CSE232A Final Exam - Winter 2004

Brief Directions:

- Write clearly: First, you don't want me to spend the whole week grading, do you? Second, it's good for you to write clearly!
- Open books, notes, even databases...
- You have 180 minutes to gain 172 points.
- Good luck!

Name:

Problem	Total	Awarded
Multiple Choice	24	
Midterm Problem Revisited	80	
SQL and Bag Algebra, I	20	
Concurrency and Recovery	18	
Concurrency Control	10	
Semantic Optimization	20	

1 Multiple Choice on Indexing and Query Processing (24)

State if the following statements are TRUE or FALSE. If the statement is false, briefly explain why. You win 3 points for each correct answer and lose 3 for each wrong answer.

1. The order of insertions into a B+ Tree will effect the tree's end structure.

2. B+ trees are more efficient than linear or extensible hashing for all types of queries.

3. Consider relations $R(A, B)$ and $S(B, C)$ where $T(R) = 5000$, $T(S) = 3000$, and B is a primary key of S. The number of tuples in $R \bowtie S$ has to be less than or equal to 3000.

4. Consider two relations $R(A, B, C)$ and $S(A, D, E)$, sharing a common attribute A. It is known that $R.A$ is a foreign key referencing $S.A$, and that $S.A$ is the primary key of relation S. Then the estimated size of $R \bowtie S$ is $T(R)$.

5. No matter how a relation is stored, it is possible to construct two separate sparse first level indexes on different keys.

6. No matter how a relation is stored, it is possible to construct two separate dense first level indexes on different keys.

7. No matter how a relation is stored, it is possible to construct a sparse first (lower) level index and a dense second (higher) level index. Both indices should be useful.

8. $\pi_S(E1 - E2) = \pi_S(E1) - \pi_S(E2)$, where E_1 and E_2 are arbitrary relational bag algebra expressions.

2 Midterm problem revisited (80 points)

Consider again the well known relations

Nation(**NationName**, **NKey**)
Customer(CName, NKey, **CKey**)
Order(Date, CKey, **OKey**)
LineItem(Product, OKey)

with the following characteristics (δ is the duplicate elimination operator)

- the bold attributes are keys of the corresponding relations (`NKey` and `NationName` are both keys of `Nation`)
- $\delta\pi_{\texttt{NKey}}\texttt{Nation} = \delta\pi_{\texttt{NKey}}\texttt{Customer}$
- $\delta\pi_{\texttt{CKey}}\texttt{Customer} = \delta\pi_{\texttt{CKey}}\texttt{Order}$
- $\delta\pi_{\texttt{OKey}}\texttt{Order} = \delta\pi_{\texttt{OKey}}\texttt{LineItem}$

The following statistics apply:

$$
\begin{aligned}
&T(\texttt{Nation}) = 5\\
&T(\texttt{Customer}) = 10^6\\
&T(\texttt{Order}) = 10^7\\
&T(\texttt{LineItem}) = 2 \times 10^7\\
&V(\mathit{LineItem}, \texttt{Product}) = 10^4
\end{aligned}
$$

All the relations are clustered and sorted along their primary key. Each block is of size 2000 bytes, the attributes `NKey`, `CKey`, and `OKey` are 4 bytes long, `NationName` and `Product` are 16 bytes long, and `CName` and `Date` are 12 bytes long. There are dense (i.e., all values of the indexed attribute appear in the index) B+ tree indices on `Customer.CKey` and `Order.OKey`.

Consider the query

```
SELECT *
FROM Nation, Customer, Order, LineItem
WHERE Nation.NationName = "Canada"
      AND Nation.NKey = Customer.NKey
      AND Customer.CKey = Order.CKey
      AND Order.OKey = LineItem.OKey
      AND LineItem.Product = "ABC123"
```

Consider an optimizer that produces plans with the following characteristics:

- no cartesian products appear in the plans
- selections of the form "attribute = constant" are applied directly on the relation that has the attribute, rather than on the result of a join that the relation participates into
- the joins are either Right-Index (RI) joins, which use the B+ tree index, or Sort-Merge joins
- the optimizer never builds a plan that creates an index
- the right argument of RI joins can only be a single relation that has the necessary B+ tree index. In our exercise this means that the only possible right arguments of RI joins are the `Customer` relation (which can be the right argument of an RI join on `Ckey`) and the `Order` relation (which can be the right argument of an RI join on `Okey`).

