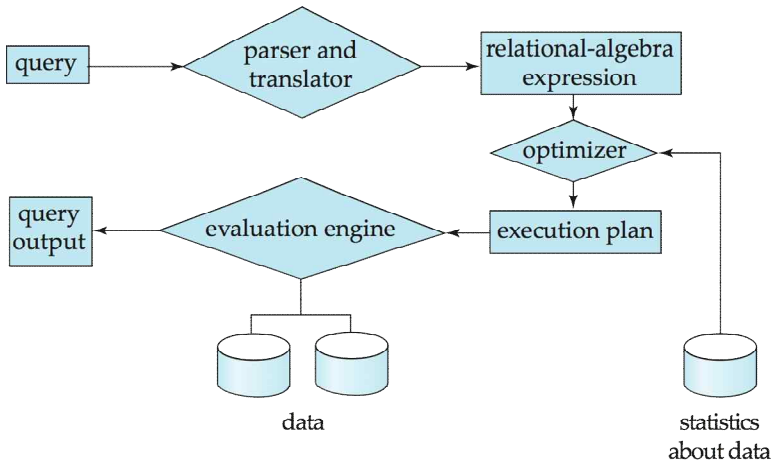
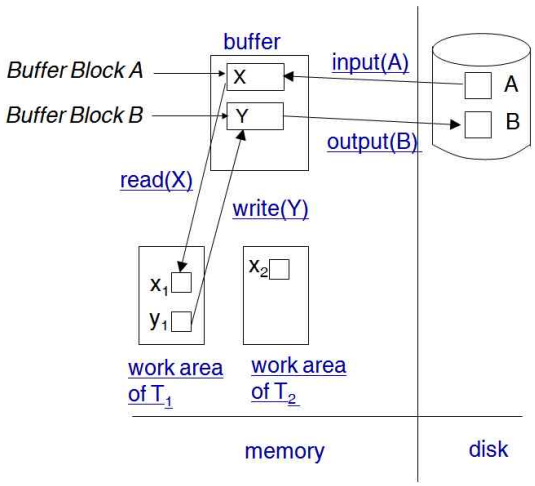
Final Exam

Instructor: Taewhi Lee
June 12, 2013

1. [5pts] Explain basic steps involved in *query processing*.
- Parsing and translation: checks syntax, verifies relations, and translate the query into its internal form
- Optimization: enumerate all possible query-evaluation plans, compute the cost for the plans, and pick up the plan having the minimum cost
- Evaluation: executes the query-evaluation plan taken from the optimizer, and returns the answers to the query.



2. (a) [5pts] Does each transaction read input data from the disk? Illustrate the process *of data access* with a diagram, including the concepts of *work area, buffer, disk,* and operations of *read, write, input, and output*.



(b) [5pts] Why do database systems choose different buffer replacement strategies from those of operating systems?
Queries have well-defined access patterns (such as sequential scans), and a DBMS can use the information in a user's query to predict future references. For example, LRU can be a bad strategy for certain access patterns involving repeated scans of data, when computing the join of relations r and s by nested loops

(c) [5pts] What is the data dictionary? Why do database systems keep data dictionary blocks in the main memory buffer?
The Data dictionary is system tables to store metadata; that is, data about data, such as Information about relations, indices, users, and statistical data.
The data dictionary information is needed to process each query, so it is frequently accessed.

3. [5pts] Describe the pros and cons of *cost-based query optimization* and *heuristic-based query optimization*.
Cost-based query optimization
- pros: finds the best execution plan with the lowest (estimated) cost
- cons: expensive because of too much search space
Heuristic-based query optimization
- pros: reduces search space
- cons: heuristics not always improves query performance, and do not find the best execution plan

4. [5pts] Transform the following query by applying the heuristics of pushing selections and projections.
instructor(id, name, dept_name, salary)
teaches(id, course_id, sec_id, semester, year)
course(course_id, title, dept_name, credits)

$\Pi_{name,\ title}(\sigma_{dept\_name=\text{"Music"} \wedge year=2009}((instructor \ltimes teaches) \bowtie \Pi_{course\_id,\ title}(course)))$

$\Pi_{name,title}(\Pi_{name,course\_id}(\sigma_{dept\_name=\text{"Music"}}(instructor) \bowtie \sigma_{year=2009}(teaches)) \bowtie \Pi_{course\_id,\ title}(course))$

5. Suppose the following statistics hold for relations r(A, B), s(B, C), and t(C, D).

$n_r = 5{,}000$ $n_s = 10{,}000$ $n_t = 8{,}000$
$f_r = 20$ $f_s = 25$ $f_t = 40$ $f_{r\bowtie s} = 10$ $f_{s\bowtie t} = 16$
V(B, r) = 2,500 V(B, s) = 5,000 V(C, s) = 1,000 V(C, t) = 8,000 (primary key)

(a) [5pts] Estimate the size (number of records) for r $\bowtie$ s and s $\bowtie$ t.

r $\ltimes$ s: $\min(\frac{n_r * n_s}{V(B,s)}, \frac{n_r * n_s}{V(B,r)}) = 5{,}000 * 10{,}000 / 5{,}000 = 10{,}000$

s $\ltimes$ t: 10,000 (C is the primary key of *t*)

(b) [10pts] Estimate the minimal number of block accesses (block transfers) required to compute r $\ltimes$ s $\bowtie$ t using *block nested-loop joins,* and describe the evaluation plan with the minimal cost. Only consider the join orders of ((r $\bowtie$ s) $\ltimes$ t) and (r $\ltimes$ (s $\bowtie$ t)). Assume that DB buffer can only hold one block of each relation. Include the cost of writing intermediate results to a disk, and ignore the cost of writing final results.
$b_r = 5{,}000 / 20 = 250$, $b_s = 10{,}000 / 25 = 400$, $b_t = 8{,}000 / 40 = 200$
$b_{r\bowtie s} = 10{,}000 / 10 = 1{,}000$, $b_{s\bowtie t} = 10{,}000 / 16 = 625$
Block-nested loop join cost (r $\bowtie$ s) = $b_r * b_s + b_r$ block transfers (+ $2 * b_r$ seeks)

s $\bowtie$ t = $b_s * b_t + b_t = 400 * 200 + 200 = 80{,}200$ (t as the outer relation)
writing intermediate results of (s $\bowtie$ t): 625
r $\bowtie$ (s $\ltimes$ t): $b_r * b_{s\bowtie t} + b_r = 250 * 625 + 250 = 156{,}500$ (r as the outer relation)
Total block accesses for r $\bowtie$ (s $\bowtie$ t): 80,200 + 625 + 156,500 = 237,325

c.f.) r $\bowtie$ s: $b_r * b_s + b_r = 250 * 400 + 250 = 100{,}250$ (r as the outer relation)
writing intermediate results of (r $\bowtie$ s): 1,000
(r $\bowtie$ s) $\ltimes$ t: $b_{r\bowtie s} * b_t + b_t = 1{,}000 * 200 + 200 = 200{,}200$ (t as the outer relation)
Total block accesses for (r $\ltimes$ s) $\bowtie$ t: 100,250 + 1,000 + 200,200 = 301,450

6. [10pts] Explain the four properties of transactions that should be maintained in DBMS.
- Atomicity: Either all operations of the transaction are properly reflected in the database or none are.
- Consistency: Execution of a transaction in isolation preserves the consistency of the database.
- Isolation: Although multiple transactions may execute concurrently, each transaction must be unaware of other concurrently executing transactions.
- Durability: After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

7. (a) [5pts] Is the following schedule for transactions T1, T2, and T3 *conflict serializable*? Justify.
No. Transforming the schedule to each of the serial schedules, <T1, T2, T3>, <T1, T3, T2> <T2, T1, T3>, <T2, T3, T1>, <T3, T1, T2>, and <T3, T2, T1> involves conflicting swaps, i.e., write(A) − read(A) or write(A) − write(A).

(b) [5pts] Is the following schedule *view serializable*? Justify.
Yes. It is view equivalent to a serial schedule <T1, T2, T3>.
In both schedules,
- T1 reads the initial value of A,
- T3 reads the value of A which produced by T2, and
- T3 performs the final write operation.

T1	T2	T3
read(A)		
	write(A)	
		read(A)
write(A)		
		write(A)

(c) [5pts] Explain the relationship and differences between conflict serializable schedules and view serializable schedules.
Every conflict serializable schedule is also view serializable.
Every view serializable schedule that is not conflict serializable has blind writes.

8. (a) [5pts] What is the *cascading rollback*? Why do we avoid it if possible?
A single transaction failure leads to a series of transaction rollbacks.
It can lead to the undoing of a significant amount of work.
(b) [5pts] Describe the *Two-phase locking protocol* (2PL).
- Phase 1 (Growing Phase): transaction may obtain locks and may not release locks
- Phase 2 (Shrinking Phase): transaction may release locks and may not obtain locks

(c) [5pts] What is the *strict 2PL*? Describe the advantages and disadvantages of using strict 2PL instead of (non-strict) 2PL.
In strict 2PL, a transaction must hold all its exclusive locks till it commits/aborts.
- Advantages: no cascading rollback
- Disadvantages: less concurrency than the original 2PL

9. (a) [5pts] Why is a checkpoint mechanism used for database recovery?
Redoing/undoing all transactions recorded in the log can be very slow.
- Processing the entire log is time-consuming if the system has run for a long time.
- We might unnecessarily redo transactions which have already output their updates to the database.

(b) [10pts] Suppose that your DBMS finds the following logs after a system crash. Describe in detail how recovery should proceed. Assume immediate-modification was used.

<T1 start>
<T1, A, 0, 10>
<T2 start>
<T2, B, 100, 200>
<T1, A, 10, 30>
<T3 start>
<T3, C, 50, 60>
<T2 commit>
<checkpoint {T1, T3}>
<T4 start>
<T4, D, 80, 70>
<T3, B, 200, 150>
<T3 commit>
<T1, C, 60, 40>

1) Redo phase
Find last checkpoint, <checkpoint {T1, T3}> record, and set undo-list L to {T1, T3}
Scan forward from above checkpoint record
<T4 start>:add T4 to undo-list → L = {T1, T3, T4}
<T4, D, 80, 70>, <T3, B, 200, 150>: redo it by writing 70, 150 to D, B respectively.
<T3 commit>: remove T3 from undo-list → L = {T1, T4}
<T1, C, 60, 40>: redo it by writing 40 to C

2) Undo phase
Scan log backwards from end
<T1, C, 60, 40>: undo by writing 60 to C, and write a log record <T1, C, 60>
<T3 commit>, <T3, B, 200, 150>: nothing to do
<T4, D, 80, 70>: undo by writing 80 to D, and write a log record <T4, D, 80>
<T4 start>: write a log record <T4 abort>, and remove T4 from undo-list → L = {T1}
<checkpoint {T1, T3}>, <T3 start>, <T3, C, 50, 60>, <T2 commit>: nothing to do
<T1, A, 10, 30>: undo by writing 10 to A, and write a log record <T1, A, 10>
<T2, B, 100, 200>, <T2 start>: nothing to do
<T1, A, 0, 10>: undo by writing 0 to A, and write a log record <T1, A, 0>
<T1 start>: write a log record <T1 abort>, and remove T1 from undo-list → L = Ø