Do the following:

1. Produce all logical algebra plans that this optimizer will produce. Do not use join operators with more than two arguments.
2. Estimate the size of each subexpression of each plan.
3. Produce all physical algebra plans by annotating the joins of the logical plans. The two possible annotations are $\bowtie^{M}$, which stands for sort-merge join, and $\bowtie^{RI}$, which stands for RI join.
4. Cost the plans, by counting the number of disk accesses. Neglect the difference between accessing sequential and non-sequential blocks. Assume that each B+ tree access costs $1+B_R$, where B_R is the number of blocks that contain the result tuples. Justify the overall cost of each plan by showing the cost of each subexpression of your physical plan. Notice that you may reduce the amount of costing you have to do in the following way: If you have already found a plan p_1 whose cost is C_1, and the plan p_2 you currently work on costing has a subexpression whose cost is $C_2 > C_1$, you may declare this and stop costing any further.

If you need to make an extra assumption, please state it clearly. Extra assumptions should not contradict the ones that are provided by the problem statement.

3 Reducing SQL Queries to Bag Algebra, Part I (20 points)

Consider the database consisting of the following relations:

`Employee(name, department, salary)`

Write bag relational algebra expressions that compute the following queries. For aggregation you may use either the operators of the book or the operators in the notes:

1.
   ```
   SELECT department, SUM(salary)
   FROM Employee
   GROUPBY department
   ```

2.
   ```
   SELECT department, SUM(salary)
   FROM Employee
   GROUPBY department
   HAVING AVERAGE(salary)>100
   ```

3.
   ```
   SELECT department, SUM(salary)
   FROM Employee
   GROUPBY department
   HAVING AVERAGE(salary)>100 AND COUNT(*) > 10
   ```

4 Concurrency Control and Recovery (18 points)

Consider three transactions

- $T_1 : r_1(a), r_1(b), w_1(a), w_1(b), w_1(c), \textbf{commit}_1$
- $T_2 : r_2(c), r_2(b), w_2(b), \textbf{commit}_2$
- $T_3 : r_3(c), r_3(a), w_3(c), \textbf{commit}_3$

Each part below asks for a schedule of a particular type, for the actions of T_1, T_2, T_3 (and no other transactions). Please represent your schedules in the provided grid, with time flowing down the vertical axis and a separate column for each data value. Notice that there is also a "commit" column where you should place the **commit** actions. It may be the case that no schedule satisfies the requested properties. in such case write "NO SCHEDULE EXISTS". Use the following definitions from the class notes (there are slight discrepancies with the book definitions):

- A schedule S is recoverable if each transaction commits only after all transactions from which it read have committed.
- A schedule S avoids cascading rollback if each transaction may read only those values written by committed transactions.
- A schedule S is strict if each transaction may read and write only items previously written by committed transactions.

1. Write a 2PL schedule that is not recoverable.

a	b	c	**commit**

2. Write a 2PL and recoverable schedule that does not avoid cascading rollback.

a	b	c	**commit**

3. Write a strict 2PL schedule that is not serial.

a	b	c	**commit**

5 Concurrency Control (10 points)

Consider the following two transactions T_1 and T_2, as well as a third transaction T_3, whose list of actions you are free to choose.

$$T_1 : r_1(A) r_1(C) w_1(A)$$

$$T_2 : r_2(C) r_2(B) w_2(B)$$

Create a schedule that contains T_1, T_2 and T_3 such

- All actions of T_1 appear before the actions of T_2. That is, the schedule will look like

$$\ldots r_1(A) \ldots r_1(C) \ldots w_1(A) \ldots r_2(C) \ldots r_2(B) \ldots w_2(B) \ldots$$

- In every serial schedule that is conflict equivalent to S, T_2 precedes T_1 (the actions of T_2 will appear before the actions of T_1).

6 Semantic Optimization (20pts)

Let R be a relation with attributes ABCD. Consider the query

$$\pi_{AB}[\pi_{BCD}(R) \bowtie \pi_{ACD}(\sigma_{A=2}(R))] \bowtie \pi_{AD}(\sigma_{B=5}(R))$$

(i) (5 points) Construct the tableau corresponding to the query.

Answer:

(ii) (10 points) Minimize the tableau in (i) knowing that the query is only applied to databases satisfying the fd's

$$B \rightarrow D, \quad D \rightarrow C, \quad A \rightarrow B.$$

Result of chase: **Result of minimization:**

(iii) (5 points) Construct an select-project-join expression corresponding to the minimized tableau obtained in (ii).

Answer: